COMPILADOR DIDATICO

Angela Saade Pagani

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.)

Aprovada por:

Nelson Maculan Filho
(Presidente)

Sueli Mendes

José Lucas Mourão Rangel Netto

RIO DE JANEIRO, RJ - BRASIL
AGOSTO DE 1981

PAGANI, ANGELA SAADE

Compilador Didático [Rio de Janeiro] 1981.

IX, 408 p. 29,7cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1981)

Tese - Univ. Fed. Rio de Janeiro. Centro de Tecnologia.

1.Compiladores I.COPPE/UFRJ II.Título(série).

À Antonio Luiz e Madalena

AGRADECIMENTOS

As pessoas e entidades abaixo relacionadas contribuiram, de alguma forma, para a realização deste trabalho:

Antonio Luiz Pagani

Antonio Júlio Lossio Botelho

Nelson Maculan Filho

Agnello Jacob Saade

Niete Gimenes Saade

Madalena Saade Pagani

Olívia Fávaro Pagani

Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)

Banco de Desenvolvimento do Estado do Espírito Santo (BANDES)

Coordenação do Aperfeiçoamento de Pessoal de Nível Superior (CAPES)

ABSTRATO

O presente trabalho apresenta a implementação de um compilador para utilização em exercícios de curso de construção de compiladores.

A linguagem fonte é um sub-conjunto do PASCAL, representativo das linguagens de programação mais difundidas atualmente com relação a estrutura, comandos e dados, dando portanto uma visão realista do problema.

A modularidade na implementação visa facilitar a alteração do compilador ou da linguagem para desenvolvimento e avaliação de outras técnicas de compilação, permitindo um trabalho a nível de detalhe, sem perda da noção de conjunto.

ABSTRACT

The present work introduces the implementation of a compiler for use in exercises on compiler construction trainning courses.

The source language is a Pascal subset, which represents, among programming languages, one that is most difused as far as structure, commands and data are concerned, thus producing a more realistic view of the problem.

The modularity on implementation aims at reducing the difficulties concerned with alterations on the compiler or language, for development and evaluation of other compiling technics, allowing a work on a more detailed basis, without loosing the view of the whole.

I N D I C E

INTRODUÇÃO

O ideal num curso de construção de compiladores seria que cada aluno ou grupo de alunos pudesse construir seu compilador para ter uma experiência prática do conjunto e de todas as partes em separado.

Por questões de tempo, porém, isto normalmente não é feito. O que se tem então é um compilador para uma linguagem muito restrita, o que tem pouco a ver com a realidade, ou cada aluno ou grupo se ocupa de uma parte do mesmo, o que pode vir a impedir que cada um tenha noção do conjunto.

Para suprir esta deficiência seria desejável que se dispusesse de um compilador que possibilitasse aos alunos a alteração de quaisquer de suas partes usando técnicas diferentes de compilação para comparação dos métodos e também a introdução de alterações na linguagem a título de exercícios do curso.

Tal compilador poderia permitir que todos os alunos pudessem desenvolver e testar outras técnicas, sem necessidade de construir um compilador inteiro, tendo porém noção de todo o conjunto e podendo avaliar o desempenho de cada algorítmo e técnica empregadas.

O presente trabalho se propõe a apresentar uma implementação de um compilador para fins didáticos.

Para atingir seu objetivo, este deve ser o mais modular possível de modo a facilitar a substituição de seus módulos; deve ser programado em uma linguagem difundida com facilidade de acesso e compreensão; deve ter boa documentação e dar uma visão realista do problema; a linguagem fonte deve representar da maneira mais completa possível as linguagens mais usadas atualmente, com relação a estrutura, comandos e dados.

No capítulo 1 é apresentada a linguagem fonte, um subconjunto do PASCAL, com pequenas modificações.

O PASCAL é uma linguagem de programação de uso geral, que alia uma ampla flexibilidade para definição de estruturas de dados a um conjunto de comandos que atende de maneira bastante satisfatória às necessidades de manipulação dessas estrutu

ras, quais sejam, comandos de atribuição, transferência condicional e incondicional; comandos de iteração, subrotinas e expressões com operações aritméticas e lógicas. Manipula dados inteiros, reais, booleanos e alfanuméricos, sob forma de constantes, variáveis simples, variáveis indexadas (arrays) e conjuntos.

Estas características, juntamente com a organização em blocos da linguagem propiciam a fácil elaboração de programas bem estruturados, justificando portanto a escolha.

Uma vez que um dos objetivos deste trabalho é permitir que os alunos introduzam modificações e novos comandos na linguagem fonte, foi utilizado um subconjunto do Pascal.

Nesta simplificação algumas características particularmente interessantes da linguagem original foram sacrificadas, tais como:

- a possibilidade de definição de novos tipos de dados, derivados dos 4 tipos básicos, por meio da declaração TYPE;
- todos os tipos de dados estruturados, com exceção de ARRAY;
- devido à eliminação dos tipos de dados estruturados citada acima, também foram eliminados os comandos e as operações relativas aos mesmos (comandos WITH e operações com conjuntos);
- alocação dinâmica explícita de variáveis e por conseqüência o tipo de dado POINTER.

No capítulo 2 é apresentada a estrutura do compilador, no tocante às análises léxica e sintática, ao tratamento de erros, à geração de código e à estrutura de dados utilizada.

Para evitar a necessária definição de gramáticas de diferentes tipos, adequadas aos diversos métodos de análise sintática, o que acarretaria a criação de vários padrões de documentação, foi utilizado praticamente um único tipo de reconhecedor para todas as construções da linguagem fonte (autômato finito determinístico).

Quanto ao tratamento de erros, foi considerado compromisso deste trabalho e deteção de todos os erros léxicos, sintáticos e de contexto. Outros aspectos entretanto foram também

considerados, porém com menos ênfase: procurou-se fazer a detecção o mais cedo possível; foram feitas algumas tentativas de recuperação de erros e umas poucas correções.

O capítulo 3 é dedicado à programação do compilador. Consta da descrição das diretrizes gerais de implementação e da descrição de cada módulo, especificando sua função, as técnicas de compilação e os algorítmos empregados.

O capítulo 4 apresenta as conclusões do trabalho e sugestões para sua utilização.

Em anexo são apresentadas: nos apêndices A e B a BNF e os diagramas da linguagem original, como orientação e sugestão para modificações na linguagem; nos apêndices C e D a BNF e os diagramas do subconjunto utilizado; no apêndice E exemplos de programas PASCAL executados sob o CDP (Compilador Didático Pascal); nos apêndices F e G as listagens das Macros e dos programas fonte e no apêndice H a descrição do modo de uso do CDP.

1. LINGUAGEM

1.1. TIPOS DE DADOS

Cada dado do programa deve pertencer a um e somente um tipo. O tipo do dado define o conjunto finito e ordenado dos valores que o dado pode assumir.

Os dados são utilizados diretamente através das variáveis e constantes usadas no programa e podem ser associados a 4 tipos: REAL, INTEGER, BOOLEAN e CHAR.

1.1.1. REAL

O tipo REAL define um subconjunto enumerável de números reais com valores absolutos compreendidos entre 10^{-75} e 7237008×10^{69}. Portanto o valor absoluto de um dado real não pode ser menor que 10^{-75} e nem maior que 7237008×10^{69}.

Se B é definido com REAL, temos como valores válidos de B, por exemplo:

8.7	10.0	0.141
-14.19	-1.0	019.001

1.1.2. INTEGER

O tipo INTEGER define o subconjunto dos números inteiros com valores absolutos compreendidos entre 0 e 2147483647. Portanto o valor absoluto de um dado inteiro não pode ser maior que 2147483647.

Se B é definido como INTEGER, temos como valores validos de B, por exemplo:

0	-14186	1000
-2	329	0081

1.1.3. BOOLEAN

O tipo BOOLEAN define o conjunto dos dois valores "TRUE" e "FALSE". Portanto, um dado booleano só pode assumir um destes dois valores. O valor "FALSE" é definido

como menor que "TRUE".

1.1.4. CHAR

O tipo CHAR define o conjunto dos caracteres disponíveis nos equipamentos de entrada e saída.

A definição deste conjunto é estritamente dependente da implementação. No nosso caso, ele contém os seguintes caracteres:

- as letras
- os digitos
- o branco (ou espaço)
- os caracteres especiais[1]: . , ; ( ) : [ ] + - / * & | < > = " ¬ % '

que estão ordenados da seguinte maneira, em ordem crescente:

branco . < (+ | &] *) ;

¬ - / , % > : [' = "

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

Um dado do tipo CHAR pode ser composto por qualquer destes caracteres, ou uma cadeia deles.

Se B é definido como CHAR, temos como valores válidos de B, por exemplo:

A	123	BOLO
ABC	%	A=B1

1.2. ELEMENTOS BÁSICOS DA LINGUAGEM

Os elementos básicos da linguagem são os identificadores, as palavras chave , as constantes (literais, números reais e números inteiros) e os símbolos especiais (delimitadores e operadores).

---

1) Na programação os símbolos [e] foram substituídos por # e !, embora neste texto sejam sempre usados os primeiros.

1.2.1. IDENTIFICADORES

Um identificador é uma sequência de no máximo oito letras ou digitos, iniciada sempre por uma letra. Os identificadores são usados para dar nome aos diversos objetos do programa, tais como: dados (como nome de constante ou variável); comandos (como rótulo); grupos de comandos (como nome de procedimento ou função) e palavras chaves da linguagem.

Identificadores válidos seriam por exemplo:

NOME	A	I14
CENTRO	ABC1	IDENT

1.2.2. PALAVRAS CHAVE

Uma palavra chave é um identificador que possui significado específico na linguagem. Toda palavra chave é também palavra reservada e portanto não pode ser usada pelo programador para dar nome aos objetos do seu programa.

São palavras chave da linguagem:

ABS	ARCTAN	ARRAY	BEGIN
BOOLEAN	CASE	CHAR	CHR
CONST	COS	DIV	DO
DOWNTO	ELSE	END	EOF
EOLN	EOPG	EXP	FALSE
FOR	FORWARD	FUNCTION	GOTO
IF	IN	INTEGER	LABEL
LN	MOD	NEW	ODD
OF	ORD	PRED	PROC
PROGRAM	REPEAT	ROUND	SIN
SQR	SQRT	SUCC	THEN
TO	TRUE	TRUNC	UNTIL
VAR	WHILE	READ	READD
READP	READPD	WRITE	WRITED
WRITELN	WRITELND	WRITEPG	WRITEPGD

1.2.3. LITERAIS

Um literal é uma constante do tipo CHAR . É qualquer sequência de no máximo 256 caracteres antecedida e sucedida por um delimitador de literais.

Os delimitadores de literais são os símbolos " (aspas) ou ' (apóstrofe).

O símbolo usado para iniciar o literal tem que ser o mesmo usado para encerrá-lo. Por este motivo quando delimitado por aspas não pode conte-las : (podendo porém conter apóstrofes) e vice-versa.

Exemplos de literais válidos:

```
"DATA = '8/12/78'"
'LINGUAGEM DE PROGRAMAÇÃO'
```

1.2.4. NÚMEROS INTEIROS E REAIS

Um número é uma sequência que comece por um dígito.

Um número inteiro é uma sequência que só contém dígitos.

Um número real se compõe de uma parte inteira seguida de uma parte fracionária e/ou uma parte exponencial. A parte fracionária se compõe de um ponto seguido por um ou mais dígitos. A parte exponencial se compõe da letra "E" seguida de um sinal opcional seguido de um ou mais dígitos.

Exemplos de números inteiros válidos:

```
8          12          -1234567
0          -1           88
```

Exemplos de números reais válidos:

```
 0.14          -008.001E-12
-8.9E+1          1E2
```

1.2.5. DELIMITADORES

Um delimitador é qualquer caráter pertencente ao conjunto dos delimitadores definido para a linguagem. Este conjunto contém os seguintes caracteres:

. , ; : ( ) [ ] := + - / * & | ¬ ¬= < > >= <= ' " % = e o branco

1.2.6. OPERADORES

Um operador é qualquer caráter pertencente ao conjunto dos operadores definido para a linguagem. Este conjunto está contido no de delimitadores e compreende os seguintes caracteres:

+ - / * & | ¬ ¬= >= <= = > <

dos quais:

+ - | são operadores de adição

* / & são operadores de produto

= ¬= >= <= < > são operadores relacionais e

¬= >= <= são chamados operadores duplos.

1.3. DECLARAÇÕES

Todos os objetos usados no programa fonte precisam ser definidos antes de sua utilização.

Com exceção dos números inteiros, números reais e literais que são auto-definidos, e dos delimitadores e das palavras-chave que são pré-definidos, todos os demais objetos são definidos por meio de comandos declarações.

Os comandos declarações dependem da natureza do objeto a definir existindo comandos para:

- Declaração de Rótulos;
- Declaração de Constantes;
- Declaração de Variáveis
- Declaração de Procedimentos e Funções

1.3.1. DECLARAÇÃO DE RÓTULOS

Qualquer comando pode ter um nome, que é seu label ou rótulo, possibilitando sua referência de outros pontos do programa.

A atribuição de um nome a um comando é feita precedendo-se o comando por um identificador de rótulo, seguido de ':' (dois pontos).

Formato da declaração de rótulo:

LABEL ident1 , ident2 , ... , identn ;

onde:

ident1, ident2, ..., identn são identificadores que devem aparecer no programa como rótulos de comandos:

Exemplos de declarações de rótulos:

```
LABEL    FIM , CÁLCULO ;
LABEL    PARTE ;
```

1.3.2. DECLARAÇÃO DE CONSTANTES

As constantes são os dados que se mantêm com o mesmo valor durante toda a execução do programa e podem pertencer a qualquer dos 4 tipos de dados já vistos.

A declaração de constantes especifica um identificador como sinônimo para um valor fixo, cujo tipo está implícito.

O uso de nomes para constantes ajuda na documentação e torna o programa mais legível no caso de constantes que representam valores codificados. Além disso, no caso de constantes que controlam a execução do programa, sua alteração fica simplificada, pois se

resume na substituição de seus valores apenas no comando de declaração da constante.

Formato da declaração de constantes:

```
CONST ident1 , ident2 , ..............., identn = valor ,
      identa , identb , ..............., identm = valor1 ;
```

onde:

ident1, ident2, ..., identn, identa, identb, ..., identm são identificadores;
valor e valor1 podem ser números (inteiros ou reais com ou sem sinal), nomes de constantes já declaradas (com ou sem sinal) ou literais.

Exemplos de declarações de constantes:

```
CONST NOTA = 8 , MEDIA = 5.0;
CONST A,B,C = 14 , D = A ;
CONST LIT = 'LISTAGEM' ;
CONST PI = 3.14159 ;
```

Obs.: Os identificadores TRUE e FALSE são pré-definidos na linguagem como identificadores das constantes booleanas "TRUE" e "FALSE", respectivamente.

1.3.3. DECLARAÇÃO DE VARIÁVEIS

As variáveis são os dados que terão seus valores alterados pela execução do programa. Podem pertencer a qualquer dos 4 tipos de dados já vistos, tendo como restrição a limitação de tamanho (1 caráter) para variável tipo CHAR.

Obs.: Literais com mais de um caráter são permitidos apenas no comando WRITE.

A declaração de uma variável especifica o tipo a que ela pertence.

Formato da declaração de variáveis:

VAR ident1 , ident2 , ... , identn : tipo ,

identa , identb , ... , identm : tipo1 ;

onde:

ident1, ident2, ..., identn, identa, identb, ..., identm são identificadores;
tipo e tipo1 são uma das palavras reservadas CHAR, BOOLEAN, INTEGER, REAL[1].

Exemplos de declarações de variáveis:

VAR UM , DOIS , TRÊS : INTEGER ;

VAR NOME : CHAR ;

VAR TIPO : INTEGER , SAI : BOOLEAN , MÉDIA : REAL ;

1.3.4. DECLARAÇÃO DE PROCEDIMENTOS E FUNÇÕES

O termo procedimento ou função identifica uma sequência de comandos que pode ser ativada de qualquer ponto do programa. Os objetos referenciados por estes comandos podem ser alterados a cada ativação (isso é válido apenas para as variáveis, procedimentos e funções). A declaração do procedimento(ou função) define a sequência de comandos associando-a a um identificador e relaciona os objetos que serão alterados na sua ativação, que são chamados de parâmetros formais do procedimento (ou função). Os objetos que substituem os parâmetros formais quando da ativação do prpcedimento (ou função) são chamados de argumentos ou parâmetros reais.

Os procedimentos diferem das funções nos seguintes aspectos:

- formato da declaração, visto a seguir;

---

[1] Foram tratadas aqui apenas as variáveis simples. Os arrays, apesar de declarados neste mesmo comando, serão descritos à frente, num tópico à parte.

- forma de ativação, descrita no tópico EXPRESSÕES, no caso de funções e no tópico COMANDOS, no caso de procedimentos;
- a execução de uma função resulta no cálculo de um valor, chamado valor da função, que substitui a referência à função, no ponto de sua ativação.

Formato da declaração de procedimento:

PROC nome (seção de parâmetros formais) ; bloco ;
ou
PROC nome ; bloco ;

Formato da declaração de função:

FUNCTION nome (seção de parâmetros formais) : tipo ; bloco ;
ou
FUNCTION nome : tipo ; bloco ;

onde:

nome é um identificador que dará nome ao procedimento ou função;
seção de parâmetros formais é a lista das declarações dos parâmetros;
bloco é o corpo do procedimento (ou função). Contém as declarações dos objetos locais (de uso exclusivo) do procedimento (ou função) e os comandos executáveis;
tipo é uma das palavras reservadas INTEGER, REAL, BOOLEAN ou CHAR e define o tipo do resultado da função.

Os parâmetros de um procedimento (ou função) podem ser variáveis simples, arrays, procedimentos e funções.

Declaração de parâmetros variáveis simples:

lista de identificadores : tipo ;
ou
VAR lista de identificadores : tipo ;

Declaração de parâmetros arrays:

lista de identificadores : tipo array ;
ou
VAR lista de identificadores : tipo array ;

Declaração de parâmetros procedimentos:

PROC lista de identificadores;

Declaração de parâmetros funções:

FUNCTION lista de identificadores : tipo ;

onde:

lista de identificadores é uma série de um ou mais identificadores separados por vírgula;
tipo é uma das palavras reservadas INTEGER, REAL, BOOLEAN ou CHAR e define o tipo dos identificadores da lista;
tipo array define as características do array, conforme descrito no tópico ARRAYS.

A seção de parâmetros formais pode conter estas declarações em qualquer ordem e repetidas quantas vezes forem necessárias.

As listas de parâmetros formais e parâmetros reais de um procedimento (ou função) devem ser compatíveis, isto é, o número de parâmetros reais deve ser igual ao de parâmetros formais, precisam estar sempre escritos na mesma ordem e ser do mesmo tipo.

Na declaração de parâmetros variáveis simples e arrays, a presença da palavra reservada VAR indica parâmetros por endereço e sua ausência indica parâmetros por valor.

Os parâmetros reais correspondentes a parâmetros formais por valor devem ser expressões (lembrando que uma variável ou constante é um caso particular de expressão) enquanto que os correspondentes à parâmetros formais por endereço devem ser variáveis simples ou arrays.

Qualquer operação sobre um parâmetro formal por endereço altera o valor de seu parâmetro real correspondente, enquanto que no caso de parâmetros formais por valor, a alteração de parâmetro formal não afeta o parâmetro real correspondente.

Quando um parâmetro formal é um procedimento

(ou função), o parâmetro real correspondente deve ser um procedimento (ou função) sem parâmetros ou com parâmetros somente por valor.

Os parâmetros formais e os objetos locais do procedimento (ou função), isto é, objetos declarados no seu corpo, não estão definidos fora do procedimento(ou função). No corpo de uma função, seu nome é considerado uma variável simples local e deve existir pelo menos um comando que lhe atribua um valor, que será o resultado da função.

1.3.4.1. FUNÇÕES INTERNAS

Algumas funções mais comumente usadas são pré-definidas na linguagem.

A tabela a seguir relaciona essas funções, seus parâmetros e os seus resultados:

ODD (X)	"TRUE" se X é impar, "FALSE" caso contrário.
EOLN	"TRUE" se fim de linha na impressora e "FALSE" caso contrário
EOF	"TRUE" se fim de arquivo na leitora de cartões e "FALSE" caso contrário
EOPG	"TRUE" se fim de página na impressora e "FALSE" caso contrário
ABS (X)	valor absoluto de X
SQR (X)	X elevado ao quadrado
TRUNC (X)	parte inteira de X
ROUND (X)	(arredondamento): TRUNC (X+0.5) se X >= 0 TRUNC (X-0.5) se X < 0
SUCC (X)	caráter seguinte a X no conjunto ordenado dos caracteres ou próximo inteiro de X (X + 1)
PRED (X)	caráter anterior a X no conjunto ordenado dos caracteres ou inteiro anterior a X (X - 1)
SIN (X)	seno de X
COS (X)	coseno de X
ARCTAN (X)	arco cuja tangente é X
LN (X)	logaritmo natural de X

EXP (X) exponencial de X
SQRT (X) raiz quadrada de X
ORD (X) número ordinal de X no conjunto ordenado dos caracteres
CHR (X) carãter cujo número ordinal é X

A tabela a seguir fornece os tipos de argumentos (X) aos quais podem ser aplicadas as funções internas e o tipo do resultado obtido:

função	argumento	resultado
ODD	inteiro	booleano
EOLN		booleano
EOF		booleano
EOPG		booleano
ABS	inteiro	inteiro
	real	real
SQR	inteiro	inteiro
	real	real
TRUNC	real	inteiro
ROUND	real	inteiro
SUCC	carãter	carãter
	inteiro	inteiro
PRED	carãter	carãter
	inteiro	inteiro
SIN COS ARCTAN LN EXP SQRT	real ou inteiro	real
ORD	carãter	inteiro
CHR	inteiro	carãter

1.4. ARRAYS

Um array é uma estrutura de dados n-dimensional, de acesso direto, usada para agrupar elementos que tenham as mesmas características, isto é, mesmo tipo e mesma precisão ou comprimento.

Somente o array recebe um nome, e seus itens individuais são referenciados pelo nome do array seguido de índices entre colchetes, que indicam sua posição relativa dentro da estrutura. Os índices tem que ser números inteiros.

Um array de uma dimensão é chamado um vetor e um de duas dimensões uma matriz, a primeira dimensão indicando o número de linhas e a segunda o de colunas.

O nome, tipo e as dimensões de um array são declarados no programa juntamente com as variáveis. O número de componentes do array é constante e definido na declaração do array.

1.4.1. DECLARAÇÃO DE ARRAY

Formato da declaração de array:

```
VAR lista de identificadores : ARRAY [ind1,ind2,...,indn]  OF
    tipo ,
    lista de identificadores : ARRAY [inda,indb,...,indm]  OF
    tipo1 ;
```

onde:

lista de identificadores contém os identificadores que indicam os nomes dos arrays e ind1,ind2,...,indn,inda,indb,..indm são as dimensões dos arrays.

Cada dimensão tem a forma:

```
expressão1 : expressão2
ou
expressão2
```

onde:

expressão1 e expressão2 são expressões compostas somente de números e/ou nomes de constantes já definidas, que tem que produzir resultados inteiros e indicam os limites inferior e superior, respectivamente, de cada dimensão. Quando expressão1 é omitida, o limite inferior da dimensão correspondente é considerado igual a 1.

Obs.: expressões são definidas na próxima seção;

tipo e tipo1 são uma das palavras reservadas INTEGER, REAL, BOOLEAN ou CHAR e indicam o tipo dos elementos dos arrays.

Para cada dimensão, tanto o limite inferior como o superior podem ser positivos ou negativos, porém, o limite inferior não pode ser maior que o limite superior.

Exemplos de declarações de arrays:

```
VAR CONJ : ARRAY [ 2:5 , -1:4 ] OF REAL ;
VAR A,B,C : ARRAY [ 0 : 6 ] OF INTEGER ,
    MAT : ARRAY [ 8 ] OF BOOLEAN ;
```

A declaração de um array pode estar contida na declaração de variáveis simples, por exemplo:

```
VAR A,B,C : INTEGER, CONTA : ARRAY [ 9 ] OF REAL;
```

1.5. EXPRESSÕES

O conceito de expressões aqui utilizado é o mesmo da aritmética comum: é uma sequência de operandos (variáveis e constantes) conectados por operadores de várias espécies.

Expressões são usadas para representação de fórmulas matemáticas e para cálculo de um valor a ser atribuído a uma variável.

Um operando pode ser uma variável simples (um nome de variável simples ou um elemento de array), uma constante (um número inteiro ou real, um literal ou um nome de constante), um nome de função (interna ou do programador) com seus argumentos, se houver, ou pode ser um operando

matricial (apenas um nome de array sem os índices).

A referência ao nome de uma função faz com que seja ativada a execução da função e seja devolvido ao ponto de referência o valor resultante do cálculo da função.

A referência a uma variável, uma constante, um elemento de um array ou a um array faz com que seja utilizado no ponto de referência o valor do dado naquele instante.

O cálculo de uma expressão é efetuado da esquerda para direita, observando-se a regra de precedência dos operadores, salvo quando a expressão contém parênteses, em cujo caso o cálculo começa pelo conteúdo dos parênteses mais internos, e a precedência dos operadores é respeitada dentro de cada par de parênteses.

Os operadores são os definidos anteriormente em 1.2.6., mais as palavras reservadas MOD e DIV, que são operadores de produto, e sua precedência é a seguinte, em ordem decrescente:

¬ (not, aplicado a um operando booleano)

operadores de produto (* / MOD DIV &)

operadores de adição (+ - |)

operadores relacionais (= ¬= < > <= >=)

Uma expressão como por exemplo:

A + B * C / A - 2 < 3 + D / E

é calculada do seguinte modo:

A + M / A - 2 < 3 + N, onde M = B*C e N = D/E

A + P - 2 < 3 + N, onde P = M/A

S - 2 < T, onde S = A+P e T = 3+N

Q < T, onde Q = S-2

V, onde V = valor da expressão (no caso, "TRUE" se Q<T ou "FALSE" se Q>=T).

Os operadores de adição + e - podem ser unários ou binários. São binários quando usados entre dois operandos (A+B ou A-B, por exemplo) especificado as operações de adição e subtração, respectivamente, entre os dois operandos, e

unários quando usados simplesmente antes de um operando (+A ou -A, por exemplo) denotando o sinal do operando. O operador ¬ é sempre unário e os demais operadores são sempre binários.

O operador DIV dá o quociente truncado (não arredondado) da divisão. Ou seja, é a divisão inteira.

Por exemplo:

```
 4  DIV  3 = 1
29  DIV 10 = 2
 5  DIV  2 = 2
```

O operador MOD dá o resto da divisão.

A MOD B = A - ((A DIV B) * B)

Por exemplo:

```
8  MOD  3 = 2
4  MOD  3 = 1
4  MOD  2 = 0
```

Só há conversão entre dados inteiros e reais. Por este motivo há restrições na mistura de tipos em expressões.

A relação entre os tipos do argumento e o resultado das funções internas está na seção 1.3.4.1. (Funções Internas).

São definidas as seguintes operações matriciais:

```
A + S          S + A
A - S          S * A
A * S          A + A
A - S
```

onde A é um nome de array e S é um escalar (uma variável simples, um elemento de um array, uma constante ou uma referência à uma função). As operações são efetuadas elemento a elemento do array e o resultado é sempre um array.

Exemplos de expressões válidas:

MEDIA >= 5.0

VALOR - VALOR * (DESCONTO / 100)

(NUM + 8) / 10 - ((CONT + 14) MOD 2)

A tabela a seguir fornece os tipos dos operandos (simples ou matriciais) aos quais podem ser aplicados os operadores e o tipo do resultado da operação.

OPERADOR	OPERANDO1	OPERANDO2	RESULTADO
& |	booleano	booleano	booleano
¬	booleano		booleano
+	inteiro	inteiro	inteiro
-	real	real	
*	real	inteiro	real
	inteiro	real	
DIV MOD	inteiro	inteiro	inteiro
/	inteiro	inteiro	
	real	real	
	real	inteiro	real
	inteiro	real	
=	inteiro	inteiro	
¬=	real	real	booleano
<	inteiro	real	
>	real	inteiro	
>=	booleano	booleano	
<=	caráter	caráter	

1.6. COMANDOS

Os comandos descrevem as ações a serem tomadas com os dados e os cálculos a serem efetuados e são executados na ordem em que aparecem no programa, salvo indicação em contrário.

Temos os seguintes tipos de comandos:

- comandos simples
- comandos condicionantes
- comandos iterativos
- comandos de entrada e saída
- comando composto

Os comandos simples são:

- comando de atribuição
- comando GOTO
- ativação de procedimento

Os comandos condicionantes são:

- comando IF
- comando CASE

Os comandos iterativos são:

- comando FOR
- comando WHILE
- comando REPEAT

Os comandos de entrada e saída são:

- comando READ
- comando WRITE

O comando composto é o comando BEGIN. Os comandos iterativos são estruturados, podendo conter 1 ou mais comandos componentes.

O símbolo ";" é considerado terminal de comando.

1.6.1. COMANDOS SIMPLES

1.6.1.1. COMANDO DE ATRIBUIÇÃO

É usado para atribuir um valor a uma variável.

Formato do comando de atribuição:

variável := expressão

O valor obtido pelo cálculo da expressão à direita do sinal de atribuição ":=" é atribuido a variável à esquerda do sinal. A expressão deve ser do mesmo tipo da variável, com uma exceção: se a variável for real ou inteira, a expressão pode ser real ou inteira.

O comando A := A + 5 é válido e deve ser entendido como: o atual valor de A é acrescido de 5 e atribuido a própria variável A.

Exemplos de comandos de atribuição válidos:

NOME := "J"

B := 0

MAT [J] := MAT [J-1] + 1

OK := VALOR <= SALDO

1.6.1.2. COMANDO GOTO

É usado para indicar uma quebra na sequência da execução dos comandos, que continua a partir do comando para o qual foi ordenado o desvio.

Formato do comando GOTO:

GOTO ident

onde:

ident é um identificador que deve ter sido declarado como rótulo e deve aparecer precedendo um outro comando do programa, para o qual a sequência de execução é desviada.

Exemplos de comandos GOTO válidos:

se A e CONTA são declarados e usados como rótulos,

GOTO A

GOTO CONTA

1.6.1.3. ATIVAÇÃO DE PROCEDIMENTO

É usado para ativar um procedimento. A execução do procedimento é ativada no instante da execução deste comando.

Formato do comando ativação de procedimento.

nome
ou
nome (lista de parâmetros)

onde:

nome é um nome de procedimento, já declarado, e lista de parâmetros é a lista dos parâmetros reais do procedimento.

Após a execução do procedimento o controle retorna ao comando seguinte ao comando de ativação do procedimento.

Exemplos válidos:

se SCAN e DELETA já foram declarados como nomes de procedimentos e DELETA requer um parâmetro,
SCAN
DELETA (ARQ)

1.6.2. COMANDOS CONDICIONANTES

Um comando condicionante especifica uma condição que será avaliada. A ação subsequente do programa vai depender do valor desta condição.

1.6.2.1. COMANDO IF

O comando IF permite a execução ou omissão de um comando, dependendo da veracidade ou falsidade da condição dada.

Há duas formas de comando IF.

IF expressão THEN comando1
e
IF expressão THEN comando1 ELSE comando2

onde:

expressão é uma expressão booleana não matricial.

Na primeira forma, se a expressão for verdadeira o comando1 é executado. Se for falsa, é ignorado. Na segunda forma, se a expressão for verdadeira o comando1 é executado e o comando2 é ignorado. Se for falsa, o comando1 é ignorado e o comando2 é executado.

Observar que não deve haver ";" antes do ELSE.

Exemplos do comando IF:

```
IF A > B THEN A := A - B

IF MES = 12 THEN MES := 1
            ELSE MES := MES + 1
```

1.6.2.2. COMANDO CASE

O comando CASE consiste de uma expressão seletora e uma lista de comandos, cada um rotulado por uma ou mais constantes do mesmo tipo daquela, que deve ser do tipo INTEGER ou CHAR. O comando selecionado para execução é o rotulado pela constante igual ao valor da expressão no instante da execução do comando CASE. Se não houver um comando com tal rótulo, o efeito do comando CASE é nulo, uma vez que nenhum de seus comandos componentes será executado.

Após a execução do comando selecionado o controle passa ao comando seguinte ao CASE, salvo se o comando selecionado for de desvio.

Formato do comando CASE:

```
CASE expressão OF
     lista de rótulos : comando ;
          .
          .
          .
     lista de rótulos : comando ;
END ;
```

onde:

lista de rótulos é uma série de 1 ou mais literais ou números inteiros sem sinal separados por vírgulas.

Os rótulos do comando CASE não são considerados como rótulos normais e não devem ser declarados como tal nem podem ser referenciados por um comando GOTO.

Exemplos do comando CASE:

se I e NOME são declarados INTEGER e CHAR, respectivamente,

```
CASE I OF 0 : X := 0 ;
          1 : X := A + B ;
          2 : X := A - B ;
          3 , 4 : X := A + 2*B ;
END

CASE NOME OF 'A' , 'B' , 'C' : NOME := SUCC(NOME) ;
             'X' , 'Y' , 'Z' : NOME := PRED(NOME) ;
END;
```

1.6.3. COMANDOS ITERATIVOS

Um comando iterativo indica que seus comandos componentes devem ser repetidamente executados, até que ou enquanto uma ou mais condições sejam satisfeitas.

1.6.3.1. COMANDO FOR

Formato do comando FOR:

```
FOR variável de controle := valor1 TO valor2 DO comando
ou
FOR variável de controle := valor1 DOWNTO valor2 DO comando
```

onde:

variável de controle é o nome de uma variável simples e valor1 e valor2 são expressões. Os três devem ser do tipo inteiro. Valor1 e valor2 são calculados apenas uma vez no

início da execução do comando e a variável de controle não deve ser alterada pelo comando FOR.

O valor final da variável de controle é deixado indefinido após saída normal do comando FOR.

O comando é executado como se segue:

Inicialmente a variável de controle é feita igual a valor1. Se este valor for maior que valor2 (na segunda forma, se for menor que valor2) o controle passa ao comando seguinte ao FOR. Caso contrário, o comando que segue o DO é executado; a variável de controle é incrementada (decrementada, no segundo caso) de um; volta a ser comparada com valor2, e assim sucessivamente.

Exemplo do comando FOR:

```
FOR I := 0 TO 8 DO CONJ [I] := 0
```

para zerar os 9 primeiros elementos do array CONJ, o que equivale a:

```
I := 0 ;
ZERA:
CONJ [I] := 0 ;
I := I + 1 ;
IF I <= 8 THEN GOTO ZERA
```

1.6.3.2. COMANDO WHILE

Formato do comando WHILE:

```
WHILE expressão DO comando ;
```

onde:

expressão deve produzir um valor booleano.

É executado da seguinte forma:

Inicialmente o valor da expressão é testado. Se for FALSE, o comando que segue o WHILE é executado. Se for TRUE, o comando que segue o DO é executado, e o valor da expressão volta a ser testado, e assim sucessivamente.

Exemplo do comando WHILE:

```
N := 0 ;
WHILE MAT [N] ¬= 0 DO
        BEGIN
            B [N] := MAT [N] ;
            N := N + 1 ;
        END
```

para fazer os elementos iniciais do array B iguais aos elementos correspondentes do array MAT, até ser encontrado um elemento de MAT igual a zero, o que equivale a:

```
N := 0 ;
VOLTA:
IF MAT [N] ¬= 0
THEN BEGIN
        B [N] := MAT [N] ;
        N := N + 1 ;
        GOTO VOLTA ;
     END
```

1.6.3.3. COMANDO REPEAT

Formato do comando REPEAT:

REPEAT lista de comandos UNTIL expressão.

onde:

lista de comandos é uma série de um ou mais comandos separados uns dos outros por ponto e vírgula, e expressão deve produzir um valor booleano.

O comando é executado da seguinte forma:

Inicialmente a lista de comandos é executada e o valor da expressão é testado. Se for TRUE, o comando que segue o REPEAT é executado. Se for FALSE, a lista de comandos é novamente executada e o valor da expressão volta a ser

testado, e assim sucessivamente.

Exemplo do comando REPEAT:

```
I := 2 ;
SOMA := 0 ;
REPEAT SOMA := SOMA + I ;
       I := I + 2 ;
UNTIL I > 3000 ;
```

para somar todos os números pares de 1 a 3000, inclusives, o que equivale a:

```
I := 2 ;
SOMA := 0 ;
INCR :
SOMA := SOMA + I ;
I := I + 2 ;
IF I <= 3000
THEN GOTO INCR ;
```

1.6.4. COMANDOS DE ENTRADA E SAÍDA

São usados para ler dados de cartões perfurados ou escreve-los através da impressora.

Somente estes dois arquivos (cartões e impressora) podem ser utilizados.

1.6.4.1. COMANDO READ

O comando de entrada tem o seguinte formato:

```
READ (lista de dados de entrada)
ou
READP (lista de dados de entrada)
ou
READD (lista de dados de entrada)
ou
READPD (lista de dados de entrada)
```

onde:

lista de dados de entrada é uma série de uma ou mais variáveis separadas por vírgula, cujos valores devem estar perfurados nos cartões. Estes valores serão lidos e atribuidos às variáveis da lista, na mesma ordem.

Na modalidade READ os dados devem estar perfurados nos cartões em um formato pré-definido, iniciando cada um na coluna 1,10,20,30,40,50,60 ou 70, dependendo de onde terminou o último dado lido.

Na modalidade READD (READ direto) os dados vem em formato livre, separados por um ou mais brancos, por vírgula ou pelo fim do cartão (exceto para dados do tipo CHAR, para os quais não existem separadores).

As modalidades READP e READPD (READ próximo e READ próximo direto) são, respectivamente, variações das modalidades READ e READD, com o primeiro dado da lista iniciando sempre na coluna 1 do cartão corrente (se ainda não foi lido nenhum dado deste cartão) ou do cartão seguinte. Tirando essa diferença, são absolutamente iguais as suas originais.

A cada comando de entrada serão lidos tantos cartões quantos forem necessários para atender à lista de dados de entrada do comando.

Os dados booleanos podem estar perfurados de duas formas: T ou TRUE e F ou FALSE.

1.6.4.2. COMANDO WRITE

O comando de saída tem o seguinte formato:

WRITE (lista de dados de saída)
ou
WRITELN (lista de dados de saída)
ou
WRITEPG (lista de dados de saída)
ou
WRITED (lista de dados de saída)
ou

WRITELND (lista de dados de saída)
ou
WRITEPGD (lista de dados de saída)

onde:

lista de dados de saída é uma série de uma ou mais expressões separadas por vírgulas, cujos valores serão impressos na mesma ordem em que aparecem na lista.

Na modalidade WRITE os dados são impressos em posições pré-definidas, iniciando cada um na coluna 1,30,60 ou 90, dependendo de onde terminou a impressão do dado precedente.

Na modalidade WRITED (WRITE direto) os dados são impressos consecutivamente, sem espaços entre eles.

Nas duas modalidades, para impressão de dados inteiros são usadas 10 colunas e 15 colunas para dados reais. Para um dado do tipo CHAR são usados tantas colunas quantos forem os caracteres do dado. Um dado booleano ocupará 4 ou 5 colunas e será impresso TRUE ou FALSE, conforme o caso.

As modalidades WRITELN e WRITEPG (WRITE próxima linha e WRITE próxima página) são variações da modalidade WRITE e as modalidades WRITELND e WRITEPGD (WRITE próxima linha direto e WRITE próxima página direto) são variações da modalidade WRITED.

Nas modalidades WRITELN e WRITELND o primeiro dado da lista é impresso sempre na coluna 1 da linha corrente (se ainda não foi impresso nenhum dado nesta linha) ou da linha seguinte.

Nas modalidades WRITEPG e WRITEPGD o primeiro dado da lista é impresso sempre na coluna 1 da primeira linha da página corrente (se ainda não foi impresso nenhum dado nesta página) ou da página seguinte.

Tirando essas diferenças, as modalidades WRITELN, WRITELND, WRITEPG e WRITEPGD são absolutamente iguais às suas originais.

A cada comando de saída serão impressas tantas linhas quantas forem necessárias para atender à lista de dados de saída do comando.

Exemplos de comandos READ e WRITE:

```
READ (A,B[I])
READPD (C)
WRITELN (A,A+1)
WRITEPGD('B[4] = ',B[4]);
WRITELND ("TEXTO CONTENDO APÓSTROFE (')")
WRITE ('TEXTO CONTENDO ASPAS (")',X)
```

1.6.5. COMANDO COMPOSTO

1.6.5.1. COMANDO BEGIN

É usado para agrupar comandos executáveis, de modo que a execução deste comando implica na execução dos comandos que o compõem, na ordem em que são escritos (a menos que um dos comandos componentes force a quebra da sequencia).

1.7. ESTRUTURA DO PROGRAMA

Formato do programa:

```
PROGRAM nome ; bloco .
```

onde:

nome é um identificador que será o nome do programa e bloco contém as declarações dos objetos e os comando executáveis do programa e tem o seguinte formato:

```
declarações BEGIN comandos END
```

onde:

declarações é a lista das declarações de rótulos, constantes, variáveis, procedimentos e funções, como descritas anteriormente, e comandos é a lista dos comandos executáveis.

As declarações são opcionais e devem aparecer na seguinte ordem:

- declaração de rótulos
- declaração de constantes
- declaração de variáveis
- declaração de procedimentos e funções

As declarações de rótulos, constantes e variáveis devem aparecer somente uma vez cada uma. As de procedimentos e funções podem aparecer mais de uma vez, uma para cada procedimento ou função do programa.

Todos os objetos declarados em um bloco não estão definidos fora dele, isto é, não podem ser referenciados por comandos que estejam fora deste bloco. Os parâmetros formais dos procedimentos e funções são considerados como declarados no bloco do procedimento ou função.

Todos os procedimentos e funções são recursivos, sendo automaticamente empilhados os parâmetros por valor e as variáveis declaradas no bloco.

1.7.1. COMENTÁRIOS

Para facilidade de compreensão das ações do programa ou para fins de documentação, podem-se inserir comentários no texto dos programas. Estes são simplesmente impressos, sendo ignorados na compilação.

Um comentário é qualquer sequência de caracteres entre dois símbolos '%', e pode aparecer no programa em qualquer lugar onde possa ser colocado um branco.

Exemplos de comentários:

```
PROC CONTA ;
% COMENTÁRIO DESCREVENDO A ACAO DESTE PROCEDIMENTO %
BEGIN
   .
   .
   .
END ;
IF A < B
% DESCRICAO DA CONDICAO %
THEN ..........
VAR IND : INTEGER % VARIAVEL INDICE % ;
```

Brancos e comentários podem ser colocados em qualquer número entre dois elementos da linguagem, não podendo aparecer dentro de um identificador, número, palavra reservada ou operador duplo.

1.8. EXEMPLO DE PROGRAMA

```
PROGRAM POSTFIX ;
% ESTE PROGRAMA LE EXPRESSOES ARITMETICAS CORRETAS E AS %
% IMPRIME NA FORMA POS-FIXADA. OS OPERANDOS PODEM SER   %
% CONSTANTES INTEIRAS OU VARIAVEIS. EM AMBOS OS CASOS   %
% DEVEM SER COMPOSTAS DE APENAS UM CARATER. OS OPERADO- %
% RES SAO: +, -, E *, E AS EXPRESSOES PODEM CONTER      %
% PARENTESES.                                           %
VAR CH : CHAR ;
  PROC FIND ;
  % ROTINA PARA PULAR BRANCOS %
    BEGIN
      REPEAT READD(CH) UNTIL(CH ¬= ' ')
    END ; % FIND %
  PROC EXPRE ;
  % ROTINA PARA ANALISAR EXPRESSAO %
    VAR OP : CHAR ;
    PROC TERM ;
    % ROTINA PARA ANALISAR TERMO %
      PROC FATOR ;
      % ROTINA PARA ANALISAR FATOR %
        BEGIN
          IF CH = '('
          THEN BEGIN
                 % ANALISA EXPRESSAO ENTRE PARENTESES %
                 FIND ;
                 EXPRE ;
               END
          ELSE WRITED(CH) ; % IMPRIME O OPERANDO %
          FIND
        END ; % FATOR %
      BEGIN
        FATOR ;
        WHILE CH = '*' DO
          BEGIN
            FIND ;
            FATOR ;
            WRITED('*') ;
```

```
        END
    END ; % TERM %
  BEGIN
    TERM ;
    WHILE CH = '+' | CH = '-' DO
      BEGIN
        OP := CH ;
        FIND ;
        TERM ;
        WRITED(OP) % IMPRIME OPERADOR %
      END
  END ; % EXPRE %
BEGIN
  FIND ;
  REPEAT
    % COMECA NOVA EXPRESSAO %
    WRITED(' ') ;
    EXPRE ;
    WRITELN(' ') ;
  UNTIL CH = '.'
END.
```

2. ESTRUTURA DO COMPILADOR

2.1. ANÁLISE

As análises léxica, sintática e semântica e a geração do código foram feitas em um só passo.

A análise léxica é feita por um analisador que devolve a cada chamada o próximo elemento léxico, fazendo deteção e tratamento de erros nestes elementos. O analisador devolve sempre um elemento correto.

Para efeito de análise sintática, a linguagem fonte foi dividida em seções. De modo geral, cada uma destas seções originou um módulo do compilador. Para melhor idéia desta divisão, ver o diagrama da linguagem, em anexo. Cada seção é analisada segundo um autômato finito determinístico, com exceção de expressões, que são analisadas segundo o método apresentado pelo Professor Pedro Salembauch no curso de Construção de Compiladores na COPPE, no ano de 1975, tendo o mesmo sido classificado por ele como empírico.

2.2. TRATAMENTO DE ERROS

As mensagens de erro são impressas logo após o cartão que produziu o erro. É sempre feita tentativa de se detetar o erro o mais cedo possível e de continuar a análise após a deteção de um erro. Em alguns casos são feitas correções simples, como inserção de ";" e de ")". Há uma rotina separada para impressão das mensagens de erro e estas mensagens estão armazenadas sob forma de tabela.

Para evitar repetição de mensagens de erro, na primeira ocorrência de um identificador não declarado é dada mensagem do erro e ele é inserido na tabela de símbolos com tipo igual a indefinido, para inibir testes semânticos nas futuras ocorrências do identificador.

2.3. GERAÇÃO DE CÓDIGO

Após as análises sintática e semântica de um comando, é gerado código para as construções corretas. Não é gerado código para expressões que só contenham constantes. Estas expressões são resolvidas durante a compilação.

O código gerado é para uma máquina virtual, cujas características são descritas a seguir.

Os detalhes da geração de código para cada comando da linguagem são mostrados junto da descrição do analisador do comando.

Após a descrição da máquina virtual é apresentado o esquema geral de geração de código.

2.3.1. DESCRIÇÃO DA MÁQUINA VIRTUAL

Na definição desta máquina (e portanto de sua linguagem), dois cuidados foram observados: não impossibilitar futura otimização do código e não orientar a linguagem para uma máquina específica, conferindo portabilidade ao compilador, uma vez que a geração de código para uma máquina real pode ser feita num passo suplementar, tomando como entrada o programa objeto da máquina virtual.

Os seguintes aspectos caracterizam a máquina para a qual é feita a geração de código: memória, registradores, forma de endereçamento e conjunto de instruções.

2.3.1.1. MEMÓRIA

A memória foi dividida em 5 áreas distintas: uma para armazenar as instruções do programa e uma para cada tipo de dado(inteiro, real, booleano e caráter)[1].

Todas as posições de memória dentro de um mesmo tipo tem o mesmo tamanho, dependente do tipo, e podem armazenar um elemento. Por exemplo: uma instrução, um dado inteiro, um dado real, um dado booleano ou um dado caráter (um caráter).

As memórias para dados tem endereços negativos e positivos.

Na memória inteira cada posição é composta por 4 bytes e armazena um número inteiro na forma de número binário em complemento a dois. Cada posição da memória real é composta por 4 bytes e armazena um número real na forma de número binário em ponto flutuante. Cada posição da memória booleana é composta por

---

[1]) Porisso as referências a área de memória e a tipo de memória terão o mesmo significado neste texto.

1 byte e armazena um dado booleano. Cada posição da memória caráter é composta por 1 byte e armazena um caráter no formato EBCDIC.

A memória para instruções só tem endereços positivos. Cada posição tem 16 bytes e armazena uma instrução no formato:

código de instrução	operando 1	operando 2	operando 3

2.3.1.2. REGISTRADORES

Existem 3 tipos de registradores: registradores base, indexadores e de código de condição.

O registrador de código de condição é apenas um e indica as condições resultantes da execução de determinadas instruções, como por exemplo as instruções de comparação. Este registrador tem 4 bits mutuamente exclusivos, isto é, a cada instante um e apenas um deles tem valor 1. Seu conteúdo é alterado pela execução de algumas instruções(ver adiante na descrição das instruções) e é testado pela instrução de desvio.

Os indexadores são em número de tres. Estão implicitamente associados a cada um dos operandos da instrução e são usados na formação do endereço efetivo do operando respectivo.

Os registradores base são usados apenas para endereçamento das áreas de dados e estão contidos numa pilha em que cada elemento é uma matriz de registradores.

Esta matriz tem 8 colunas(duas para cada tipo de dados: uma para os endereços negativos e outra para os endereços positivos). O número de linhas desta matriz é definido por ocasião da carga do programa, enquanto que a pilha não tem tamanho limitado.

Embora seja bastante distanciado da realidade, este esquema foi adotado para os registradores base para simplificar a implementação. Isto porém não impossibilita a posterior transformação do programa objeto da máquina virtual para o de uma máquina real.

2.3.1.3. FORMA DE ENDEREÇAMENTO

O endereço efetivo de cada operando é calculado por ocasião da execução da instrução. De um modo geral, tres parcelas são utilizadas no cálculo do endereço efetivo de cada operando: o conteúdo de um registrador base, o conteúdo de um registrador indexador e o deslocamento do operando.

A utilização do registrador indexador depende da instrução sendo executada, conforme descrito adiante em Conjunto de Instruções. A seleção do indexador a ser usado é implicita, uma vez que cada um está associado a um operando.

O deslocamento é sempre usado no calculo do endereçamento efetivo do operando. Seu valor é indicado na instrução no campo do operando correspondente.

Tres fatores influenciam na determinação do registrador base a ser utilizado para o cálculo do endereço efetivo do operando.

Em primeiro lugar, o tipo da instrução sendo executada, pois define o tipo do operando (inteiro , real , booleano ou caráter).

Em segundo lugar o valor do deslocamento, se positivo ou negativo.

Em conjunto, estes dois fatores determinam a coluna da matriz de registradores a ser utilizada(lembrando que é sempre usada a matriz que está no topo da pilha).

Finalmente, a indicação da linha da matriz de registradores a ser utilizada é feita na instrução, no campo do operando.

2.3.1.4. CONJUNTO DE INSTRUÇÕES

Conforme foi dito anteriormente, as instruções tem tamanho fixo, sob forma de quádruplas:

código da instrução	operando 1	operando 2	operando 3

O Campo de código da instrução ocupa 8 bits enquanto que cada campo de operando ocupa 40 bits.

Quando o campo de operando indicar o endereço do operando, será subdividido em 2 partes: número do registrador base (os oito primeiros bits) e deslocamento do operando (últimos 32 bits).

O valor zero no campo número do registrador base indica que não será utilizado o valor do registrador base para cálculo do endereço efetivo do operando, uma vez que os registradores base são numerados a partir de um.

Na descrição das instruções abaixo, será usado o seguinte esquema, salvo indicação em contrário:

mnemônico (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

onde:

. mnemônico representa o código da instrução;
. B1, B2 e B3 são os números dos registradores base dos operandos;
. D1, D2 e D3 são os deslocamentos e
. I1, I2 e I3 indicam a utilização ou não do registrador indexador correspondente.

O sinal menos (-) em qualquer destes campos indica que o mesmo não é usado pela instrução. A indicação dos tipos dos operandos será feita em cada instrução.

Os estados do registrador de código de condição, resultantes da execução das instruções, serão descritos pelo número decimal correspondente à configuração binária do mesmo, conforme tabela abaixo, associado ao resultado da instrução:

número decimal	configuração binária
1	0001
2	0010
4	0100
8	1000

A omissão da descrição dos estados do registrador de código de condição indica que o mesmo não é afetado pela execução da instrução em questão.

DESCRIÇÃO DAS INSTRUÇÕES

1) ADI - adição inteira
formato: ADI (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Soma os valores dos operandos 1 e 2, colocando o resultado no operando 3. Os 3 operandos são inteiros.

2) SUBI - subtração inteira
formato: SUBI (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Subtrai do valor do operando 1 o do operando 2, colocando o resultado no operando 3. Os 3 operandos são inteiros.

3) MLTI - multiplicação inteira
formato: MLTI (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Multiplica os valores dos operandos 1 e 2, colocando o resultado no operando 3. Os 3 operandos são inteiros.

4) DIVI - divisão inteira
formato: DIVI (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Divide o valor do operando 1 pelo do operando 2, colocando o resultado no operando 3. Os 3 operandos são inteiros.

5) MOD - módulo da divisão
formato: MOD (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Divide o valor do operando 1 pelo do operando 2, colocando o valor do resto da divisão no operando 3. Os 3 operandos são inteiros.

6) ADR - adição real
formato: ADR (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Soma os valores dos operandos 1 e 2, colocando o resultado no operando3. Os 3 operandos são reais.

7) SUBR - subtração real
formato: SUBR (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Subtrai do valor do operando 1 o do operando 2, colocando o resultado no operando 3. Os 3 operandos são reais.

8) MLTR - multiplicação real

formato: MLTR (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Multiplica os valores dos operandos 1 e 2, colocando o resultado no operando 3. Os 3 operandos são reais.

9) DIVR - divisão real

formato: DIVR (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Divide o valor do operando 1 pelo do operando 2, colocando o resultado no operando 3. Os 3 operandos são reais.

10) ATRIB - atribuição

formato: ATRIB (T),(B2,D2,I2),(B3,D3,I3)

onde T é o tipo da atribuição: 1 - inteira
2 - real
3 - booleana
4 - caráter

O operando 2 é a origem e o 3 o destino. Os dois são do mesmo tipo, igual ao da atribuição. O valor do operando de origem é copiado para o de destino.

11) CVT - conversão

formato: CVT (T),(B2,D2,I2),(B3,D3,I3)

onde T é o tipo da conversão: 1 - inteira
2 - real

É uma atribuição onde o valor do operando 2 é convertido ao tipo do operando 3 e copiado para êle. Se T=1, o operando 2 é real e o 3 inteiro e, se T=2, o operando 2 é inteiro e o 3 real.

12) DESV - desvio

formato: DESV (T),(CC),(E)

onde:

T - tipo do desvio: 0 se direto,
1 se indexado

CC - código da condição que deve ser satisfeita para que ocorra o desvio, segundo tabela:

2 - maior
4 - menor
6 - diferente

8 - igual
10 - maior ou igual
12 - menor ou igual
15 - incondicional

E - endereço de uma posição da memória de instruções

É efetuada a operação lógica & entre o operando CC e o registrador de código de condição. Se o resultado desta operação for diferente de zero, haverá um desvio do fluxo de execução do programa para a instrução especificada pelo endereço de desvio. Caso contrário o fluxo prosseguirá normalmente.

O endereço de desvio será o valor de E no caso de desvio direto e no caso de desvio indexado será a soma do valor de E com o conteúdo do registrador indexador 3.

13) COMP - comparação

formato: COMP (T),(B2,D2,I2),(B3,D3,I3)

onde:

T - tipo da comparação: 1 - inteira
2 - real
3 - booleana
4 - caráter

Efetua a comparação entre os valores dos operandos 2 e 3 e altera o valor do registrador de código de condição, de acordo com o resultado da comparação:

2 se operando 1 > operando 2
4 se operando 1 < operando 2
8 se operando 1 = operando 2

Os dois operandos são do mesmo tipo, igual ao da comparação.

14) SET - carrega registrador indexador

formato: SET (-,-,-),(R),(B3,D3,-)

onde:

R - número de um dos registradores indexadores: 1, 2 ou 3.

Copia para o indexador R o valor do operando 3. O operando 3 é do tipo inteiro.

15) ADREG - incrementa registrador indexador

formato: ADREG (-,-,-),(R),(B3,D3,-)

onde:

R - número de um dos registradores indexadores: 1, 2 ou 3.

Soma o valor do operando 3 ao do indexador R, colocando o resultado no indexador R. O operando 3 é do tipo inteiro.

16) CREG - compara conteúdo de registrador indexador

formato: CREG (-,-,-),(R),(B3,D3,-)

onde:

R - número de um dos registradores indexadores: 1, 2 ou 3.

Compara o valor do indexador R com o do operando 3, alterando o valor do registrador de código de condição, de acordo com o resultado da comparação:

2 se indexador > operando 3

4 se indexador < operando 3

8 se indexador = operando 3

O operando 3 é do tipo inteiro.

17) ZEREG - zera registradores indexadores

formato: ZEREG (V1),(V2),(V3)

onde Vn = 0 ou 1.

Zera o indexador cujo operando correspondente é igual a 1: R1 = 0 se V1 = 1,
R2 = 0 se V2 = 1,
R3 = 0 se V3 = 1.

18) SETM - carrega simultaneamente registradores indexadores

formato: SETM (B1,D1,-),(B2,D2,-),(B3,D3,-)

Coloca no indexador 1 o valor do operando 1, no indexador 2 o do operando 2 e no indexador 3 o do operando 3. Os 3 operandos são inteiros.

19) SETB - carrega registrador base

formato: SETB (T),(B),(B3,D3,-)

onde:

T - tipo do registrador base B, conforme tabela (indica a coluna da matriz de registradores base em que se encontra o re-

gistrador): 1 - registrador base de variáveis inteiras
2 - registrador base de variáveis reais
3 - registrador base de variáveis booleanas
4 - registrador base de variáveis caráter
5 - registrador base de temporários inteiros
6 - registrador base de temporários reais
7 - registrador base de temporários booleanos
8 - registrador base de temporários caráter

B - indicação da linha da matriz de registradores base onde se se encontra o registrador base

Copia o valor do operando 3 no registrador base especificado por B e T. O operando 3 é inteiro.

20) STOB - armazena conteúdo de registrador base
formato: STOB (T),(B),(B3,D3,-)

onde:

T - o mesmo da instrução anterior (SETB)

B - o mesmo da instrução anterior (SETB)

Copia o valor do registrador base especificado por B e T no operando 3. O operando 3 é inteiro.

21) EMPB - empilha matriz de registradores base
formato: EMPB (-,-,-),(-,-,-),(B3,D3,-)

Coloca no topo da pilha uma nova matriz de registradores base, copiando para a nova matriz os valores dos registradores desde a linha 1 até a linha especificada pelo operando 3. O operando 3 é o endereço da área que contém o número do registrador base até o qual deverá ser efetuada a cópia, inclusive. O operando 3 é do tipo inteiro.

22) DESB - desempilha matriz de registradores base
formato: DESB (-,-,-),(-,-,-),(-,-,-)

Retira do topo da pilha a matriz de registradores base.

23) MSG - imprime mensagem
formato: MSG (-,-,-),(-,-,-),(N)

onde:

N - número da mensagem a ser impressa

24) EXIT - encerra o processamento
formato: EXIT (-,-,-),(-,-,-),(-,-,-)

Imprime mensagem de fim normal de processamento e encerra a execução do programa.

25) NOT - operação not
formato: NOT (-,-,-),(B2,D2,I2),(B3,D3,I3)

Efetua a operação NOT no operando 2, colocando o resultado no operando 3. Os 2 operandos são booleanos.

26) AND - operação and
formato: AND (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Efetua a operação AND entre os operandos 1 e 2, colocando o resultado no operando 3. Os 3 operandos são booleanos.

27) OR - operação or
formato: OR (B1,D1,I1),(B2,D2,I2),(B3,D3,I3)

Efetua a operação OR entre os operandos 1 e 2, colocando o resultado no operando 3. Os 3 operandos são booleanos.

28) READ - instrução de entrada de dados
formato: cod (A),(N),(P)

onde:

cod - um dos códigos de operação: READ, READP, READD ou READPD

A - código do arquivo (atualmente usado apenas 0, leitora de cartões)

N - número de operandos a serem lidos

P - ponteiro (endereço absoluto) para lista de descritores dos operandos

A lista de descritores dos operandos está na memória inteira positiva e existem 3 entradas para cada operando nesta lista:

| TEND | END | RBASE |

TEND - indica o tipo do dado a ser lido e a forma de endereçamento usado, de acordo com a tabela abaixo:

TEND	Descrição
3	inteiro endereçamento direto
7	real endereçamento direto
11	booleano endereçamento direto
14	caráter endereçamento direto
4	inteiro endereçamento indireto
8	real endereçamento indireto
12	booleano endereçamento indireto
15	caráter endereçamento indireto

END - endereço do operando

RBASE - número da linha da matriz de registradores base a ser usada no cálculo do endereço efetivo do operando

No caso de endereçamento direto, END contém o deslocamento do operando, RBASE é a base relativa ao operando e a coluna a ser usada da matriz de registradores base será 1, 2, 3 ou 4, conforme o tipo do dado (inteiro, real, booleano ou caráter).

No caso de endereçamento indireto, END contém o deslocamento da posição de memória inteira que contém o endereço absoluto do operando. RBASE é a base relativa a esta posição de memória e a coluna a ser usada da matriz de registradores base será sempre 5.

Os dados são lidos de cartões obedecendo ao posicionamento relativo a cada tipo de comando READ (ver comandos de entrada e saída na descrição da linguagem).

29) WRITE - instrução de saída de dados

formato: cod (A),(N),(P)

onde:

cod - um dos códigos de operação: WRITE, WRITED, WRITELN, WRITELND, WRITEPG ou WRITEPGD

A - código do arquivo (atualmente usado apenas 0, impressora)

N - número de operandos a serem impressos

P - ponteiro (endereço absoluto) para lista de descritores dos operandos

A lista de descritores dos operandos está na memória inteira positiva e existem quatro entradas para cada operan

do nesta lista:

TEND	TAM	END	RBASE

TEND - indica o tipo do dado a ser impresso e a forma de endereçamento usada, conforme a tabela abaixo:

TEND	Descrição
1	inteiro, endereçamento direto, endereço negativo
2 e 3	inteiro, endereçamento direto, endereço positivo
4	inteiro, endereçamento indireto
5	real, endereçamento direto, endereço negativo
6 e 7	real, endereçamento direto, endereço positivo
8	real, endereçamento indireto
9	booleano, endereçamento direto, endereço negativo
10 e 11	booleano, endereçamento direto, endereço positivo
12	booleano, endereçamento indireto
13	caráter, endereçamento direto, endereço negativo
14	caráter, endereçamento direto, endereço positivo
15	caráter, endereçamento indireto

TAM - tamanho do operando (número de caracteres, se o operando for caráter ou 1, em caso contrário)

END - endereço do operando

RBASE - número da linha da matriz de registradores base, a ser usada no cálculo do endereço efetivo do operando

No caso de endereçamento direto, END contém o deslocamento do operando, RBASE é a base relativa ao operando e a coluna da matriz de registradores base a ser usada será 1, 2, 3 ou 4 (conforme o tipo do dado: inteiro, real, booleano ou caráter) para endereços positivos e 5, 6, 7 ou 8 (conforme o tipo do dado: inteiro, real, booleano ou caráter) para endereços negativos.

No caso de endereçamento indireto, END contém o deslocamento da posição de memória inteira que contém o endereço absoluto do operando. RBASE é a base relativa a esta posição de memória e a coluna a ser usada da matriz de registradores base será sempre 5.

Os dados são impressos obedecendo ao posicionamento relativo a cada tipo de comando WRITE (ver comandos de entrada e saída na descrição da linguagem).

30) CDE - calcula deslocamento de elemento de array

formato: CDE (P1),(P2),(B3,D3,-)

onde:

P1 - ponteiro (endereço absoluto) para a lista descritora do array

P2 - ponteiro (endereço absoluto) para a lista de descritores dos índices

Esta instrução calcula o deslocamento de um elemento (especificado pela lista apontada por P2) de um array (especificado pela lista apontada por P1) com relação ao início deste array, colocando-o no operando 3.

Para cálculo do deslocamento do elemento é utilizada a seguinte fórmula:

$$(d_1-1)\ D^2D^3\ldots D^{n-1}D^n + (d_2-1)\ D^3\ldots D^{n-1}D^n + \ldots + (d_{n-1}-1)D^n + d_n$$

que pode ser reduzida a:

$$d_1D^2D^3\ldots D^{n-1}D^n + d_2D^3\ldots D^{n-1}D^n + \ldots + d_{n-1}D^n + d_n -$$

$$(D^2D^3\ldots D^{n-1}D^n + D^3\ldots D^{n-1}D^n + \ldots + D^n),$$

onde:

D^i - tamanho da i-ésima dimensão do array

d_i - deslocamento do elemento dentro da i-ésima dimensão

Sendo, por exemplo, a declaração de um array:

VAR A : ARRAY [$i_1:j_1, i_2:j_2, \ldots, i_k:j_k, \ldots, i_{n-1}:j_{n-1}, i_n:j_n$]
OF tipo,

e a referência a um de seus elementos:

A [a_1 , a_2 , a_3 , ... , a_k , ... , a_{n-1} , a_n]

temos:

$d_k = a_k - (i_k - 1)$

$D^k = j_k - (i_k + 1)$

A lista descritora do array está na memória inteira positiva e tem o seguinte formato:

n	LI_1-1	C_1	LI_2-1	C_2	...	LI_k-1	C_k	...	LI_{n-1}-1	C_{n-1}	LI_n-1	C_n

onde:

n = número de dimensões do array

LI_k = limite inferior da k-ésima dimensão

C_k = produto $D^{k+1}\ D^{k+2}\ \ldots D^{n-1}\ D^n$

C_n = $-(D^2D^3\ldots D^{n-1}D^n + D^3\ldots D^{n-1}D^n + \ldots + D^n)$ para n>1, ou

0 para n=1

A lista de descritores dos índices está na memória inteira positiva e existem 3 entradas para cada índice nesta lista:

TEND	END	RBASE

TEND - indica o tipo de endereçamento do índice, conforme tabela abaixo (todos os índices estão na memória inteira):

TEND	Descrição
1	endereçamento direto, endereço negativo
2 e 3	endereçamento direto, endereço positivo
4	endereçamento indireto

END - endereço do índice

RBASE - número da linha da matriz de registradores base a ser usada no cálculo do endereço efetivo do índice

No caso de endereçamento direto, END contém o deslocamento do índice, RBASE é a base relativa ao índice e a coluna a ser usada da matriz de registradores base será 1 (no caso de endereço positivo) ou 5 (no caso de endereço negativo).

No caso de endereçamento indireto, END contém o deslocamento da posição de memória inteira que contém o endereço absoluto do índice. RBASE é a base relativa a esta posição de memória e a coluna a ser usada da matriz de registradores base será 5.

Esta instrução, do mesmo modo que as instruções READ e WRITE, está bastante distanciada da realidade. Sua definição teve por objetivo a simplificação na parte de implementação, sendo porém viável a transformação para instruções de uma máquina real, num passo complementar.

2.3.1.5. IMPLEMENTAÇÃO DA MÁQUINA VIRTUAL

A execução do programa gerado pelo compilador é feita por uma rotina que interpreta diretamente as quádruplas.

É uma rotina completamente isolada do compilador. Não usa suas variáveis e é ligada a êle somente por um CALL.

O programa objeto para a máquina virtual é passado do compilador para o interpretador através de um arquivo.

2.3.2. ESQUEMA GERAL DE GERAÇÃO DE CÓDIGO

2.3.2.1. ORGANIZAÇÃO DA MEMÓRIA

Conforme foi dito anteriormente, existem na máquina virtual 5 tipos de memória: uma para armazenar as instruções do programa e uma para cada tipo de dado.

A memória de instruções é usada pelo compilador de maneira estática, isto é, apesar dos procedimentos e funções (do programa do usuário) serem recursivos, existirá em fase de execução apenas uma cópia de cada uma.

Cada memória de dado é dividida pelo compilador em 3 partes:

- os endereços negativos são usados como uma área dinâmica para armazenar os temporários gerados pelo compilador;
- os últimos endereços positivos são usados como uma área dinâmica para armazenar as variáveis do programa do usuário e
- os primeiros endereços positivos são tratados como uma área estática, usada para as constantes (do programa do usuário e as geradas pelo compilador) e para áreas de comunicação entre os blo-

cos do programa objeto (programa principal, procedimentos e funções).

A atribuição de endereços para as instruções e os dados do programa sendo compilado é feita pelo compilador dentro do seguinte esquema:

- as instruções recebem endereços absolutos, consecutivamente a partir de um, na ordem de suas ocorrências;
- os elementos da área estática (primeiros endereços positivos) de cada memória de dado recebem endereços absolutos consecutivamente a partir de um;
- a cada bloco do programa sendo compilado são associados pelo compilador 4 áreas para temporários e 4 para variáveis, uma para cada tipo de dado. Para cada elemento dentro dessas áreas, é atribuido como endereço o deslocamento com relação ao início da área respectiva.

O valor utilizado como deslocamento para o primeiro elemento de cada área (variável e temporário) é prefixado e é o mesmo para todos os blocos. No presente caso foram adotados deslocamento inicial -1 para os temporários e 1001 para as variáveis. As áreas de temporários crescem no sentido negativo e as de variáveis no sentido positivo.

O valor do deslocamento inicial das variáveis define o tamanho máximo permitido para a área estática - 1000 posições no presente caso.

Como, para todos os blocos, as origens das áreas de dados são as mesmas, existe um conflito devido à superposição de endereços, representados pelos deslocamentos atribuidos aos dados. Este problema é resolvido pela utilização dos registradores base.

Este esquema permite o reaproveitamento das áreas de dados. As áreas de dados usadas por um bloco são liberadas ao final da execução deste, ficando disponíveis para outros blocos.

Em fase de execução do programa compilado, a área de temporários, a área estática e a área de variáveis ocupam posições contíguas dentro de cada memória. A justaposição destas áreas é obtida pela utilização dos registradores base.

2.3.2.2. ENDEREÇAMENTO

Conforme dito anteriormente, para cada tipo de dado estão associadas duas colunas da matriz de registradores base: uma para os endereços positivos e outra para os negativos.

O compilador utiliza as colunas referentes aos endereços negativos como registradores base para os temporários e as demais para as variáveis.

A tabela abaixo resume a situação:

Número da coluna	Tipo de dado
1	variável inteira
2	variável real
3	variável booleana
4	variável caráter
5	temporário inteiro
6	temporário real
7	temporário booleano
8	temporário caráter

Todos os procedimentos e funções do programa sendo compilado estão embutidos no programa principal, isto é, são definidos no programa principal. Além disso, em decorrência da estrutura da linguagem, pode ocorrer a existência de procedimentos e funções embutidos em outros procedimentos e funções.

O conceito de nível traduz a posição de um procedimento ou função, com relação aos demais. Assim, o programa principal é considerado de nível 1. Os procedimentos e funções definidos dentro de um procedimento ou função de nível n tem como nível n+1.

A cada nível o compilador associa uma linha da matriz de registradores base. Portanto, ao programa principal está associada a linha um e aos procedimentos e funções de nível i está associada a linha i. Isto significa que aos temporários gerados num procedimento ou função de nível i, bem como às variáveis

neles definidas estão associados os registradores base da linha i. Portanto os registradores base desta linha é que serão usados no endereçamento destes temporários e variáveis durante a execução do procedimento ou função de nível i e de todos os demais nele embutidos, sendo a linha liberada ao final da execução deste procedimento ou função, ficando disponível para outros procedimentos ou funções de mesmo nível.

Diversos procedimentos e funções de mesmo nível podem estar ativos simultaneamente, isto é, a execução de cada um deles foi iniciada e ainda não foi encerrada. O trecho de programa abaixo exemplifica esta situação:

```
PROGRAM A ; % PROGRAMA PRINCIPAL - NIVEL 1 %
  PROC B ; % PROCEDIMENTO DE NIVEL 2 %
    BEGIN
      .........
      .........
    END ;
  PROC C ; % PROCEDIMENTO DE NIVEL 2 %
    BEGIN
      .........
      .........
      B ; % ATIVA PROCEDIMENTO B %
      .........
      .........
    END ;
  BEGIN
    .........
    .........
    C ; % ATIVA PROCEDIMENTO C %
    .........
    .........
  END.
```

Apesar de estarem ativos simultaneamente, os procedimentos B e C não são executados ao mesmo tempo. A execução do procedimento C é suspensa quando o procedimento B é ativado e só continua após a execução de B.

Como as variáveis e temporários do procedimento C não podem ser diretamente referenciadas no procedimento B, o empilhamento da matriz de registradores base resolve o problema.

As fórmulas abaixo dão os valores dos registradores base para cada bloco, em fase de execução:

$B_x = T_x + 1 - V_x$, para variáveis e

$B_x = - T_x - 1 - V_x$, para temporários

onde:

x - indica a coluna da matriz de registradores base ($1 \leq x \leq 8$);
T - tamanho da área já ocupada pelos blocos externos ao corrente = tamanho da área estática + Σ áreas de variáveis se $x \leq 4$ ou Σ áreas de temporários se $x > 4$;
V - valor usado para deslocamento inicial dos temporários (se $x > 4$) ou das variáveis (caso contrário). No presente caso esses valores são, respectivamente, -1 e 1001.

Observação: a linha da matriz é determinada pelo nível do bloco.

De modo geral, o endereço efetivo de um operando é calculado pela soma de 3 parcelas: o conteúdo de um registrador base, o deslocamento e o conteúdo do registrador indexador correspondente.

A tabela a seguir resume a especificação básica destas parcelas, por tipo de operando:

TIPO DO OPERANDO	VALOR DO DESLOCAMENTO	NÚMERO DO REG. BASE	CONTEÚDO DO REG. INDEXADOR
constante	endereço da constante	0	0
variável simples; parâmetro variável simples por valor; temporários	deslocamento associado ao operando	nível do bloco onde foi definido o operando	0
elemento de variável matricial ou de parâmetro variável matricial por valor com índice constante	deslocamento associado à matriz + deslocamento do elemento em relação ao início da matriz	nível do bloco onde foi definida a matriz	0
elemento de variável matricial ou de parâmetro variável matricial por valor com índice variável	deslocamento associado à matriz	nível do bloco onde foi definida a matriz	deslocamento do elemento com relação ao início da matriz
parâmetro variável simples por endereço	0	0	endereço absoluto do parâmetro
elemento de parâmetro variável matricial por endereço (com índice constante ou não)	0	0	endereço absoluto da matriz + deslocamento do elemento com relação ao início da matriz

2.3.2.3. PRÓLOGO E EPÍLOGO

Para cada bloco (programa principal, procedimentos e funções) é gerado um prólogo e, com exceção do programa principal, um epílogo.

No prólogo são geradas instruções para:

1) Carregar os valores dos registradores base para variáveis e temporários do bloco. Estes valores, calculados segundo a fórmula mencionada anteriormente, são mantidos na área estática inteira nas posições 3 a 10;

2) Atualizar o valores a serem usados como base para o próximo bloco interno a este. Isto equivale a alocar área para as variáveis e temporários do bloco. A atualização destes valores é feita incrementando-se as posições 3 a 10 da área estática inteira com os tamanhos respectivos das áreas de variáveis e temporários deste bloco;

3) Salvar endereço de retorno, no caso de procedimento ou função, copiando-o para o primeiro temporário inteiro do procedimento ou função;

4) Salvar os endereços dos parâmetros por endereço, copiando-os para temporários inteiros do procedimento ou função e os valores dos parâmetros por valor, copiando-os para temporários de mesmo tipo do parâmetro no caso de procedimento ou função com parâmetros;

5) Salvar o endereço do temporário que conterá o valor da função, no caso de função, copiando-o para um temporário inteiro da função.

No epílogo são geradas instruções para:

1) Liberar áreas de variáveis e temporários do bloco, decrementando as posições 3 a 10 da área estática inteira dos tamanhos respectivos das áreas de variáveis e temporários do bloco;

2) Retorno do procedimento ou função.

2.3.2.4. CHAMADA DE PROCEDIMENTO E FUNÇÃO

O seguinte protocolo é observado na geração de código para chamada de procedimentos e funções.

- é montada uma lista, denominada lista de chamada de procedimento e função;
- é comunicado ao procedimento ou função chamado o endereço absoluto desta lista, através de sua área de ligação;
- se o procedimento ou função chamado for de nível menor ou igual ao do bloco que contém a chamada é gerada instrução para empilhar a matriz de registradores base e
- é gerado um desvio para a primeira instrução do procedimento ou função chamado.

O conteúdo da lista de chamada é o seguinte:

endereço de retorno	endereço do resultado	parâmetro 1	....	parâmetro i	...	último parâmetro

onde:

endereço de retorno - é o endereço da primeira instrução após as geradas para a chamada do procedimento ou função

endereço do resultado - este campo só existe no caso de chamada de função. É o endereço do temporário que conterá o resultado da função

parâmetro i - se este parâmetro for um procedimento (ou função) este campo será subdividido em 3: endereço (menos 1) da primeira instrução do procedimento (ou função) passado como parâmetro; endereço de sua área de ligação e seu nível. Nos demais casos este campo conterá o endereço absoluto do parâmetro

A cada procedimento ou função é associada uma posição na área estática inteira. Esta posição recebe o nome de área de ligação e é usada para comunicar ao procedimento (ou função) o endereço absoluto da lista de chamada.

2.4. ESTRUTURA DE DADOS

Neste tópico são descritas as estruturas globais utilizadas no processo de compilação, indicando sua função, sua organização e modo de utilização.

2.4.1. TABELA DE SÍMBOLOS

Sua função é armazenar os símbolos definidos no programa sendo compilado, juntamente com seus atributos.

A tabela de símbolos é composta de um vetor cabeça de "hash"(CABEC), uma estrutura de atributos(TABELA) e um apontador(AVAIL).

As entradas em TABELA são ocupadas sequencialmente, a partir da primeira, a medida que os símbolos vão aparecendo no programa fonte. As entradas que possuem o mesmo " home address", ou seja, a função "hash" produziu o mesmo resultado quando aplicada a cada uma delas, estão ligadas em uma lista que será apontada pelo elemento do vetor CABEC cujo índice é o valor da função "hash". Tal lista é encadeada em ordem invertida, isto é, o elemento de CABEC aponta para o último identificador inserido na lista.

Descrição dos componentes da tabela:

CABEC - vetor cabeça de "hash", inicializado com todas as entradas iguais a 0, para indicar listas vazias. Cada elemento de CABEC aponta para uma lista de entradas encadeadas de TABELA que possuem o mesmo "home address".

TABELA - estrutura de atributos, contendo cada elemento os seguintes campos:

- . NOME - de tamanho 8, contém o identificador. Se este tiver menos que 8 caracteres será completado com brancos à direita.
- . NÍVEL - contém o número do bloco a que pertence a declaração do símbolo.
- . LINK - é o campo de ligação da lista de entradas de mesmo "home address". Cada elemento contém o índice da entrada de TABELA que o precede em sua lista, ou 0, para indicar fim de lista.

. IDTIPO - é o código do tipo do identificador, conforme tabela contida na descrição da macro PLAMV04.

. PTCTE - ponteiro para tabela de constantes, tabela de dimensões de arrays ou para tabela de lista de parâmetros, se o identificador for uma constante, um array ou um procedimento ou função, respectivamente. Mais detalhes sobre a interpretação do conteúdo deste campo são dados na descrição das tabelas referidas(macros PLAMV07,08,09). No caso de identificador de constante booleana, PTCTE conterá o valor da constante (1 ou 0, respectivamente TRUE ou FALSE). No caso de rótulo, este campo é usado para detetar sua existência como rótulo de mais de um comando no mesmo bloco e para indicar referência ao rótulo antes de sua definição, isto é, referência antes dele aparecer como rótulo de um comando.

. ENDER - endereço correspondente ao símbolo. O conteúdo deste campo depende do tipo do símbolo, conforme tabela abaixo:

TIPO	CONTEÚDO
constante,variável ou parâmetro variável por valor	apontador para área de memória correspondente ao tipo do símbolo (inteiro, real, booleano ou caráter)
rótulo	endereço do comando
procedimento ou função	endereço da primeira instrução do procedimento ou função
parâmetro variável por endereço	endereço do temporário que conterá, em fase de execução, o endereço absoluto do parâmetro real
parâmetro procedimento ou parâmetro função	endereço da lista de temporários que conterá, em fase de execução, a descrição do procedimento ou função

. AVAIL - apontador da lista de espaços disponíveis. Contém o índice de TABELA a partir do qual todas as entradas que o seguem (inclusive AVAIL) estão disponíveis. É inicializado com 1 para indicar tabela vazia.

Durante a análise do programa fonte, o número do bloco é inicialmente 0. Cada vez que um novo bloco é aberto, o número do bloco é incrementado de 1, e cada vez que um bloco é fechado, todos os identificadores referentes a ele são retirados da tabela(pois não são definidos nos blocos mais externos e portanto perdem seu significado no programa) e o número do bloco é decrementado de 1. É considerado "bloco" um procedimento ou uma função(o programa principal é considerado um procedimento). Desta forma é aberto um novo bloco no início de um procedimento ou função, e é fechado o bloco no END final do procedimento ou função (o programa principal é o bloco 1).

A retirada dos identificadores referentes aos blocos já fechados permite que se tenha na tabela a cada instante somente os identificadores válidos naquele instante, dispensando o teste de validade de escopo na busca de um identificador.

O encadeamento invertido das listas de identificadores agrupa no começo da lista os identificadores referentes ao último bloco válido, o que diminue o tempo de busca destes identificadores e facilita a retirada dos referentes aos blocos fechados.

2.4.2. TABELAS DE CONSTANTES

Estas estruturas correspondem às áreas estáticas do programa sendo compilado. Sua função é armazenar, durante a compilação, as constantes encontradas no programa fonte ou geradas pelo compilador .

Existem 3 tabelas: uma para constantes inteiras, outra para constantes reais e a terceira para constantes alfanuméricas.

As entradas das tabelas são ocupadas em ordem sequencial, a partir da primeira.

As duas primeiras são mantidas em listas encadea-

das em ordem crescente. Antes da inserção de cada constante numérica(inteira ou real) é feita uma busca na tabela correspondente. Se a constante já se encontra lá, não é feita nova inserção.

A tabela de constantes inteiras é utilizada também para armazenar as listas auxiliares geradas pelo compilador para execução do programa sendo compilado, tais como lista para os comandos de entrada e saída, lista descritora de arrays, etc.. Estas listas não fazem parte do encadeamento normal da tabela pois suas entradas precisam ocupar posições contíguas.

A tabela de constante alfanuméricas é composta de 3 vetores: PTALFA, TAMALFA e TEXTO. TEXTO contém os literais. Cada elemento de PTALFA aponta para o início de um literal (em TEXTO) e o elemento correspondente de TAMALFA dá o tamanho do literal em caracteres. Uma variável auxiliar indica, a cada instante, a última posição ocupada de PTALFA(que é a mesma para TAMALFA).

2.4.3. TABELA DE DIMENSÕES DE ARRAYS

Sua função é armazenar as informações referentes aos arrays declarados no programa sendo compilado. Esta tabela é composta de um vetor(TABDIM) que contém estas informações e um apontador para a próxima entrada disponível na tabela.

São armazenadas nesta tabela, para cada array, as seguintes informações: o número de dimensões do array, o número de elementos, um apontador para uma lista na área estática inteira e os limites inferior e superior de cada dimensão do array.

A lista na área estática inteira contém o número de elementos do array e as constantes necessárias para cálculo do endereço de cada elemento do array, em fase de execução do programa, quando houver referência ao elemento.

2.4.4. TABELA DE PARÂMETROS

Sua função é armazenar a lista descritora dos parâmetros de cada procedimento ou função declarado no programa sendo compilado. A tabela é composta de um vetor(TABPARM) que contém esta lista e um apontador para a próxima entrada disponível na tabela.

São armazenadas nesta tabela, para cada procedimento

ou função, as seguintes informações: o endereço da área de ligação, o número de parâmetros e o tipo de cada um deles (conforme tabela contida na macro PLAMV09). No caso de um parâmetro array, após seu tipo estão também na tabela o número de dimemsões, o número de elementos e os limites inferior e superior de cada dimensão.

Área de ligação de um procedimento ou função é uma posição da área estática inteira que é usada em fase de execução para conter o endereço absoluto da lista de chamada do procedimento ou função.

2.4.5. ARQUIVO OBJ1

Sua função é armazenar as quádruplas geradas durante a compilação. É um arquivo sequencial, com registros de tamanho fixo, de 80 bytes. Cada registro contém 5 quádruplas de 16 bytes, com exceção do último registro, que pode não estar todo ocupado.

2.4.6. ARQUIVO OBJ2

Sua função é armazenar o programa em formato executável. É um arquivo sequencial, com registros de tamanho fixo, de 80 bytes.

O primeiro registro contém os tamanhos das áreas necessárias para execução do programa compilado. Tem o seguinte formato:

|cod|CINT|CREAL|CBOOL|CCHAR|TINT|TREAL|TBOOL|TCHAR|NCINT|NCREAL|

|NCBOOL|NCCHAR|NQUAD|NBMAX|

onde:

cod - é o código para indicar registro com tamanho das áreas.
CINT - tamanho da área para constantes e variáveis inteiras.
CREAL - tamanho da área para constantes e variáveis reais.
CBOOL - tamanho da área para constantes e variáveis booleanas.
CCHAR - tamanho da área para constantes e variáveis caráter (em caracteres).
TINT - tamanho da área para temporários inteiros.

TREAL - tamanho da área para temporários reais.
TBOOL - tamanho da área para temporários booleanos.
TCHAR - tamanho da área para temporários caráter (em caracteres).
NCINT - número de constantes inteiras.
NCREAL - número de constantes reais.
NCBOOL - número de constantes booleanas.
NCCHAR - número de constantes caráter (em caracteres).
NQUAD - tamanho da área para instruções (número de quádruplas).
NBMAX - tamanho da matriz de registradores base (número de linhas da matriz).

Em seguida há um ou mais registros para cada tipo de constante gerada, conforme o tipo delas. Cada tipo de constante inicia um novo registro, sendo abandonado o resto do anterior.

Formato dos registros contendo as constantes:

cod	NC	C_1	C_2	.....	C_i	.....	C_n

onde:
cod - código para indicar registro com constantes inteiras, reais, booleanas ou caráter (um código para cada tipo).
NC - número de constantes geradas (inteiras, reais, booleanas ou caráter, dependendo do valor de cod).
C_i - valor da i-ésima constante.
Observação: este formato é valido apenas para o primeiro registro de cada grupo. Para os demais registros do grupo não existem os campos cod e NC.

A partir do próximo registro livre estão as instruções (copiadas do arquivo OBJ1), 5 em cada registro. O resto do último registro com as constantes é abandonado.

2.4.7. ARQUIVO SCARDS

Sua função é armazenar o texto do programa a ser compilado e os dados para execução deste programa.

É um arquivo sequencial com registros de tamanho fixo de 80 "bytes". Separando o programa fonte dos seus dados de

ve existir um registro com o caráter $ na primeira posição. Este registro poderá ser omitido no caso de inexistência de dados para a execução do programa.

2.4.8. ARQUIVO SPRINT

O arquivo SPRINT é o arquivo de impressão tanto da fase de compilação como de execução.

É um arquivo sequencial com registros de tamanho fixo de 133 "bytes", sendo que o primeiro contém o caráter para controle da impressora, na modalidade ASA.

2.4.9. TABELA DE MENSAGENS DE ERROS DE COMPILAÇÃO

A função desta tabela é armazenar as mensagens de erros de compilação.

É implementada através de uma tabela dupla, interna à rotina para impressão de mensagens de erro (PLAPP01).

A primeira parte, o vetor de estruturas MENS, é composta por 2 campos:

INICIO - campo alfanumérico de 53 caracteres que contém a parte inicial do texto da mensagem;

SEG - apontador para a continuação da mensagem, na segunda parte da tabela, se houver.

A segunda parte, o vetor de estruturas COMP, é composta tambem por 2 campos:

COMPLEM - campo alfanumérico de 53 caracteres que contém as complementações do texto da mensagem;

TER - apontador para a continuação da mensagem, nesta mesma parte da tabela, se houver.

Um apontador com valor zero, tanto na primeira como na segunda parte da tabela, indica o fim do texto da mensagem.

2.4.10. TABELA DE MENSAGENS DE EXECUÇÃO

A função desta tabela é armazenar as mensagens a serem emitidas durante a execução do programa compilado.

É implementada através de uma tabela dupla, interna à rotina para impressão de mensagens durante a execução (PLAPP33), idêntica à Tabela de Mensagens de Erros de Compilação descrita no tópico anterior.

3. PROGRAMAÇÃO

3.1. DIRETRIZES

A programação foi feita na linguagem PL1 por ser uma linguagem bastante difundida e possuir razoável capacidade para manipulação de textos.

Foi adotada uma estrutura modular e foram evitados os "defaults" e as sofisticações do PL1. Foi usado o PL1-FULL. A eficiência foi muitas vêzes sacrificada em função da clareza.

De modo geral, as variáveis comuns são declaradas como globais ao invés de passadas como parâmetros para as subrotinas.

As declarações das variáveis, tabelas, estruturas e subrotinas externas estão em macros separadas, que são incluidas nas rotinas que as utilizam. Foi incluida em cada macro, sob forma de comentário, a relação das rotinas que utilizam esta macro.

Foram incluidas umas poucas facilidades para auxiliar na depuração do compilador, como impressão do fluxo e, em alguns casos, das variáveis principais de cada rotina e impressão das tabelas. Estas facilidades são ativadas por parâmetros do compilador.

O compilador é composto de um programa principal, rotinas de análise e geração de código, rotinas de apoio, rotinas para execução do programa compilado e macros que definem as variáveis comuns, as tabelas e as rotinas externas.

O programa principal é PLAPP00.

As rotinas de análise e geração de código são:

PLAPP08 - Reconhecedor de Bloco
PLAPP09 - Reconhecedor de Declaração de Constantes
PLAPP10 - Reconhecedor de Declaração de Variáveis
PLAPP11 - Reconhecedor de Cabeçalho de Procedimento
PLAPP12 - Reconhecedor de Cabeçalho de Função
PLAPP17 - Reconhecedor de Comandos

PLAPP18 - Reconhecedor de Tipo de Variáveis
PLAPP19 - Reconhecedor de Expressão Tipo 1
PLAPP20 - Reconhecedor de Seção de Parâmetros Formais
PLAPP21 - Reconhecedor de Comando "REPEAT"
PLAPP22 - Reconhecedor de Comando "WHILE"
PLAPP23 - Reconhecedor de Comando "IF"
PLAPP24 - Reconhecedor de Comando "CASE"
PLAPP25 - Reconhecedor de Comando "FOR"
PLAPP26 - Reconhecedor de Comando "READ"
PLAPP27 - Reconhecedor de Comando "WRITE"
PLAPP28 - Reconhecedor de Seção de Parâmetros Reais
PLAPP29 - Reconhecedor de Índices de Arrays
PLAPP30 - Reconhecedor de Expressão

As rotinas de apoio são:

PLAPA03 - Cálculo da Função HASH
PLAPP01 - Impressão de Mensagens de Erros
PLAPP02 - Impressão da Tabela de Símbolos
PLAPP03 - Reconhecedor de Opções de Compilação
PLAPP04 - Reconhecedor Léxico
PLAPP05, PLAPP06 e PLAPP07 - Manipulação da Tabela de Símbolos
PLAPP13, PLAPP14 e PLAPP15 - Manipulação das Tabelas de Constantes
PLAPP31 - Rotina para Gerar o Programa Objeto em Disco
PLAPP34 - Rotina para Gravar o Registro de Instruções

As rotinas para execução do programa compilado são:

PLAPP32 - Rotina para Executar o Programa Compilado
PLAPP33 - Rotina para Imprimir Mensagens durante a Execução

As macros são as seguintes:

MACRO	DEFINE
PLAMA03	PLAPA03 (procedimento externo)
PLAMP01	PLAPP01 (procedimento externo)
PLAMP02	PLAPP02 (procedimento externo)
PLAMP03	PLAPP03 (procedimento externo)
PLAMP04	PLAPP04 (procedimento externo)
PLAMP05	PLAPP05 (procedimento externo)

MACRO	DEFINE
PLAMP06	PLAPP06 (entry-point da PLAPP05)
PLAMP07	PLAPP07 (entry-point da PLAPP05)
PLAMP08	PLAPP08 (procedimento externo)
PLAMP09	PLAPP09 (procedimento externo)
PLAMP10	PLAPP10 (procedimento externo)
PLAMP11	PLAPP11 (procedimento externo)
PLAMP12	PLAPP12 (procedimento externo)
PLAMP13	PLAPP13 (procedimento externo)
PLAMP14	PLAPP14 (entry-point da PLAPP13)
PLAMP15	PLAPP15 (entry-point da PLAPP13)
PLAMP17	PLAPP17 (procedimento externo)
PLAMP18	PLAPP18 (procedimento externo)
PLAMP19	PLAPP19 (procedimento externo)
PLAMP20	PLAPP20 (procedimento externo)
PLAMP21	PLAPP21 (procedimento externo)
PLAMP22	PLAPP22 (procedimento externo)
PLAMP23	PLAPP23 (procedimento externo)
PLAMP24	PLAPP24 (procedimento externo)
PLAMP25	PLAPP25 (procedimento externo)
PLAMP26	PLAPP26 (procedimento externo)
PLAMP27	PLAPP27 (procedimento externo)
PLAMP28	PLAPP28 (procedimento externo)
PLAMP29	PLAPP29 (procedimento externo)
PLAMP30	PLAPP30 (procedimento externo)
PLAMP31	PLAPP31 (procedimento externo)
PLAMP32	PLAPP32 (procedimento externo)
PLAMP33	PLAPP33 (procedimento externo)
PLAMP34	PLAPP34 (procedimento externo)
PLAMV01	Estrutura AUXSCAN
PLAMV02	Variáveis BLOCO e BLOCMAX
PLAMV03	Variável EOF
PLAMV04	Tabela de Símbolos (TABSIMB)
PLAMV05	Variável TRACE
PLAMV06	Variáveis ERRO e COLUNA

MACRO DEFINE

PLAMV07 - Tabelas de Constantes:
TABINT - constantes inteiras
TABREAL - constantes reais
TABALFA - constantes alfanuméricas

PLAMV08 - Tabela de Dimensões de Arrays (TABDIM) e seu Ponteiro (PROXDIM)

PLAMV09 - Tabela de Parâmetros Formais (TABPARM) e seu Ponteiro (PRXPARM)

PLAMV10 - Variáveis TIPEXP, PTMAT e TAMCHAR

PLAMV11 - Variável GERA e Estrutura AUXGERA

PLAMV12 - Variáveis EXEC e LIST

PLAMV13 - Arquivo OBJ1

PLAMV14 - Arquivo OBJ2

PLAMV15 - Estrutura CODOPER

PLAMV16 - Estrutura REGAREA

PLAMV18 - Arquivo SCARDS

PLAMV19 - Arquivo SPRINT

Observação: na parte restante deste capítulo, em cada rotina de análise sintática, a sequência a ser analisada pela mesma está sendo apresentada por meio de uma forma simplificada da BNF, isto é, sem usar de todos os seus formalismos. Além disso, a seguinte notação complementar foi adotada:

$$\{A\}_a^b$$

indicando que o elemento A pode ocorrer de um número mínimo de vezes (a) até um número máximo de vezes (b). O limite máximo (b) igual a n indica que este limite é dependente da implementação.

3.2. DESCRIÇÃO DE CADA MÓDULO

3.2.1. PROGRAMA PRINCIPAL - PLAPP00

ANÁLISE

Sua função é reconhecer o cabeçalho do programa fonte e, ao término da compilação, a finalização correta do mesmo (se todos os blocos foram fechados e se foi encontrado o fim do arquivo) e imprimir os erros de finalização, se houver.

Sequência a ser analisada:

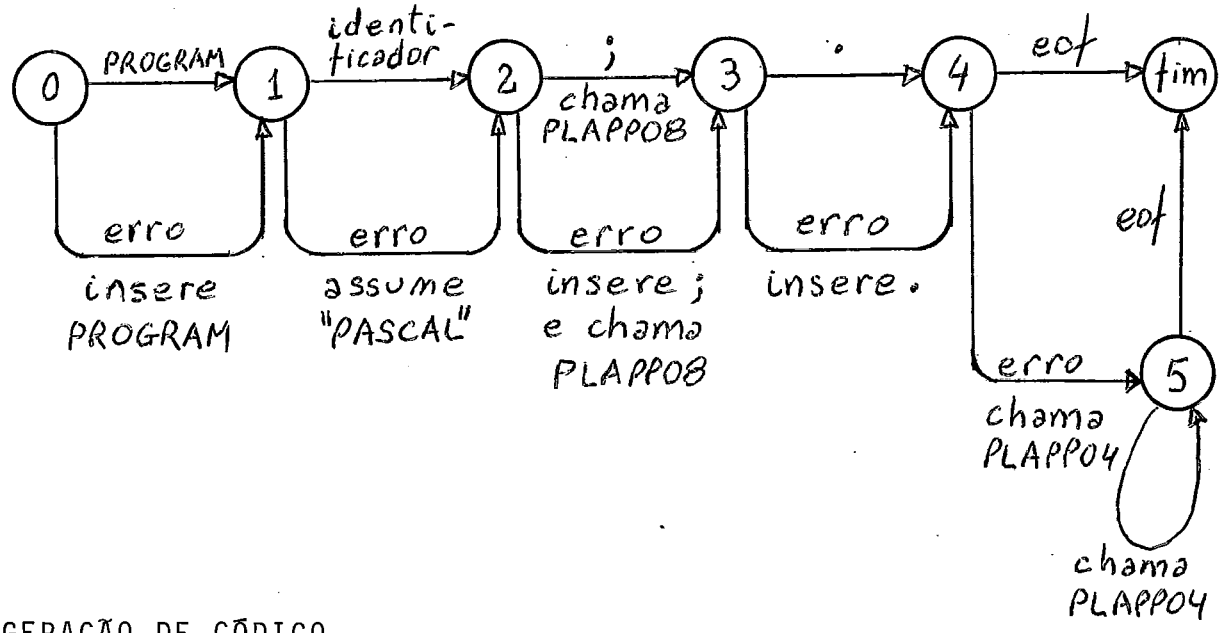
PROGRAM <identificador> ; <bloco> .

onde:

<identificador> é o nome do programa.

O bloco é analisado pelo Reconhecedor de Bloco (PLAPP08).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

Esta rotina gera o prólogo do programa principal e a instrução EXIT após a última instrução do programa. Grava o último registro de instruções, chama a rotina para gerar o programa objeto em disco (PLAPP31), se não houve erro na compilação.

3.2.2. ROTINAS DE ANÁLISE E GERAÇÃO DE CÓDIGO

3.2.2.1. RECONHECEDOR DE BLOCO - PLAPP08

ANÁLISE

A função desta rotina é reconhecer os blocos do programa principal e dos procedimentos e funções.

É chamada pelo Programa Principal (PLAPP00) e por si mesma com o primeiro elemento do bloco lido (o símbolo que segue o BEGIN) e retorna com TIPO contendo o código correspondente ao símbolo que segue o END que encerra o bloco, ou com fim de arquivo.

Sequência a ser analisada:

{ LABEL <identificador> { , <identificador> }$_0^n$; }$_0^1$

{ CONST <lista de definição de constantes> }$_0^1$

{ VAR <lista de definição de variáveis> }$_0^1$

{ { PROC <cabeçalho de procedimento> <bloco> }$_0^n$

{ FUNCTION <cabeçalho de função> <bloco> }$_0^n$ }$_0^n$

BEGIN { <comando> }$_0^1$ { ; { <comando> }$_0^1$ }$_0^n$ END

A lista de definição de constantes é analisada pelo Reconhecedor de Declaração de Constantes (PLAPP09); a lista de definição de variáveis pelo Reconhecedor de Declaração de Variáveis (PLAPP10); o cabeçalho de procedimento pelo Reconhecedor de Cabeçalho de Procedimento (PLAPP11); o cabeçalho de função pelo Reconhecedor de Cabeçalho de Função (PLAPP12); os comandos pelo Reconhecedor de Comandos (PLAPP17) e os blocos por esta mesma rotina (PLAPP08).

A variável BLOCO, que representa o nível do bloco sendo analisado, é incrementada de um no início da análise do bloco e decrementada de um ao final.

Os elementos inseridos nas tabelas de dimensões de arrays, de parâmetros formais e de símbolos, durante a análise do bloco, são retiradas ao final da mesma.

GERAÇÃO DE CÓDIGO

Ao ser encontrada a declaração do primeiro procedimento ou função do bloco é gerada uma instrução de desvio indexado para pular os procedimentos e funções deste bloco. O endereço de desvio é o correspondente ao comando BEGIN do bloco que está sendo analisado.

Nas declarações de procedimentos e funções, após a análise do bloco correspondente é gerado um epílogo contendo instruções para liberar áreas de variáveis e temporários do procedimento ou função, instruções para retorno do procedimento ou função e são geradas constantes com os tamanhos das áreas para variáveis e temporários do procedimento ou função.

3.2.2.2. RECONHECEDOR DE DECLARAÇÃO DE CONSTANTES - PLAPP09

ANÁLISE

A função desta rotina é analisar a declaração das constantes do programa fonte.

É chamada pelo Reconhecedor de Bloco (PLAPP08)com TIPO contendo o código correspondente à palavra reservada CONST e retorna com o TIPO correspondente ao símbolo seguinte ao ";" final da declaração, ou fim de arquivo.

Sequência a ser analisada:

CONST <identificador> $\{$, <identificador> $\}_0^n$ = <constante>

$\{$, <identificador> $\{$, <identificador> $\}_0^n$ = <constante> $\}_0^n$;

onde:
<constante> é uma constante ou um identificador de constante já declarado.

À medida que vão sendo encontrados os identificadores, os mesmos são inseridos na tabela de símbolos e suas posições nesta tabela vão sendo guardadas num vetor (PILHA).

Ao ser encontrada a constante, a mesma é inserida na tabela de constantes correspondente ao seu tipo (TABINT,

TABREAL ou TABALFA) e as entradas da tabela de símbolos correspondentes aos identificadores são completadas com o tipo da constante, a posição da constante na tabela respectiva e o deslocamento atribuido à constante.

GERAÇÃO DE CÓDIGO

Esta rotina não gera instruções.

3.2.2.3. RECONHECEDOR DE DECLARAÇÃO DE VARIÁVEIS - PLAPP10

ANÁLISE

A função desta rotina é analisar a declaração das variáveis do programa fonte. É chamada pelo Reconhecedor de Bloco (PLAPP08) com TIPO contendo o código correspondente à palavra reservada VAR e retorna com o código do símbolo que segue o ";" final da declaração ou fim de arquivo.

Sequência a ser analisada:

VAR <identificador> $\{$, <identificador> $\}_0^n$: <tipo da variável>

$\{$, <identificador> $\{$, <identificador> $\}_0^n$:

<tipo da variável> $\}_0^n$;

O tipo da variável é analisado pelo Reconhecedor de Tipo de Variáveis (PLAPP18).

À medida que vão sendo encontrados os identificadores, os mesmos são inseridos na tabela de símbolos e suas posições nesta tabela vão sendo guardadas num vetor (PILHA).

Após a análise do tipo da variável as entradas da tabela de símbolos correspondentes aos identificadores são completadas com o tipo da variável, o deslocamento atribuido à variável e, no caso de array, o apontador para a tabela de dimensões de arrays.

GERAÇÃO DE CÓDIGO

Esta rotina não gera instruções.

3.2.2.4. RECONHECEDOR DE CABEÇALHO DE PROCEDIMENTO - PLAPP11

ANÁLISE

A função desta rotina é analisar cabeçalho de procedimentos. É chamada pelo Reconhecedor de Bloco (PLAPP08) com TIPO contendo o código da palavra reservada PROC e retorna com o código correspondente ao símbolo que segue o ";" que encerra o cabeçalho do procedimento ou com fim de arquivo.

Sequência a ser analisada:

PROC <identificador> $\{$ (<seção de parâmetros formais>) $\}_0^1$;

A seção de parâmetros formais é analisada pelo Reconhecedor de Seção de Parâmetros Formais (PLAPP20).

Ao ser encontrado o identificador, que dá o nome ao procedimento, o mesmo é inserido na tabela de símbolos com: tipo igual a nome de procedimento; NÍVEL igual ao valor corrente da variável BLOCO (após o que a variável BLOCO é incrementada de 1); endereço igual ao da próxima instrução a ser gerada e o apontador para a tabela de parâmetros.

Após a análise da seção de parâmetros formais, quando for o caso, é completada a tabela de parâmetros com o número de parâmetros reconhecidos.

GERAÇÃO DE CÓDIGO

Esta rotina reserva 9 posições contíguas na área estática inteira (para o endereço de ligação e para as constantes com os tamanhos das áreas de variáveis e temporários do procedimento) e coloca o endereço da primeira posição na tabela de parâmetros.

Gera a parte inicial do prólogo do procedimento contendo instruções: para carregar os valores dos registradores base para o procedimento; para atualizar os valores a serem usados como base para o próximo bloco interno (próximo procedimento ou função dentro deste bloco) e para salvar o endereço de retorno do procedimento.

3.2.2.5. RECONHECEDOR DE CABEÇALHO DE FUNÇÃO - PLAPP12

ANÁLISE:

A função desta rotina é analisar cabeçalho de funções.

É chamada pelo Reconhecedor de Bloco(PLAPP08) com TIPO contendo o código da palavra reservada FUNCTION e retorna com o código correspondente ao símbolo que segue o ";" que encerra o cabeçalho da função ou com fim de arquivo.

Sequência a ser analisada:

FUNCTION <identificador> {(<seção de parâmetros formais>) }$_{0}^{1}$

: <tipo da função> ;

A seção de parâmetros formais é analisada pelo Reconhecedor de Seção de Parâmetros Formais(PLAPP20).

Ao ser encontrado o identificador, que dá nome à função, o mesmo é inserido na tabela de símbolos com NÍVEL igual ao valor corrente da variável BLOCO(após o que a variável BLOCO é incrementada de 1), endereço igual ao da próxima instrução a ser gerada e o apontador para a tabela de parâmetros.

O nome da função é novamente inserido na tabela de símbolos, agora como parâmetro variável simples, para permitir a existência de comandos de atribuição para o nome da função dentro da mesma e para inibir a utilização deste identificador na seção de parâmetros formais, nas declarações de rótulos, constantes e variáveis e como nome de outros procedimentos ou funções nesta função. Esta segunda ocorrência do nome da função na tabela de símbolos tem como NÍVEL o novo valor de BLOCO e como endereço o endereço de um temporário.

Após análise do tipo da função as duas entradas da tabela de símbolos correspondentes ao nome da função são completadas: a primeira com o tipo da função e a segunda como parâmetro variável simples do mesmo tipo da função.

Após a análise da seção de parâmetros formais , quando for o caso, é completada a tabela de parâmetros com o número de parâmetros reconhecidos.

GERAÇÃO DE CÓDIGO

Esta rotina reserva 9 posições contíguas na área estática inteira (para o endereço de ligação e para as constantes com os tamanhos das áreas de variáveis e temporários da função) e coloca o endereço da primeira posição na tabela de parâmetros. Gera a parte inicial do prólogo da função contendo instruções: para carregar os valores dos registradores base para a função; para atualizar os valores a serem usados como base para o próximo bloco interno (próximo procedimento ou função dentro deste bloco); para salvar o endereço de retorno da função e para salvar o endereço do temporário que receberá o valor da função.

3.2.2.6. RECONHECEDOR DE COMANDOS - PLAPP17

ANÁLISE

A função desta rotina é reconhecer os comandos executáveis do programa fonte. É chamada pelo Reconhedor de Bloco (PLAPP08), por si mesma e pelos reconhecedores específicos dos comandos REPEAT, WHILE, IF, CASE e FOR (PLAPP21, PLAPP22, PLAPP23, PLAPP24 e PLAPP25), com TIPO contendo o código do primeiro símbolo do comando a ser analisado.

Sequência a ser analisada:

<variável simples> := <expressão> |
<variável indexada> { <índices> }$_0^1$:= <expressão> |
<identificador de procedimento> { <seção de parâmetros reais> }$_0^1$ |
BEGIN { <comando> }$_0^1$ { ; { <comando> }$_0^1$ }$_0^n$ END |
GOTO <rótulo> | REPEAT <comando repeat> |
WHILE <comando while> | IF <comando if> |
CASE <comando case> | FOR <comando for> |
READ <comando read> | READP <comando read> |
READD <comando read> | READPD <comando read> |
WRITE <comando write> | WRITELN <comando write> |
WRITEPG <comando write> | WRITED <comando write> |
WRITELND <comando write> | WRITEPGD <comando write>

As expressões são analisadas pelo Reconhecedor de Expressão (PLAPP30); os índices pelo Reconhecedor de Índices (PLAPP29); a seção de parâmetros reais pelo Reconhecedor de Seção de Parâmetros Reais (PLAPP28) e os comandos REPEAT, WHILE, IF, CASE, FOR, READ e WRITE pelos reconhecedores específicos de cada um deles (PLAPP21, PLAPP22, PLAPP23, PLAPP24, PLAPP25, PLAPP26 e PLAPP27, respectivamente).

As construções iniciadas por variáveis são consideradas comandos de atribuição e as iniciadas por identificador de procedimento como ativação de procedimento.

Nos comandos de atribuição a variável indexada pode estar ou não acompanhada de seus índices. No primeiro caso a atribuição é feita ao elemento do array definido pelos índices. No segundo, a atribuição é feita a todos os elementos do array (atribuição matricial).

GERAÇÃO DE CÓDIGO

Para comando de atribuição é utilizado o seguinte algorítmo para geração de código:

1) se a atribuição é para um elemento de array com índices variáveis são geradas pelo Reconhecedor de Índices de Arrays as instruções necessárias para o cálculo do deslocamento do elemento com relação ao início do array.

2) se a expressão (à direita do sinal :=) é variável são geradas pelo Reconhecedor de Expressão as instruções necessárias para o cálculo da expressão.

3) são geradas as instruções:

```
SET     2,(BXUOPND,XUOPND)
SET     3,(BINDEX3,ENDINDEX3)
cod     TIPOVAR,(BUOPND,EUOPND),(BVAR,ENDVAR)        (1)
```

4) se a expressão é matricial, é gerada a instrução:

```
ADREG   2,(0,ENDUM)
```

5) se a atribuição é matricial são geradas as instruções:

```
ADREG   3,(0,ENDUM)
```

```
CREG    3,(0,NELEM)
DESV    0,CCNEQ,ENDDESV
```

onde:

XUOPND - endereço relativo à expressão. Se a expressão se resumir a um elemento de array com índice variável é o endereço do temporário que contém o deslocamento calculado para este elemento. Caso contrário é o endereço da constante inteira zero.

BXUOPND - número do registrador base relativo a XUOPND.

ENDINDEX3 - o mesmo que XUOPND, com relação à variável de destino.

BINDEX - número do registrador base relativo a ENDINDEX3.

cod - ATRIB se a variável e a expressão são do memso tipo. CVT em caso contrário.

TIPOVAR - é o tipo da atribuição ou conversão, dependente dos tipos da variável e da expressão.

EUOPND - endereço do resultado da expressão.

BUOPND - número do registrador base relativo a EUOPND.

ENDVAR - endereço da variável de destino.

BVAR - número do registrador base relativo a ENDVAR.

ENDUM - endereço da constante inteira 1.

NELEM - endereço da constante inteira com o número de elementos do array.

ENDDESV - endereço da instrução (1).

A geração de código para comando de ativação procedimento é feita pelo Reconhecedor de Seção de Parâmetros Reais, mesmo no caso de procedimentos sem parâmetros. Uma descrição geral deste esquema é apresentada no tópico Esquema Geral de Geração de Código - Chamada de Procedimentos e Funções.

Código gerado para comando GOTO:

1) se o rótulo ainda não foi definido (ainda não foi encontrado um comando com este rótulo), são geradas as instruções:

```
SET     3,(0,ENDCTE)
DESV    1,CCQQR,0
```

2) se o rótulo já foi definido, é gerada a instrução:

```
DESV    0,CCQQR,ENDDESV
```

onde:

ENDCTE - endereço da constante inteira com o endereço do rótulo.
ENDDESV - endereço do rótulo.

3.2.2.7. RECONHECEDOR DE TIPO DE VARIÁVEIS - PLAPP18

ANÁLISE

A função desta rotina é analisar tipo das variáveis na declaração de variáveis e na seção de parâmetros formais, preenchendo, no caso de array, a tabela de dimensões de array com as informações referentes ao mesmo.

É chamada pelo Reconhecedor de Declaração de Variáveis (PLAPP10) e pelo Reconhecedor de Seção de Parâmetros Formais (PLAPP20) com TIPO contendo o código do símbolo ":" que segue o nome da variável e retorna (em TIPVAR) o tipo da variável e no caso de array (em PTDIM) o índice da tabela de dimensões de array (TABDIM) onde foram armazenadas as dimensões do array.

Este reconhecedor efetua a análise até o tipo da variável. Se o tipo for válido, então retorna posicionado no próprio símbolo (que deverá ser pulado pela rotina que o chamou). Caso contrário retorna posicionado no símbolo que ocupa o lugar do tipo (que deverá ser analisado pela rotina que o chamou).

Sequência a ser analisada:

ARRAY [<dim> { , <dim> }$_{0}^{n}$] OF <tipo> | <tipo>

onde:

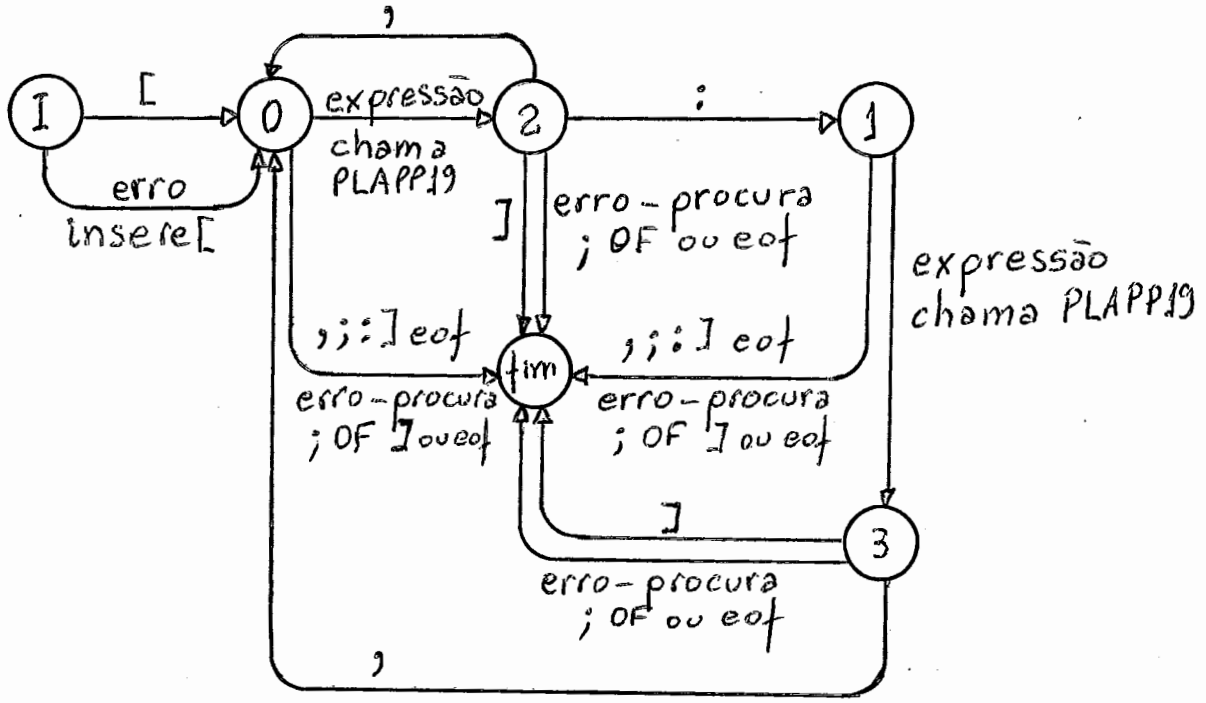
<tipo> - uma das palavras reservadas INTEGER, REAL, BOOLEAN ou

CHAR.

<dim> ::= <expressão tipo 1> |
<expressão tipo 1> : <expressão tipo 1>

A expressão tipo 1 é analisada pelo Reconhecedor de Expressão Tipo 1 (PLAPP19).

A análise das dimensões do array é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

Esta rotina não gera instruções.

3.2.2.8. RECONHECEDOR DE EXPRESSÃO TIPO 1 - PLAPP19

ANÁLISE

Esta função analisa expressão utilizada nas declarações de variáveis para dimensionamento de arrays.

É chamada pelo Reconhecedor de Tipo de Variáveis (PLAPP18) com o primeiro elemento da expressão lido (mesmo que inválido) e retorna (em VALOR) o valor calculado da expressão e '0'B se a expressão estiver correta ou '1'B em caso de erro.

TIPO retorna sempre com o código correspondente a um dos símbolos: ",", ";", ":" ou "]".

Sequência a ser analisada:

```
<termo 1>  |  <sinal>  <termo 1>  |
<expressão tipo 1>  +  <termo 1>  |
<expressão tipo 1>  -  <termo 1>
```

onde:

```
<sinal>  ::=  +  |  -
<termo 1>  ::=  <fator 1>  |  <termo 1>  *  <fator 1>  |
                <termo 1>  /  <fator 1>  |
                <termo 1>  DIV  <fator 1>  |
                <termo 1>  MOD  <fator 1>
<fator 1>  ::=  <número sem sinal> | ( <expressão tipo 1> ) |
                <identificador de constante>
```

É empregado o método citado anteriormente para a análise da expressão. Como esta se compõe somente de constantes, seu valor é calculado à medida que a análise sintática é feita.

O método consiste em se atribuir uma prioridade estática a cada operador e utilizar-se uma variável base, com valor inicial zero, que é incrementada de um valor maior que a maior prioridade estática (no caso, 5) cada vez que é encontrado um "(" na expressão e é decrementada deste mesmo valor cada vez que é encontrado um ")". Esta variável deve ser sempre, durante a análise, maior ou igual a zero, e no final, igual a zero. Caso contrário há um desbalanceamento de parênteses na expressão.

É utilizada uma pilha onde são colocados os operadores, suas prioridades dinâmicas e os operandos.

Prioridade dinâmica de um operador é a sua prioridade estática somada ao valor da base no momento da inserção do operador na pilha.

Os operandos são sempre inseridos na pilha, independente de qualquer condição.

Antes da inserção de cada operador na pilha, compara-se a prioridade dinâmica no topo da pilha com a do candidato à inserção. Se a da pilha for menor que a do operador, é feita a inserção. Se for maior ou igual, a expressão é reduzida, isto é, são efetuadas as operações, até ser encontrada na pilha uma prioridade menor que a do candidato à inserção, quando é então feita a inserção.

Operador	Prioridade estática
+ -	1
* / MOD DIV	2

No início da análise é inserido na pilha um operador simbólico indicando início de expressão, com prioridade estática igual a zero. Os símbolos ",", ":" ou "]" indicam fim da expressão e têm prioridade dinâmica zero, independente do valor corrente da base.

O sinal "+" unário é ignorado e o "-" unário provoca a inserção na pilha do operando zero seguido do operador "-" com sua prioridade dinâmica.

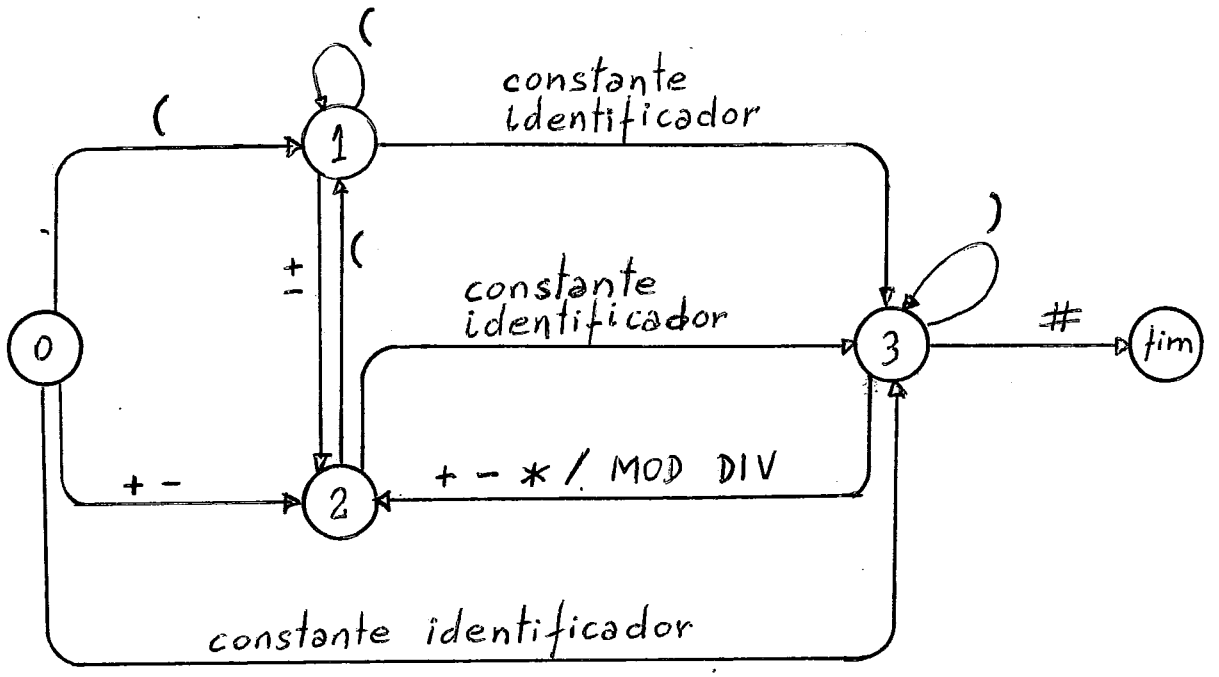
Se o resultado final da expressão for um número real, este valor será truncado.

Em caso de erro na expressão, é impressa uma mensagem de erro e os elementos seguintes são ignorados até ser encontrado um dos símbolos: ";", ":" ou "]", ou fim de arquivo. Neste caso a função retorna igual a '1'B e com VALOR igual a 1 para evitar futuras mensagens de erros desnecessárias.

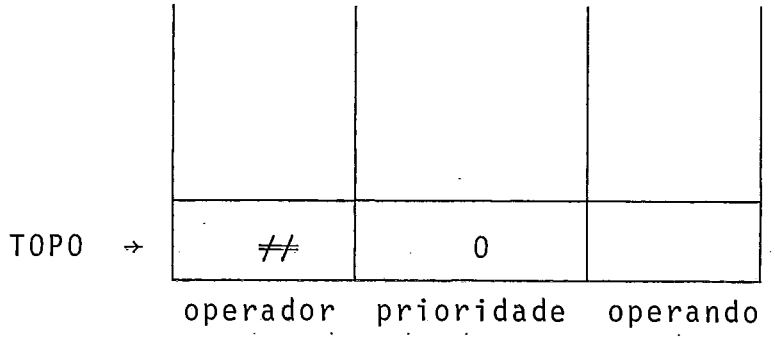
É considerado erro na expressão:

- desbalanceamento de parênteses;
- identificador não numérico ou não definido;
- divisão por zero;
- dois operadores seguidos;
- dois operandos seguidos e
- qualquer outra situação não prevista no diagrama a seguir:

A análise sintática é feita de acordo com o seguinte diagrama de estados:



Situação da pilha no início da análise:



Observação:

- representa início e fim da expressão.
TOPO - aponta sempre para o último operador (e prioridade) inserido e próximo operando a ser inserido.

GERAÇÃO DE CÓDIGO

Esta rotina não gera instruções.

3.2.2.9. RECONHECEDOR DE SEÇÃO DE PARÂMETROS FORMAIS - PLAPP20

ANÁLISE:

A função desta rotina é analisar seção de parâmetros formais de procedimentos e funções, preenchendo a tabela de parâmetros(TABPARM) com as informações referentes aos parâmetros reconhecidos.

É chamada pelo Reconhecedor de Cabeçalho de procedimento(PLAPP11) e pelo Reconhecedor de Cabeçalho de Função (PLAPP12), com TIPO contendo o código correspondente ao símbolo "(" e retorna com valor '0'B se não reconheceu nenhum parâmetro ou '1'B caso contrário. Além disto retornará ao ser detetado o símbolo ")", que encerra a seção de parâmetros, e com TIPO contendo o código correspondente ao primeiro símbolo encontrado após o ")".

Esta rotina recebe dois parâmetros: o primeiro (ENDLISTA) com o endereço de ligação do procedimento ou função cujos parâmetros formais estão sendo analisados e o segundo (DESLOC) indicando quem chamou esta rotina(0 se Reconhecedor de Cabeçalho de procedimento e 1 de Reconhecedor de Cabeçalho de Função).

Sequência a ser analisada:

(<definição de parâmetros formais> { ; <definição de parâmetros formais> }$_0^n$)

onde:

<definição de parâmetros formais> ::= <grupo de parâmetros> |
VAR <grupo de parâmetros> |
FUNCTION <identificador> {, <identificador> }$_0^n$: <tipo>|
PROC <identificador> {, <identificador> }$_0^n$
<grupo de parâmetros> ::= <identificador> { , <identificador>}$_0^n$
: <tipo da variável>

O tipo da variável é analisado pelo Reconhecedor de Tipo de Variável (PLAPP18).

GERAÇÃO DE CÓDIGO:

Para cada parâmetro formal reconhecido é gerada uma instrução para copiar seu endereço para um temporário interno ao procedimento ou função.

No caso de parâmetro procedimento ou função o endereço do parâmetro corresponde ao endereço da primeira instrução do procedimento ou função passada como parâmetro e são geradas também instruções para copiar seu endereço de ligação e nível.

No caso de parâmetro por valor é gerada também uma instrução para copiar para um temporário do procedimento ou função o valor corrente do parâmetro.

3.2.2.10. RECONHECEDOR DE COMANDO "REPEAT" - PLAPP21

ANÁLISE:

A função desta rotina é analisar comando "REPEAT".

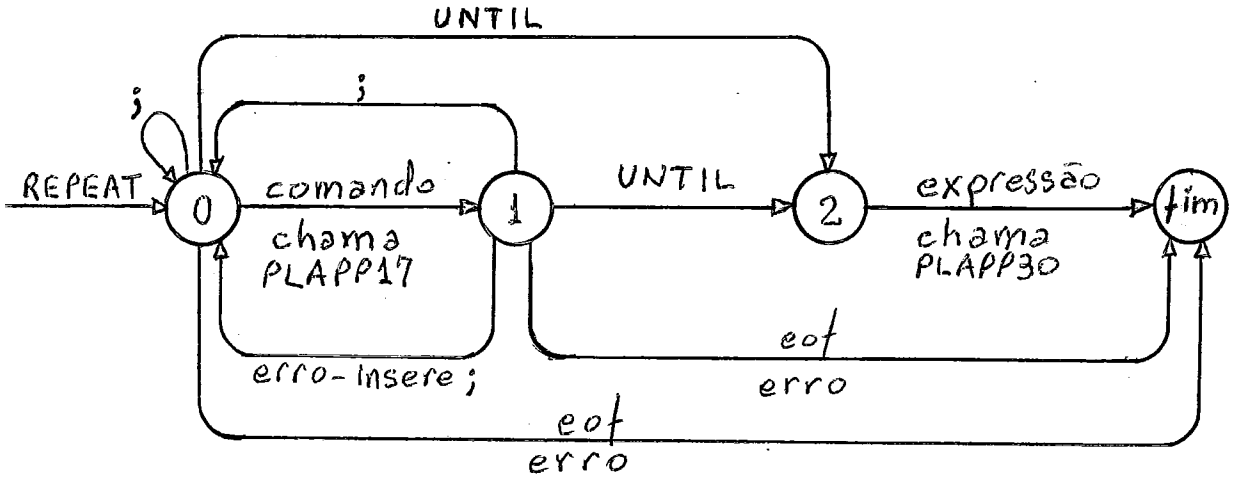
É chamada pelo Reconhecedor de Comandos (PLAPP17) com TIPO contendo o código da palavra reservada REPEAT e retorna com o código do símbolo após o comando.

Sequência a ser analisada:

REPEAT { <comando> }$_0^1$ {; { <comando> }$_0^1$ }$_0^n$ UNTIL <expressão>

Os comandos são analisados pelo Reconhecedor de Comandos (PLAPP17) e a expressão pelo Reconhecedor de Expressão (PLAPP30).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

É utilizado o seguinte algorítmo para geração de código para o comando "REPEAT":

1) são geradas pelo Reconhecedor de Comandos as instruções relativas aos comandos por êle analisados.

2) se a expressão for constante e igual a FALSE é gerada a instrução:

```
DESV        0,CCQQR,ENDTEST
```

3) se a expressão for constante e igual a TRUE é encerrada a geração de código.

4) se a expressão for variável são geradas pelo Reconhecedor de Expressão as instruções relativas à expressão e em seguida as instruções:

```
SET         2,(BXUOPND,XUOPND)
SET         3,(0,ENDZERO)
COMP        3,(BUOPND,EUOPND),(0,EFALSE)
DESV        0,CCEQ,ENDTEST
```

onde:

ENDTEST - endereço da primeira instrução correspondente ao primeiro comando do comando "REPEAT".

XUOPND - endereço relativo à expressão. Se a expressão se resumir a um elemento de array com índice variável é o endereço do temporário inteiro que contém o deslocamento calculado para este elemento. Caso contrário é o endereço da constante inteira zero.

BXUOPND - número do registrador base relativo a XUOPND.

ENDZERO - endereço da constante inteira zero.

EUOPND - endereço do resultado da expressão.

BUOPND - número do registrador base relativo a EUOPND.

EFALSE - endereço da constante booleana FALSE.

3.2.2.11. RECONHECEDOR DE COMANDO "WHILE" - PLAPP22

ANÁLISE

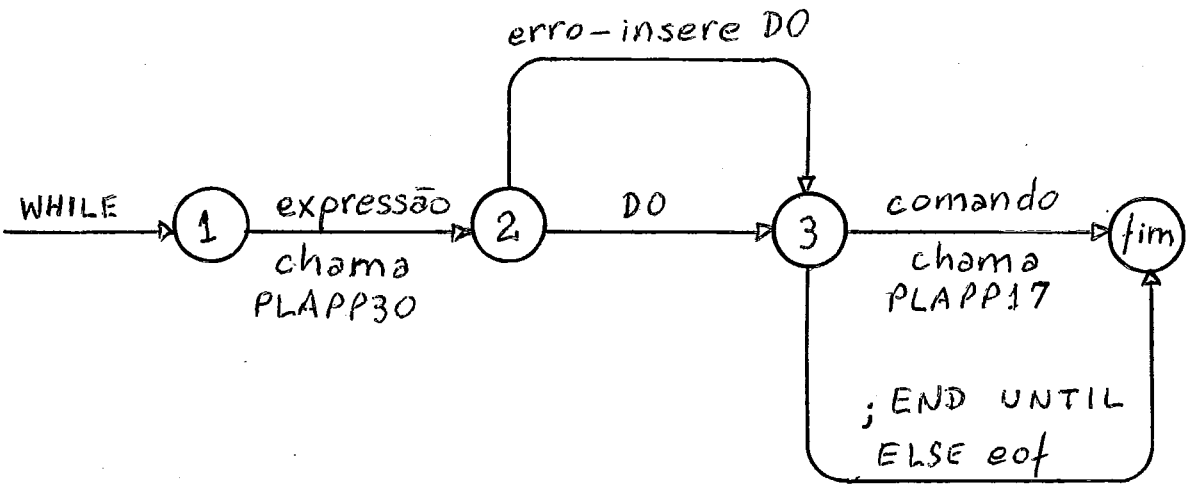
A função desta rotina é analisar comando "WHILE". É chamada pelo Reconhecedor de Comandos (PLAPP17) com TIPO contendo o código da palavra reservada WHILE e retorna com o código do símbolo após o comando.

Sequência a ser analisada:

WHILE <expressão> DO { <comando> }$_{0}^{1}$

A expressão é analisada pelo Reconhecedor de Expressão (PLAPP30) e o comando pelo Reconhecedor de Comandos (PLAPP17).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

É utilizado o seguinte algorítmo para geração de código para o comando "WHILE":

1) se a expressão for variável são geradas pelo Reconhecedor de Expressão as instruções relativas à expressão e em seguida as instruções:

```
SET     2,(BXUOPND,XUOPND)                    (1)
SET     3,(0,ENDZERO)
COMP    3,(BUOPND,EUOPND),(0,ETRUE)
```

```
SET     3,(0,ENDDESV)
DESV    1,CCNEQ,0
```

2) se a expressão for constante e igual a FALSE são geradas as instruções:

```
SET     3,(0,ENDDESV)                                    (2)
DESV    1,CCQQR,0
```

3) são geradas pelo Reconhecedor de Comandos as instruções relativas ao comando por êle analisado.

4) é gerada a instrução:

```
DESV    0,CCQQR,ENDTEST
```

onde:

XUOPND - endereço relativo à expressão. Se a expressão se resumir a um elemento de array com índice variável é o endereço do temporário inteiro que contém o deslocamento calculado para este elemento. Caso contrário é o endereço da constante inteira zero.

BXUOPND - número do registrador base relativo a XUOPND.

ENDZERO - endereço da constante inteira zero.

EUOPND - endereço do resultado da expressão.

BUOPND - número do registrador base relativo a EUOPND.

ETRUE - endereço da constante booleana TRUE.

ENDDESV - endereço da constante inteira com o endereço da primeira instrução após o comando "WHILE".

ENDTEST - endereço da instrução (1) se a expressão for variável; da instrução (2) se a expressão for constante e igual a FALSE ou o endereço da primeira instrução gerada pelo Reconhecedor de Comandos (ítem 3 do algorítmo).

3.2.2.12. RECONHECEDOR DE COMANDO "IF" - PLAPP23

ANÁLISE

A função desta rotina é analisar comando "IF".

É chamada pelo Reconhecedor de Comandos (PLAPP17) com TIPO contendo o código da palavra reservada "IF" e retorna

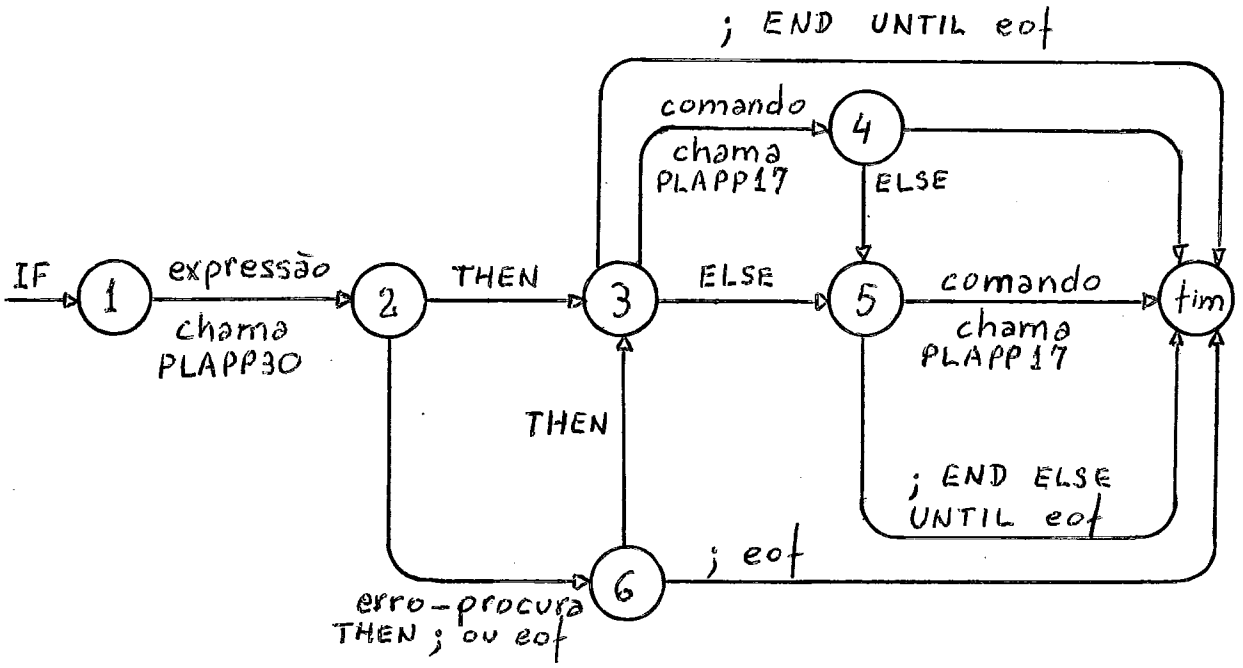
com o código do símbolo após o comando.

Sequência a ser analisada:

IF <expressão> THEN { <comando> }$_0^1$ { ELSE { <comando> }$_0^1$ }$_0^1$

A expressão é analisada pelo Reconhecedor de Expressão (PLAPP30) e os comandos pelo Reconhecedor de Comandos (PLAPP17).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO:

É utilizado o seguinte algorítmo para geração de código para o comando "IF":

1) se a expressão for variável, são geradas pelo Reconhecedor de Expressão as instruções relativas à expressão e, em seguida, as instruções:

```
 SET  2,(BXUOPND,XUOPND)
 SET  3,(0,ENDZERO)
COMP  3,(BUOPND,EUOPND),(0,CBOOL)
 SET  3,(0,END1)
DESV  1,CCEQ,0
```

2) se a expressão for constante e igual a TRUE e não existir o comando após a palavra reservada THEN, ou se a expressão for

constante e igual a FALSE porém existindo o comando após a a palavra reservada THEN, são geradas as instruções:

```
SET  3,(0,END1)
DESV 1,CCQQR,0
```

3) se existe o comando após a palavra reservada THEN, são geradas pelo Reconhecedor de Comandos as instruções relativas ao comando e em seguida as instruções:

```
SET  3,(0,END2)
DESV 1,CCQQR,0
```

4) se existe o comando após a palavra reservada ELSE, são geradas pelo Reconhecedor de Comandos as instruções relativas ao comando.

onde:

XUOPND - endereço relativo à expressão. Se a expressão se resumir a um elemento de array com índice variável é o endereço do temporário que contém o deslocamento calculado para este elemento. Caso contrário é o endereço da constante inteira zero.

BXUOPND - número do registrador base relativo a XUOPND.

ENDZERO - endereço da constante inteira zero.

EUOPND - endereço do resultado do expressão.

BUOPND - número do registrador base relativo a EUOPND.

CBOOL - endereço da constante booleana FALSE se existe o comando após a palavra reservada THEN ou o endereço da constante booleana TRUE em caso contrário.

END1 - endereço da constante que contém o endereço da primeira instrução gerada após as instruções correspondentes ao comando da cláusula THEN.

END2 - endereço da constante que contém o endereço da primeira instrução gerada após as instruções correspondentes ao comando da cláusula ELSE.

3.2.2.13. RECONHECEDOR DE COMANDO "CASE" - PLAPP24

ANÁLISE:

A função desta rotina é analisar comando "CASE".

É chamada pelo Reconhecedor de Comandos (PLAPP17), com TIPO contendo o código da palavra reservada CASE e retorna com o código do símbolo após o comando.

Sequência a ser analisada:

CASE <expressão> OF {<elemento de case> {; <elemento de case> }$_0^n$

{ ; }$_0^1$ }$_0^1$ END

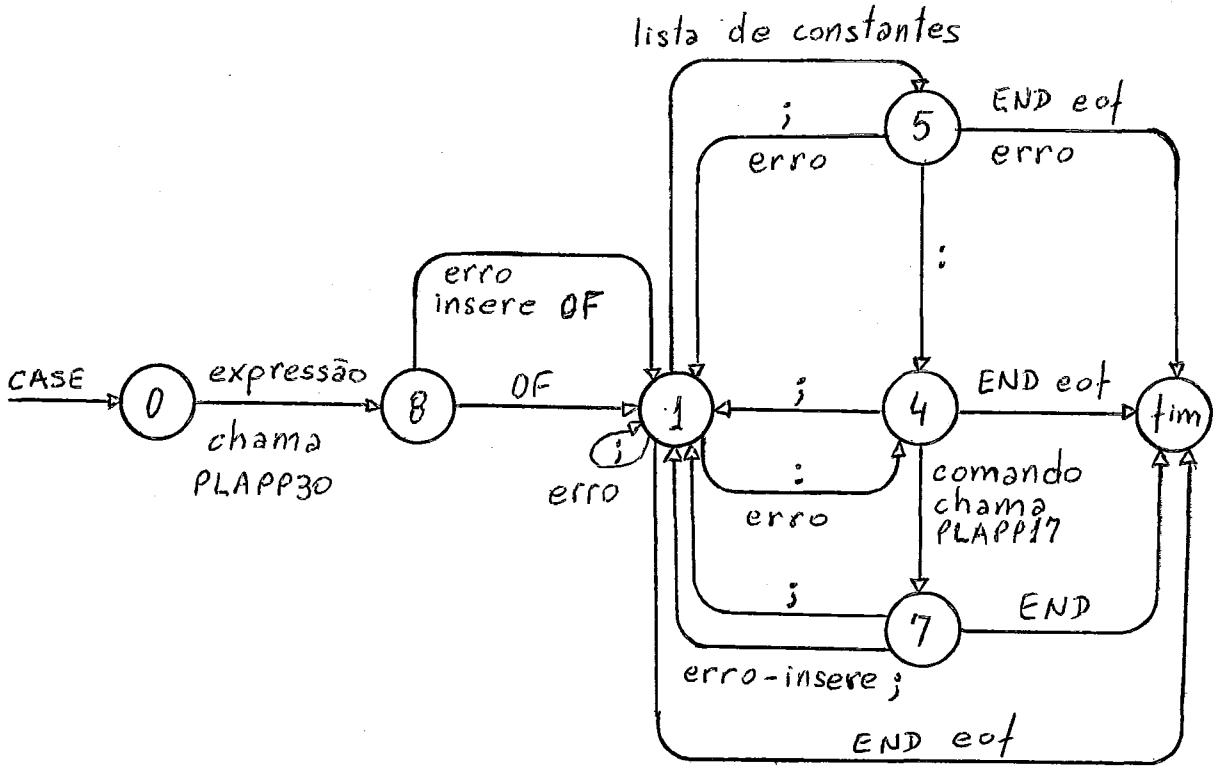
onde:

<elemento de case> ::= <constante> {, <constante> }$_0^n$:

{ <comando> }$_0^1$

A expressão é analisada pelo Reconhecedor de Expressão (PLAPP30) e os comandos pelo Reconhecedor de Comandos (PLAPP17).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

Para execução do comando "CASE" são geradas durante a compilação duas tabelas na área estática: uma para rótulos e outra para endereços dos comandos analisados.

O número de posições da tabela de rótulos é igual ao número de rótulos do comando "CASE" mais um. Em cada posição é colocado o valor de um rótulo. Na posição excedente, última posição da tabela, é colocado durante a execução o valor corrente da expressão seletora.

A tabela de endereços tem o mesmo número de posições que a tabela de rótulos. Na última posição da tabela de endereços e nas posições correspondentes a rótulos sem comandos, isto é, rótulos com comando vazio, é colocado zero. Nas demais é colocado o valor da diferença entre o endereço da primeira instrução do comando e o endereço da primeira instrução gerada após o comando "CASE".

Em fase de execução do comando "CASE" é feita uma busca sequencial na tabela de rótulos, tendo como argumento o valor corrente da expressão seletora. A inclusão deste valor no final da tabela garante que a busca será sempre bem sucedida.

Ao final da busca é feito um desvio indexado para o endereço contido na posição correspondente da tabela de endereços, tendo como endereço básico para desvio o endereço da primeira instrução gerada após o comando "CASE".

É utilizado o seguinte algorítmo para geração de código para o comando "CASE":

1) são geradas pelo Reconhecedor de Expressão as instruções relativas à expressão.

2) são geradas as instruções:

```
SET     3,(0,REFINI)
DESV    1,CCQQR,0
```

3) para cada comando do "CASE" são geradas pelo Reconhecedor de Comandos as instruções correspondentes ao comando. Em seguida esta rotina gera, para cada comando existente, as instruções:

```
SET     3,(0,REFEND)
```

```
   DESV      1,CCQQR,0
```

4) São geradas as instruções:

```
   SET       2,(BXUOPND,XUOPND)                              (1)
   SET       3,(0,ENDZERO)
   ATRIB     TIPOSEL,(BUOPND,EUPOND),(0,E1)
   SET       2,(0,ENDZERO)
   SET       3,(0,ENDZERO)
   COMP      TIPOSEL,(0,E2),(0,E1)                           (2)
   DESV      0,CCEQ,E3
   ADREG     2,(0,ENDUM)
   DESV      0,CCQQR,E4
   SET       3,(0,ENDZERO)                                   (3)
   ATRIB     1,(0,E5),(BTEMP,TEMP)
   SET       3,(BTEMP,TEMP)
   DESV      1,CCQQR,E6
```

onde:

REFINI - endereço da constante inteira com o endereço da instrução (1).

REFEND - endereço da constante inteira com o endereço da primeira instrução gerada após o comando "CASE".

XUOPND - endereço relativo à expressão seletora. Se a expressão se resumir a um elemento de array com índice variável, é o endereço do temporário inteiro que contém o deslocamento calculado para este elemento. Caso contrário é o endereço da constante inteira zero.

BXUOPND - número do registrador base relativo a XUOPND.

ENDZERO - endereço da constante inteira zero.

TIPOSEL - tipo da expressão seletora.

EUOPND - endereço do resultado da expressão seletora.

BUOPND - número do registrador base relativo a EUOPND.

E1 - endereço da última posição da tabela de rótulos.

E2 - endereço da primeira posição da tabela de rótulos.

E3 - endereço da instrução (3).

ENDUM - endereço da constante inteira um.

E4 - endereço da instrução (2).

E5 - endereço da primeira posição da tabela de endereços.

TEMP - endereço de um temporário inteiro do bloco.
BTEMP - número do registrador base relativo a TEMP.
E6 - endereço da primeira instrução gerada após o comando "CASE".

3.2.2.14. RECONHECEDOR DE COMANDO "FOR" - PLAPP25

ANÁLISE

A função desta rotina é analisar comando "FOR".

É chamada pelo Reconhecedor de Comandos (PLAPP17), com TIPO contendo o código da palavra reservada FOR e retorna com o código do símbolo após o comando.

Sequência a ser analisada:

FOR <identificador de variável> := <expressão 1> <incremento>

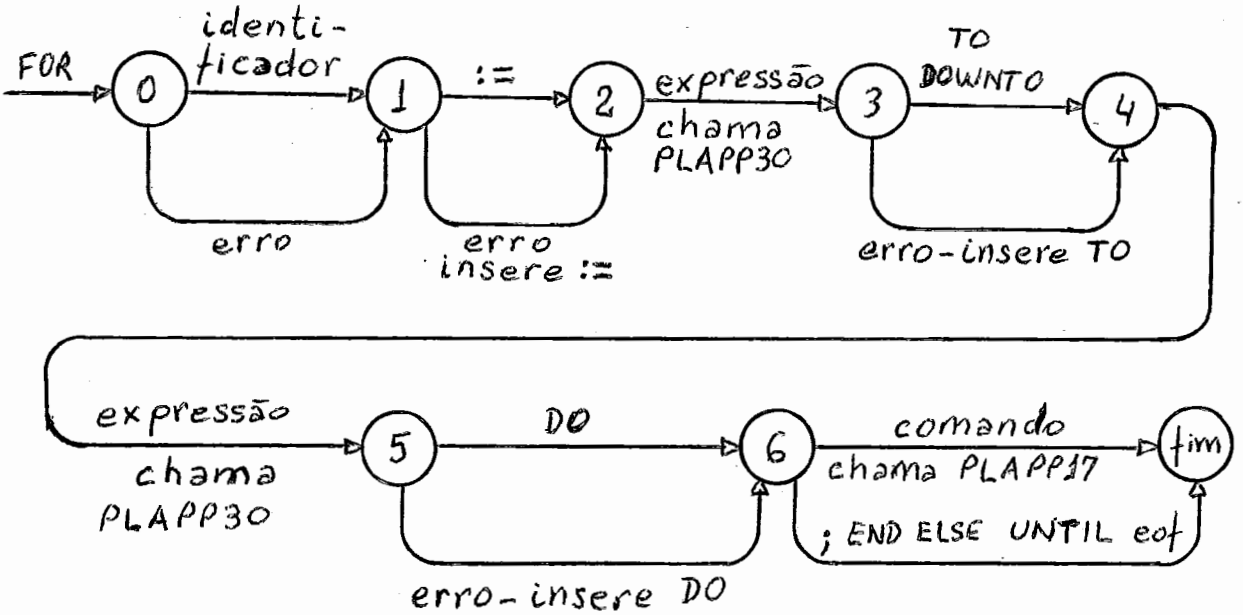
<expressão 2> DO { <comando> }$_{0}^{1}$

onde:

<incremento> ::= TO | DOWNTO

As expressões são analisadas pelo Reconhecedor de Expressão (PLAPP30) e o comando pelo Reconhecedor de Comandos (PLAPP17).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

É utilizado o seguinte algorítmo para geração de código para comando "FOR":

1) São geradas pelo Reconhecedor de Expressão as instruções relativas à expressão 1.

2) Se a expressão 2 for variável, são geradas pelo Reconhecedor de Expressão as instruções relativas à expressão e em seguida as instruções:

```
SET     2,(BXUOPND2,XUOPND2)
SET     3,(0,ENDZERO)
ATRIB   1,(BUOPND2,EUOPND2),(BTEMP,TEMP)
```

3) São geradas as instruções:

```
SET     2,(BXUOPND1,XUOPND1)
SET     3,(0,ENDZERO)
ATRIB   1,(BUOPND1,EUOPND1),(BVAR,ENDVAR)
ZEREG   0,1,1                                          (1)
COMP    1,(BVAR,ENDVAR),(BUOPND2,EUOPND2)
SET     3,(0,ENDDESV)
DESV    1,CODDESV,0
```

4) São geradas pelo Reconhecedor de Comandos as instruções relativas ao comando por êle analisado.

5) São geradas as instruções:

```
ZEREG   1,1,1
ADI     (0,ENDINC),(BVAR,ENDVAR),(BVAR,ENDVAR)
DESV    0,CCQQR,ENDTEST
```

onde:

XUOPND2 - endereço relativo à expressão 2. Se a expressão 2 se resumir a um elemento de array com índice variável, é o endereço do temporário inteiro que contém o deslocamento calculado para este elemento. Caso contrário é o endereço da constante inteira zero.

BXUOPND2 - número do registrador base relativo a XUOPND2.

ENDZERO - endereço da constante inteira zero.

EUOPND2 - endereço do resultado da expressão 2.

BUOPND2 - número do registrador base relativo a EUOPND2.

TEMP - endereço de um temporário inteiro do bloco.
BTEMP - número do registrador base relativo a TEMP.
XUOPND1 - o mesmo que XUOPND2, com relação à expressão 1.
BXUOPND1 - número do registrador base relativo a XUOPND1.
EUOPND1 - endereço do resultado da expressão 1.
BUOPND1 - número do registrador base relativo a EUOPND1.
ENDVAR - endereço do identificador (variável de controle do comando "FOR").
BVAR - número do registrador base relativo a ENDVAR.
ENDDESV - endereço da constante inteira com o endereço da primeira instrução gerada após o comando "FOR".
CODDESV - se <incremento> = TO, CODDESV = CCGT;
se <incremento> = DOWNTO, CODDESV = CCLT.
ENDINC - se <incremento> = TO é o endereço da constante inteira 1. Se <incremento> = DOWNTO é o endereço da constante inteira -1.
ENDTEST - endereço da instrução (1).

3.2.2.15. RECONHECEDOR DE COMANDO "READ" - PLAPP26

ANÁLISE

A função desta rotina é analisar comando "READ".

É chamada pelo Reconhecedor de Comandos (PLAPP17) com TIPO contendo o código de uma das palavras reservadas - READ, READD, READP ou READPD - e retorna com o código do símbolo após o comando.

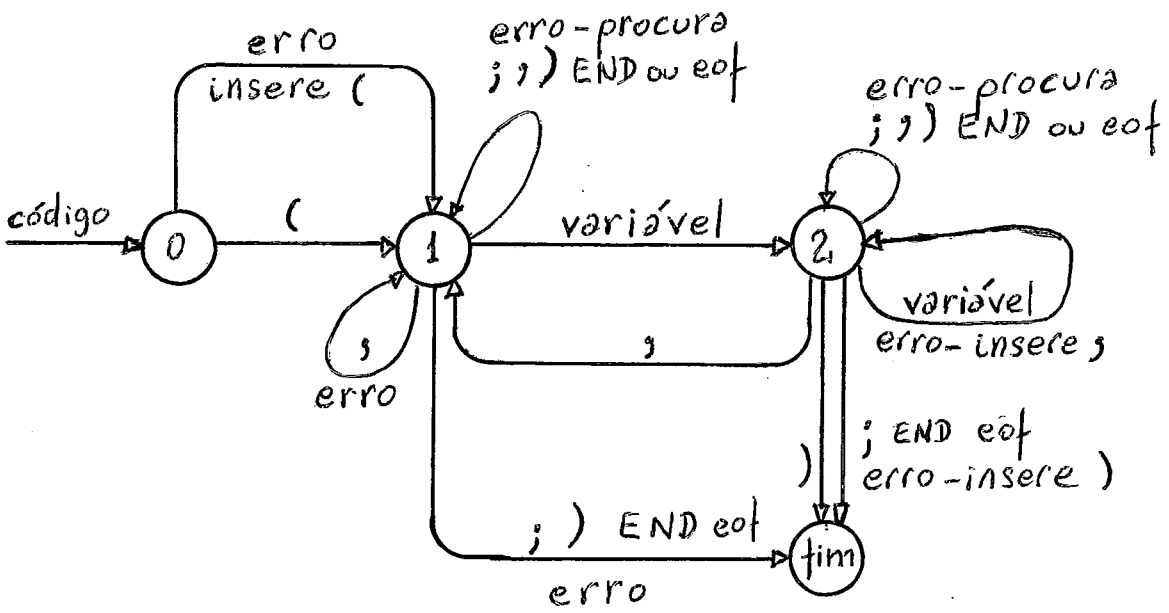
Sequência a ser analisada:

<código> (<operando> { , <operando> }$_{0}^{n}$)

onde:
<código> ::= READ | READD | READP | READPD
<operando> ::= <identificador de variável simples> |
<identificador de variável indexada>
{ <índices> }$_{0}^{1}$

Os índices são analisados pelo Reconhecedor de Índices de Arrays (PLAPP29).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

Para execução do comando "READ" é gerada durante a compilação uma lista, na área estática inteira, que descreve os operandos a serem lidos.

Existem nesta lista tres entradas para cada operando (no caso de operando matricial, cada elemento da matriz é considerado um operando): o tipo do endereço do operando; o endereço do operando e o número do registrador base relativo ao endereço do operando.

O tipo do endereço indica se o operando é inteiro, real, booleano ou caráter, com endereço direto ou indireto, conforme tabela abaixo:

Tipo do Endereço	Operando
3	inteiro com endereço direto
4	inteiro com endereço indireto
7	real com endereço direto
8	real com endereço indireto
11	booleano com endereço direto
12	booleano com endereço indireto

Tipo do Endereço	Operando
14	caráter com endereço direto
15	caráter com endereço indireto

Para os operandos com tipo de endereço direto, o endereço do operando é o próprio deslocamento do operando. Nos demais casos o endereço do operando é o deslocamento do temporário inteiro que contém o endereço absoluto do operando.

O tipo de endereço é direto se o operando é uma variável simples, um parâmetro variável simples por valor, um elemento de array com índices constantes, um elemento de parâmetro array por valor com índices constantes, um array ou um parâmetro array por valor.

O tipo de endereço é indireto se o operando é um parâmetro variável simples por endereço, um elemento de array com índice variável, um elemento de parâmetro array por valor com índice variável, um elemento de parâmetro array por endereço ou um parâmetro array por endereço.

É utilizado o seguinte algorítmo para geração de código para comando "READ":

1) Para cada operando:

1.1) Se o operando for um elemento de array, são geradas pelo Reconhecedor de Índices de Array as instruções para calcular o deslocamento do elemento. Nos casos de elemento de array com índice variável e elemento de parâmetro array por valor com índice variável são geradas também as seguintes instruções para cálculo do endereço absoluto do elemento:

```
STOB    T,BVAR,(BTEMP,TEMP)
ZEREG   1,1,1
ADI     (0,END1),(BTEMP,TEMP),(BTEMP,TEMP)
ADI     (BTEMP,TEMP),(BIND,ENDIND),(BTEMP,TEMP)
```

1.2) Se o operando for parâmetro array por endereço são geradas as seguintes instruções para calcular o endereço absoluto de cada elemento do array:

```
ZEREG       1,1,1
ATRIB       1,(BVAR,ENDVAR),(BTEMP,TEMP)
SET         3,(0,ENDUM)
ADI         (BTEMP,TEMP),(0,ENDUM),(BTEMP,TEMP)          (1)
ADREG       1,(0,ENDUM)
ADREG       3,(0,ENDUM)
CREG        3,(0,NELEM)
DESV        0,CCNEQ,END2
```

2) É gerada a instrução:

```
cod         A,N,END3
```

onde:

T - número da coluna da matriz de registradores base correspondente ao tipo do array (1 - inteiro, 2 - real, 3 - booleano, 4 - caráter) ou do tipo do parâmetro array por valor (5 - inteiro, 6 - real, 7 - booleano, 8 - caráter).

BVAR - número do registrador base correspondente ao operando.

TEMP - endereço de um temporário inteiro do bloco.

BTEMP - número do registrador base relativo a TEMP.

END1 - endereço da constante inteira com o endereço do array.

ENDIND - endereço do temporário inteiro com o deslocamento calculado para o elemento.

BIND - número do registrador base relativo a ENDIND.

ENDVAR - endereço do operando. Neste caso (parâmetro por endereço) é o endereço do temporário inteiro com o endereço absoluto do operando.

ENDUM - endereço da constante inteira um.

NELEM - endereço da constante inteira com o número de elementos do array.

END2 - endereço da instrução (1).

cod - READ, READD, READP ou READPD, conforme o código do comando.

A - código do arquivo. No caso presente 0, para indicar arquivo de cartões.

END3 - endereço da lista que descreve os operandos a serem lidos.

N - número de operandos descritos nesta lista.

3.2.2.16. RECONHECEDOR DE COMANDO "WRITE" - PLAPP27

ANÁLISE

A função desta rotina é analisar comando "WRITE". É chamada pelo Reconhecedor de Comandos (PLAPP17) com TIPO contendo o código de uma das palavras reservadas - WRITE, WRITED, WRITELN, WRITELND, WRITEPG ou WRITEPGD - e retorna com o código do símbolo após o comando.

Sequência a ser analisada:

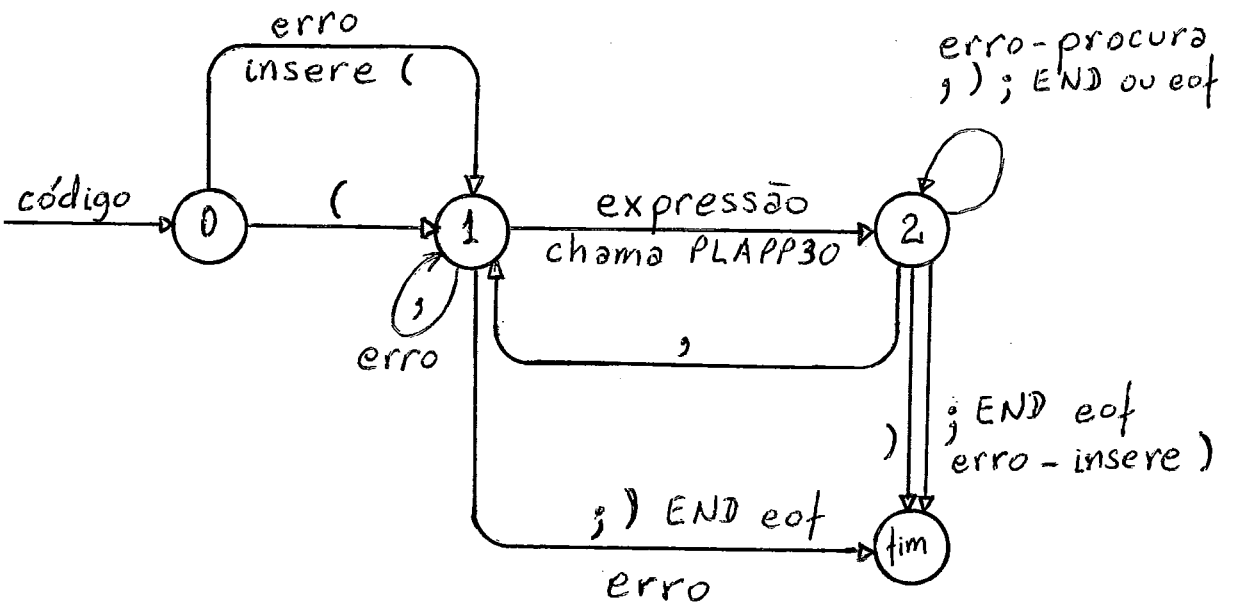
<código> (<expressão> { , <expressão> }$_{0}^{n}$)

onde:

<código> ::= WRITE | WRITED | WRITELN | WRITELND | WRITEPG | WRITEPGD

As expressões são analisadas pelo Reconhecedor de Expressão (PLAPP30).

A análise é feita de acordo com o seguinte diagrama de estados:



GERAÇÃO DE CÓDIGO

Para execução do comando "WRITE" é gerada durante a compilação uma lista, na área estática inteira, que descreve os operandos a serem impressos.

Existem nesta lista quatro entradas para cada operando (no caso de operando matricial, cada elemento da matriz é considerado um operando): o tipo do endereço do operando; o tamanho do operando; o endereço do operando e o número do registrador base relativo ao endereço do operando.

O tipo do endereço indica se o operando é inteiro, real, booleano ou caráter, se é constante, temporário ou variável e se o endereço é direto ou indireto, conforme tabela abaixo:

Tipo do Endereço	Operando
1	inteiro, temporário, com endereço direto
2	inteiro, constante, com endereço direto
3	inteiro, variável, com endereço direto
4	inteiro, variável, com endereço indireto
5	real, temporário, com endereço direto
6	real, constante, com endereço direto
7	real, variável, com endereço direto
8	real, variável, com endereço indireto
9	booleano, temporário, com endereço direto
10	booleano, constante, com endereço direto
11	booleano, variável, com endereço direto
12	booleano, variável, com endereço indireto
13	caráter, constante, com endereço direto
14	caráter, variável, com endereço direto
15	caráter, variável, com endereço indireto

Para os operandos com tipo do endereço direto, o endereço do operando é o próprio deslocamento do operando. Nos demais casos o endereço do operando é o deslocamento do temporário inteiro que contém o endereço absoluto do operando.

O tipo do endereço de um operando é indireto se a expressão correspondente a esse operando se resumir a um parâmetro variável simples por endereço, um elemento de array com índice variável, um elemento de parâmetro array por valor com índice variável, um elemento de parâmetro array por endereço ou um parâmetro array por endereço, sendo direto nos demais casos.

É utilizado o seguinte algorítmo para geração de código para o comando "WRITE":

1) Para cada operando:

1.1) São geradas pelo Reconhecedor de Expressão as instruções relativas à expressão por êle analisada.

1.2) Se o operando é um parâmetro array por endereço, são geradas as seguintes instruções para calcular o endereço absoluto de cada elemento do array:

```
ZEREG     1,1,1
ATRIB     1,(BVAR,ENDVAR),(BTEMP,TEMP)
SET       3,(0,ENDUM)
ADI       (BTEMP,TEMP),(0,ENDUM),(BTEMP,TEMP)          (1)
ADREG     1,(0,ENDUM)
ADREG     3,(0,ENDUM)
CREG      3,(0,NELEM)
DESV      0,CCNEQ,END1
```

1.3) Se o operando é um elemento de array com índice variável ou um elemento de parâmetro array por valor com índice variável, são geradas as seguintes instruções para calcular o endereço absoluto do elemento:

```
STOB      T,BVAR,(BTEMP,TEMP)
ZEREG     1,1,1
ADI       (0,END2),(BTEMP,TEMP),(BTEMP,TEMP)
ADI       (BTEMP,TEMP),(BIND,ENDIND),(BTEMP,TEMP)
```

2) É gerada a instrução:

```
cod       A,N,END3
```

onde:

ENDVAR - endereço do operando. Neste caso (parâmetro por endereço) é o endereço do temporário inteiro com o endereço absoluto do operando.

BVAR - número do registrador base correspondente ao operando.

TEMP - endereço de um temporário inteiro do bloco.

BTEMP - número do registrador base relativo a TEMP.

ENDUM - endereço da constante inteira um.

NELEM - endereço da constante inteira com o número de elementos do array.

END1 - endereço da instrução (1).
T - número da coluna da matriz de registradores base correspondente ao tipo do array (1 - inteiro, 2 - real, 3 - booleano, 4 - caráter) ou ao tipo do parâmetro array por valor (5 - inteiro, 6 - real, 7 - booleano, 8 - caráter).
END2 - endereço da constante inteira com o endereço do array.
ENDIND - endereço do temporário inteiro com o deslocamento calculado para o elemento.
BIND - número do registrador base relativo a ENDIND.
cod - WRITE, WRITED, WRITELN, WRITELND, WRITEPG ou WRITEPGD, conforme o código do comando.
A - código do arquivo. No caso presente 0, para indicar arquivo de impressão.
END3 - endereço da lista que descreve os operandos a serem impressos.
N - número de operandos descritos nesta lista.

3.2.2.17. RECONHECEDOR DE SEÇÃO DE PARÂMETROS REAIS - PLAPP28

ANÁLISE

A função desta rotina é analisar seção de parâmetros reais de procedimentos e funções, verificando a compatibilidade com os parâmetros formais correspondentes.

Nos casos de seção de parâmetros reais de parâmetro procedimento ou parâmetro função a compatibilidade não é verificada, pois não existe a seção de parâmetros formais correspondente, sendo apenas verificado se todos os parâmetros reais são expressões. No caso de parâmetro matricial não é verificada a compatibilidade do dimensionamento dos arrays.

Esta rotina é chamada pelo Reconhecedor de Comandos (PLAPP17) e pelo Reconhecedor de Expressão (PLAPP30) com TIPO contendo o código do primeiro símbolo após o nome da procedure ou função, mesmo que esta não tenha parâmetros, e retorna com o código do símbolo seguinte ao ")" que encerra a seção de parâmetros reais ou com o valor de TIPO inalterado.

Esta rotina recebe como parâmetro o apontador para o nome o procedimento ou função na tabela de símbolos e devolve o endereço do temporário que conterá o resultado da função, no caso de função.

Sequência a ser analisada:

$\{ \; (\; \langle \text{parâmetro real} \rangle \; \{ \; , \; \langle \text{parâmetro real} \rangle \; \}_0^n \;) \; \}_0^1$

onde:

<parâmetro real> ::= <expressão> |
<identificador de procedimento> |
<identificador de função>

As expressões são analisadas pelo Reconhecedor de Expressão (PLAPP30).

Para efeito de análise um parâmetro real só não será considerado como expressão se o primeiro símbolo do parâmetro real for um identificador de procedimento ou função e o parâmetro formal correspondente for definido como procedimento ou função, respectivamente.

GERAÇÃO DE CÓDIGO

É efetuada nesta rotina a geração de código para ativação de procedimentos e funções, mesmo que se trate de procedimento ou função sem parâmetro, consistindo o código gerado em instruções para: montar, na área estática inteira, a lista de chamada; colocar na área de ligação do procedimento ou função o endereço absoluto desta lista e para desviar para a primeira instrução do procedimento ou função.

É utilizado o seguinte algorítmo para geração de código:

1) Para cada parâmetro real analisado pelo Reconhecedor de Expressão são geradas por êle as instruções relativas à expressão e:

1.1) Se a expressão se resumir a um elemento de array com índice variável ou um elemento de parâmetro array por valor com índice variável, são geradas as seguintes instruções

para cálculo do endereço do elemento:

```
ZEREG       1,1,1
ADI         (0,ENDVAR),(BXUOPND,XUOPND),(BXUOPND,XUOPND)
```

1.2) Se a expressão for constante, são geradas as seguintes instruções para colocar num temporário o valor resultante da expressão:

```
ZEREG       0,1,1
ATRIB       TIPEXP,(0,ENDEXP),(BTEMP,TEMP)
```

1.3) Se o parâmetro não for uma expressão composta apenas de: um elemento de parâmetro array por endereço, um parâmetro variável simples por endereço ou um parâmetro array por endereço, são geradas as seguintes instruções para calcular o endereço absoluto do parâmetro:

```
STOB        TIPB,BEND1,(BTEMP1,TEMP1)
ZEREG       1,1,1
ADI         (BEND1,end1),(BTEMP1,TEMP1),(BTEMP1,TEMP1)
```

2) Se a seção de parâmetros reais que está sendo analisada pertencer a um procedimento ou função não parâmetro, isto é, que não tenha sido recebido como parâmetro por outro procedimento ou função externo à chamada, são geradas as seguintes instruções para colocar na área de ligação o endereço absoluto da lista de parâmetros:

```
STOB        5,NIVEL1,(0,ENDLIG)
ZEREG       1,1,1
ADI         (0,ENDLIG),(0,ENDLISTA),(0,ENDLIG)
```

3) Se a seção de parâmetros reais que está sendo analisada pertencer a uma parâmetro procedimento ou parâmetro função são geradas as seguintes instruções para colocar o endereço absoluto da lista de chamada na área de ligação:

```
STOB        5,BTEMP3,(BTEMP3,TEMP3)
SETM        (0,ENDZERO),(0,ENDZERO),(BTEMP4,TEMP4)
ADI         (BTEMP3,TEMP3),(0,ENDLISTA),(0,0)
```

4) Se a seção de parâmetros reais que está sendo analisada pertencer a uma função ou parâmetro função são geradas as seguintes instruções para colocar na lista de chamada o endere

ço absoluto do temporário que receberá o resultado da função:

```
STOB        T,BTEMP5,(BTEMP5,TEMP5+1)
ZEREG       1,1,1
ADI         (0,ENDRESUL),(BTEMP5,TEMP5+1),(BTEMP5,TEMP5+1)
```

5) São geradas as seguintes instruções para mover para a lista de chamada as informações relativas aos parâmetros reais:

```
ZEREG       1,1,1
```

5.1) Para cada parâmetro real são geradas as instruções:

5.1.1) `ATRIB       1,(BPAR,ENDPAR),(BTEMP5,PTEMP5)`

5.1.2) Se o parâmetro real for um identificador de procedimento ou de função são geradas as instruções:

```
ATRIB       1,(0,ENDLIGPAR),(BTEMP5,PTEMP5)
ATRIB       1,(0,NIVPAR),(BTEMP5,PTEMP5)
```

6) Gera instrução para colocar na lista de chamada o endereço de retorno:

```
ATRIB       1,(0,RETORNO),(BTEMP5,TEMP5)
```

7) Gera as seguintes instruções para efetuar o desvio para o procedimento ou função cuja seção de parâmetros reais está sendo analisada:

7.1) Se o procedimento ou função não for parâmetro:

7.1.1) Se o nível do procedimento ou função chamado for menor que o nível do bloco que está sendo analisado é gerada a instrução abaixo para colocar uma nova matriz no topo da pilha de matrizes de registradores base:

```
EMPB        (0,NIVEL)
```

7.1.2) É gerada a instrução:

```
DESV        0,CCQQR,ENDER                                   (1)
```

7.2) Se o procedimento ou função for parâmetro são geradas as seguintes instruções para colocar uma nova matriz no topo da pilha de matrizes de registradores base e para efetuar o desvio:

```
SET     3,(BTEMP6,TEMP6)
EMPB    (BTEMP2,TEMP2)
DESV    1,CCQQR,1                                    (2)
```

8) Se o ítem 7.1.1 ou 7.2 acima foi executado, gera a seguinte instrução para retirar do topo da pilha de matrizes de registradores base a última matriz empilhadada:

```
DESB
```

onde:

ENDVAR - endereço da constante inteira com o endereço do array.

XUOPND - endereço do temporário inteiro que contém o deslocamento calculado para o elemento.

BXUOPND - número do registrador base relativo a XUOPND.

TIPEXP - tipo da expressão (1 - inteira, 2 - real, 3 - booleana, 4 - caráter).

ENDEXP - endereço da constante resultante da expressão.

TEMP - endereço de um temporário de mesmo tipo que a expressão.

BTEMP - número do registrador base relativo a TEMP.

TEMP1 - endereço de um temporário de mesmo tipo que a expressão.

BTEMP1 - número do registrador base relativo a TEMP1.

end1 - endereço do parâmetro. Se o parâmetro corresponder ao ítem 1.1 acima, end1 será igual a XUOPND. Se corresponder ao ítem 1.2, será igual a TEMP. Nos demais casos será o endereço do resultado da expressão.

BEND1 - número do registrador base relativo a end1.

TIPB - número da coluna da matriz de registradores base correspondente a end1 (5 - inteiro, 6 - real, 7 - booleano, 8 - caráter, se end1 for um temporário e 1 - inteiro, 2 - real, 3 - booleano, 4 - caráter, em caso contrário).

NIVEL - endereço da constante inteira com o nível do procedimento ou função cuja seção de parâmetros reais está sendo analisada.

NIVEL1 - nível do bloco que está sendo analisado.

ENDLIG - endereço da área de ligação do procedimento ou função cuja seção de parâmetros reais está sendo analisada.

ENDLISTA - endereço da constante inteira com o endereço da lista de chamada do procedimento ou função cuja seção de parâmetros reias está sendo analisada.

TEMP2 - endereço do temporário inteiro que contém o nível do parâmetro procedimento ou parâmetro função, cuja seção de parâmetros reais está sendo analisada.

BTEMP2 - número do registrador base relativo a TEMP2.

TEMP3 - endereço de um temporário inteiro do bloco que está sendo analisado.

BTEMP3 - número do registrador base relativo a TEMP3.

ENDZERO - endereço da constante inteira zero.

TEMP4 - endereço do temporário inteiro que contém o endereço da área de ligação do parâmetro procedimento ou parâmetro função.

BTEMP4 - número do registrador base relativo a TEMP4.

T - número da coluna da matriz de registradores base correspondente do tipo da função (5 - inteira, 6 - real, 7 - booleana, 8 - caráter).

TEMP5 - endereço da primeira posição da lista de chamada. A lista de chamada será formada por um conjunto contíguo de temporários inteiros.

BTEMP5 - número do registrador base relativo à lista de chamada.

ENDRESUL - endereço da constante inteira com o endereço do temporário, de mesmo tipo que a função, que receberá o resultado da função.

ENDPAR - endereço da posição de memória inteira que contém: o endereço absoluto do parâmetro ou o endereço (-1) da primeira instrução do procedimento ou função no caso de parâmetro procedimento ou parâmetro função.

BPAR - número do registrador base relativo a ENDPAR.

PTEMP5 - endereço da próxima posição disponível da lista de chamada (endereço da posição anterior + 1).

ENDLIGPAR - endereço da constante inteira com o endereço da área de ligação do parâmetro procedimento ou parâmetro função.

NIVPAR - endereço da constante inteira com o nível do parâmetro procedimento ou parâmetro função.

RETORNO - endereço da constante inteira com o endereço da primeira instrução gerada: após a instrução (1) se o procedimento ou função cuja seção de parâmetros reais está sendo analisada não for parâmetro ou a instrução (2) caso contrário.

ENDER - endereço da primeira instrução do procedimento ou função cuja seção de parâmetros reais está sendo analisada.

TEMP6 - endereço do temporário inteiro com o endereço (-1) do procedimento ou função cuja seção de parâmetros reais está sendo analisada.

BTEMP6 - número do registrador base relativo a TEMP6.

3.2.2.18. RECONHECEDOR DE ÍNDICES DE ARRAYS - PLAPP29

ANÁLISE

A função desta rotina é analisar índices de arrays, verificando a compatibilidade com a declaração do array.

É chamada pelo Reconhecedor de Comandos (PLAPP17), pelo Reconhecedor de Comando READ (PLAPP26) e pelo Reconhecedor de Expressão (PLAPP30), com TIPO contendo o código correspondente ao "[" que segue o nome do array e retorna com o código do símbolo seguinte ao "]" que encerra a lista de índices.

Esta rotina recebe como parâmetro (PT1) a posição do nome do array na tabela de símbolos e devolve três outros parâmetros (ENDIND, TIPEND e BASE) que serão descritos adiante, no tópico Geração de Código pois não dizem respeito à análise.

Sequência a ser analisada:

[<expressão> { , <expressão> }$_{0}^{n}$]

As expressões são analisadas pelo Reconhecedor de Expressão (PLAPP30).

GERAÇÃO DE CÓDIGO

Esta rotina gera código para calcular o deslocamento do elemento com relação ao início do array e para calcular os valores dos tres parâmetros devolvidos.

Nos casos em que todas as expressões correspondentes aos índices são constantes, o cálculo do deslocamento do elemento com relação ao início do array é efetuado durante a compilação. Para os demais casos é gerada uma instrução CDE.

Para execução da instruçao CDE é gerada durante a compilação uma lista, na área estática inteira, que descreve os índices reconhecidos (a outra lista necessária, lista descritora do array, já foi previamente gerada pelo Reconhecedor de Tipo de Variáveis - PLAPP18 - onde a mesma está descrita).

Existem nesta lista tres entradas para cada índice: o tipo do endereço do índice; o endereço do índice e o número do registrador base relativo a este endereço.

O tipo do endereço do índice indica se o endereço é direto (1, 2 ou 3) ou indireto (4). Para os índices com tipo do endereço direto o endereço do índice é o próprio deslocamento do índice. Caso contrário é o deslocamento do temporário inteiro que contém o endereço absoluto do índice.

O tipo de endereço de um índice é indireto se a expressão correspondente a este índice se resumir a um parâmetro variável simples por endereço, um elemento de array com índice variável, um elemento de parâmetro array por valor com índice variável ou um elemento de parâmetro array por endereço, sendo direto nos demais casos.

É utilizado o seguinte algorítmo para a geração de código para o cálculo do deslocamento do elemento:

1) Para cada índice são geradas pelo Reconhecedor de Expressão as instruções relativas à expressão e se a mesma se resumir a um elemento de array com índice variável ou a um elemento de parâmetro array por valor com índice variável são geradas as instruções:

```
ZEREG       1,1,1
ADI         (0,ENDVAR),(BXUOPND,XUOPND),(BXUOPND,XUOPND)
STOB        1,BVAR,(BTEMP1,TEMP1)
ADI         (BTEMP1,TEMP1),(BXUOPND,XUOPND),(BXUOPND,XUOPND)
```

2) Se a lista de índices pertence a um array ou a um parâmetro array por valor:

2.1) Se todos os índices são constantes o deslocamento do elemento dentro do array é calculado durante a compilação e este valor é somado ao endereço do array, resultando no endereço do elemento. É devolvido (em ENDIND) este endereço, (em TIPEND) a indicação de que se trata de uma variável com endereço direto e (em BASE) o número do registrador base relativo ao array.

2.2) Se pelo menos um dos índices é variável é gerada a instrução abaixo para calcular o deslocamento do elemento, sendo devolvido (em ENDIND) o deslocamento do temporário inteiro que conterá o deslocamento calculado para o elemento, (em TIPEND) a indicação de que se trata de uma variável com endereço indireto e (em BASE) o número do registrador base relativo a este temporário:

```
CDE         ENDLISTA1,ENDLISTA2,(BTEMP2,TEMP2)
```

3) Se a lista de índices pertence a um parâmetro array por endereço:

3.1) Se todos os índices são constantes o deslocamento do elemento dentro do array é calculado durante a compilação e são geradas as instruções abaixo para calcular o endereço absoluto do elemento, sendo devolvido (em ENDIND) o deslocamento do temporário inteiro que conterá o endereço absoluto do elemento, (em TIPEND) a indicação de que se trata de um pa-

râmetro e (em BASE) o número do registrador base relativo a este temporário:

```
ZEREG       1,1,1
ADI         (BTEMP3,TEMP3),(0,DESLOC),(BTEMP4,TEMP4)
```

3.2) Se pelo menos um dos índices é variável são geradas as instruções abaixo para calcular o endereço absoluto do elemento, sendo devolvido (em ENDIND) o deslocamento do temporário inteiro que conterá o endereço absoluto do elemento, (em TIPEND) a indicação de que se trata de um parâmetro e (em BASE) o número do registrador base relativo ao temporário:

```
CDE         ENDLISTA1,ENDLISTA2,(BTEMP4,TEMP4)
ZEREG       1,1,1
ADI         (BTEMP3,TEMP3),(BTEMP4,TEMP4),(BTEMP4,TEMP4)
```

onde:

ENDVAR - endereço da constante inteira com o endereço do array.

XUOPND - endereço do temporário inteiro que contém o deslocamento calculado para o elemento.

BXUOPND - número do registrador base relativo a XUOPND.

BVAR - número do registrador base relativo ao array.

TEMP1 - endereço do temporário inteiro que conterá o endereço absoluto do elemento.

BTEMP1 - número do registrador base relativo a TEMP1.

ENDLISTA1 - endereço da lista descritora do array.

ENDLISTA2 - endereço da lista de descritores dos índices.

TEMP2 - deslocamento do temporário inteiro que conterá o deslocamento calculado para o elemento.

BTEMP2 - número do registrador base relativo a TEMP2.

TEMP3 - deslocamento do temporário inteiro que contém o endereço absoluto do parâmetro array.

BTEMP3 - número do registrador base relativo a TEMP3.

DESLOC - endereço da constante inteira com o deslocamento calculado para o elemento.

TEMP4 - deslocamento do temporário inteiro que conterá o endereço absoluto do elemento.

BTEMP4 - número do registrador base relativo a TEMP4.

3.2.2.19. RECONHECEDOR DE EXPRESSÃO - PLAPP30

ANÁLISE

A função desta rotina é analisar expressões, fornecendo para os demais reconhecedores informações a respeito da mesma.

É chamada pelo Reconhecedor de Comandos (PLAPP17), pelo Reconhecedor de Comando REPEAT (PLAPP21), pelo Reconhecedor de Comando WHILE (PLAPP22), pelo Reconhecedor de Comando IF (PLAPP23), pelo Reconhecedor de Comando CASE (PLAPP24), pelo Reconhecedor de Comando FOR (PLAPP25), pelo Reconhecedor de Comando WRITE (PLAPP27), pelo Reconhecedor de Seção de Parâmetros Reais (PLAPP28) e pelo Reconhecedor de Índices de Arrays (PLAPP29) com TIPO contendo o código do primeiro elemento da expressão e retorna com o código do primeiro símbolo estranho à mesma, que será analisado pela rotina que chamou esta.

As informações fornecidas por este reconhecedor, relativas à expressão analisada são:

1) O tipo da expressão (variável TIPEXP), segundo tabela:

TIPEXP	Tipo da Expressão
-1	tipo indefinido
0	não matricial de tipo indefinido
1	não matricial de tipo inteiro
2	não matricial de tipo real
3	não matricial de tipo booleano
4	não matricial de tipo caráter
50	matricial de tipo indefinido
51	matricial de tipo inteiro
52	matricial de tipo real
53	matricial de tipo booleano
54	matricial de tipo caráter
100	parâmetro matricial de tipo indefinido
101	parâmetro matricial de tipo inteiro
102	parâmetro matricial de tipo real
103	parâmetro matricial de tipo booleano
104	parâmetro matricial de tipo caráter

2) Se a expressão for matricial, o ponteiro para a tabela de dimensões, indicando as dimensões da matriz resultante;

3) Se a expressão resultante for do tipo caráter, o tamanho da expressão em caracteres.

Sequência a ser analisada:

<expressão> ::= <expressão simples> | <expressão simples> <operador relacional> <expressão simples>

onde:

<operador relacional> ::= = | ¬= | < | > | <= | >=

<expressão simples> ::= <termo> | + <termo> | - <termo> | <expressão simples> <operador de adição> <termo>

<operador de adição> ::= + | - | OR[1]

<termo> ::= <fator> | <termo> <operador de multiplicação> <fator>

<operador de multiplicação> ::= * | / | DIV | MOD | &

<fator> ::= <variável> | <constante sem sinal> | (<expressão>) | <ativação de função> | ¬ <fator>

Na análise das expressões é empregado o mesmo método já apresentado em 3.2.2.8., na descrição do Reconhecedor de Expressão Tipo 1 (PLAPP19).

Uma vez que as expressões aqui analisadas são mais complexas que as do tipo 1, foram necessárias as seguintes alterações no método anteriormente descrito:

1) Novo conjunto de operadores:

Operadores	Prioridade Estática
relacionais	1
de adição	2
de multiplicação	3
¬	4

---

1) o operador | foi substituido aqui por OR para evitar ambiguidade.

2) Na pilha são armazenados sempre os endereços de operandos, ao invés de seus valores.

3) Na redução da expressão é gerado código para a operação em questão. Os resultados intermediários são sempre colocados em temporários, cujos endereços são inseridos na pilha. As operações entre valores constantes são efetuadas durante a compilação, sendo inserido na pilha o endereço da constante resultante.

4) A pilha usada neste reconhecedor contém os seguintes campos:

OPER	PRIOR	OPAND	TIPOPAND	TIPEND	ENDINDEX	TCHAR	BOPAND	BINDEX

onde:

OPER - código do operador.

PRIOR - prioridade dinâmica do operador (prioridade estática + valor da base no momento da inserção).

OPAND - endereço do operando.

TIPOPAND - tipo do operando codificado com os mesmos valores usados para tipo da expressão.

TIPEND - tipo do endereço do operando, se o operando não for matricial, ou um ponteiro para a tabela de dimensões de arrays, se o operando for matricial. No primeiro caso pode assumir um dos 3 valores: constante - se o operando for uma constante; direto - se o operando for um temporário, uma variável simples, um elemento de array (ou de um parâmetro array por valor) com índice constante ou um parâmetro variável simples por valor; e indireto - se o operando for um elemento de array (ou de um parâmetro array por valor) com índice variável, um parâmetro variável simples por endereço ou um elemento de parâmetro array por endereço (independente do índice).

ENDINDEX - endereço do valor a ser carregado no registrador indexador, para referência ao operando. Se o tipo do endereço do operando for constante ou direto (ver TIPEND) é o endereço da constante inteira zero.

TCHAR - tamanho em caracteres, se o operando é caráter, ou contém o valor 1, caso contrário.

BOPAND - número do registrador base relativo ao operando.
BINDEX - número do registrador base relativo ao endereço contido em ENDINDEX.

5) As operações são sempre efetuadas entre o operando anterior ao do topo da pilha - OPAND(TOPO - 1) - e o do topo OPAND(TOPO) - nesta ordem.

Algumas operações envolvendo matrizes foram previstas, constam da análise da expressão, mas não foram implementadas, ou seja, não constam da geração de código.

Abaixo, na relação dessas operações, S indica um escalar, inteiro ou real, A indica um array, inteiro ou real e todas as operações tem como resultado um array temporário, real se um dos operandos ou os dois são reais e inteiro se os dois operandos são inteiros:

S + A - a cada elemento de A é somado o valor de S.
S * A ou A * S - cada elemento de A é multiplicado pelo valor de S.
A1 $\pm$ A2 - a cada elemento de A1 é somado ou subtraido o valor do elemento correspondente de A2.
A $\pm$ S - a cada elemento de A é somado ou subtraido o valor de S.

GERAÇÃO DE CÓDIGO

Para facilitar a apresentação, a geração de código para expressões está separada por classes de operações.

Algumas variáveis que são usadas na descrição da geração de código para mais de uma classe de operações, serão descritas aqui:

TOPO - ponteiro para a última posição ocupada na pilha.
TEMBOOL - endereço de um temporário booleano do bloco que contém a expressão.
BTBOOL - número do registrador base relativo a TEMBOOL.
EFALSE - endereço da constante booleana FALSE.
ETRUE - endereço da constante booleana TRUE.
ENDZERO - endereço da constante inteira zero.

TEMREAL, TEMREAL1 e TEMREAL2 - endereços de temporários reais do bloco que contém a expressão.

BTREAL - número do registrador base relativo a TEMREAL, TEMREAL1 e TEMREAL2.

TEMINT - endereço de um temporário inteiro do bloco que contém a expressão.

BTINT - número do registrador base relativo a TEMINT.

1) Para o operador ¬ (NOT), são geradas as instruções:

```
SET     2,(BINDEX(TOPO),ENDINDEX(TOPO))
NOT     (BOPAND(TOPO),OPAND(TOPO)),(BTBOOL,TEMBOOL)
```

2) Para os operadores relacionais (¬=, =, <, >, <= e >=) entre operandos de mesmo tipo, são geradas as instruções:

```
SET     2,(BINDEX(TOPO-1),ENDINDEX(TOPO-1))
SET     3,(BINDEX(TOPO),ENDINDEX(TOPO))
COMP    tipo,(BOPAND(TOPO-1),OPAND(TOPO-1)),
            (BOPAND(TOPO),OPAND(TOPO))
ZEREG   1,1,1
ATRIB   3,(0,EFALSE),(BTBOOL,TEMBOOL)
DESV    0,cod,E1
ZEREG   1,1,1
ATRIB   3,(0,ETRUE),(BTBOOL,TEMBOOL)                    (1)
```

onde:

tipo - tipo correspondente aos operandos (1 para inteiros, 2 para reais, 3 para booleanos e 4 para caracteres).

cod - um dos seguintes códigos de condição, dependendo do operador: igual, menor ou igual, maior ou igual, diferente, menor, maior.

E1 - endereço da instrução seguinte a (1).

3) Para operadores & e | entre operandos booleanos, são geradas as instruções:

```
SETM    (BINDEX(TOPO-1),ENDINDEX(TOPO-1)),
            (BINDEX(TOPO),ENDINDEX(TOPO)),(0,ENDZERO)
cod     (BOPAND(TOPO-1),OPAND(TOPO-1)),
            (BOPAND(TOPO),OPAND(TOPO)),(BTBOOL,TEMBOOL)
```

onde cod é o código de operação correspondente ao operador (AND ou OR).

4) Para operadores relacionais entre operandos numéricos, sendo um real e o outro inteiro (é consideredo aqui que o operando inteiro ocupa a posição PT na pilha):

4.1) São geradas as seguintes instruções para converter para real o operando inteiro:

```
SET     2,(BINDEX(PT),ENDINDEX(PT))
SET     3,(0,ENDZERO)
CVT     2,(BOPAND(PT),OPAND(PT)),(BTREAL,TEMREAL)
```

4.2) O endereço do temporário resultante da conversão (TEMREAL), o número do registrador base correspondente (BTREAL), o endereço da constante inteira zero (ENDZERO) e o valor zero são colocados , respectivamente, nos campos OPAND, BOPAND, ENDINDEX e BINDEX da pilha, na posição PT.

4.3) São geradas as mesmas instruções geradas para operadores relacionais entre operandos de mesmo tipo, descritas no ítem 2.

5) Para operadores aritméticos (+, -, * e /) entre dois operandos reais, são geradas as instruções:

```
SETM    (BINDEX(TOPO-1),ENDINDEX(TOPO-1)),
             (BINDEX(TOPO),ENDINDEX(TOPO)),(0,ENDZERO)
cod     (BOPAND(TOPO-1),OPAND(TOPO-1)),
             (BOPAND(TOPO),OPAND(TOPO)),(BTREAL,TEMREAL)
```

onde cod é o código de operação correspondente ao operador (ADR, SUBR, MLTR ou DIVR).

6) Para operadores aritméticos (+, -, * e /) entre operandos numéricos, sendo um real e o outro inteiro (é considerado aqui que o operando inteiro ocupa a posição PT na pilha):

6.1) São geradas as seguintes instruções para converter para real o operando inteiro:

```
SET     2,(BINDEX(PT),ENDINDEX(PT))
SET     3,(0,ENDZERO)
CVT     2,(BOPAND(PT),OPAND(PT)),(BTREAL,TEMREAL)
```

6.2) O endereço do temporário resultante da conversão (TEMREAL), o número do registrador base correspondente (BTREAL), o endereço da constante inteira zero (ENDZERO) e o valor zero são colocados, respectivamente, nos campos OPAND, BOPAND, ENDINDEX e BINDEX da pilha, na posição PT.

6.3) São geradas as mesmas instruções geradas para operadores aritméticos entre operandos reais, descritas no ítem 5.

7) Para operadores aritméticos (+, -, *, DIV e MOD) entre operandos inteiros, são geradas as instruções:

```
SETM    (BINDEX(TOPO-1),ENDINDEX(TOPO-1)),
             (BINDEX(TOPO),ENDINDEX(TOPO)),(0,ENDZERO)
cod     (BOPAND(TOPO-1),OPAND(TOPO-1)),
             (BOPAND(TOPO),OPAND(TOPO)),(BTINT,TEMINT)
```

onde cod é o código de operação correspondente ao operador (ADI, SUBI, MLTI, DIVI ou MOD).

8) Para o operador / entre dois operandos inteiros:

8.1) São geradas as seguintes instruções para converter para real os dois operandos:

```
SET     2,(BINDEX(TOPO-1),ENDINDEX(TOPO-1))
SET     3,(0,ENDZERO)
CVT     2,(BOPAND(TOPO-1),OPAND(TOPO-1)),(BTREAL,TEMREAL1)
SET     2,(BINDEX(TOPO),ENDINDEX(TOPO))
SET     3,(0,ENDZERO)
CVT     2,(BOPAND(TOPO),OPAND(TOPO)),(BTREAL,TEMREAL2)
```

8.2) Os endereços dos temporários resultantes das conversões (TEMREAL1 e TEMREAL2), o número do registrador base correspondente (BTREAL), o endereço da constante inteira zero (ENDZERO) e o valor zero são colocados, respectivamente, nos campos OPAND, BOPAND, ENDINDEX e BINDEX da pilha, nas posições TOPO-1 e TOPO, respectivamente.

8.3) São geradas as instruções:

```
SETM    (BINDEX(TOPO-1),ENDINDEX(TOPO-1)),
             (BINDEX(TOPO),ENDINDEX(TOPO)),(0,ENDZERO)
DIVR    (BOPAND(TOPO-1),OPAND(TOPO-1)),
             (BOPAND(TOPO),OPAND(TOPO)),(BTREAL,TEMREAL)
```

3.2.3. ROTINAS DE APOIO

3.2.3.1. ROTINA PARA CÁLCULO DA FUNÇÃO HASH - PLAPA03

A função HASH utilizada para manipulação da tabela de símbolos consiste em tres ou-exclusivos dos "bytes" do identificador, desprezando-se no final os dois "bits" mais à esquerda.

O identificador é dividido em duas partes iguais, sendo efetuado o ou-exclusivo entre a primeira metade e a segunda. Como o identificador é sempre formado por uma cadeia de 8 "bytes", resulta desta primeira iteração uma cadeia de 4 "bytes".

O processo é repetido para a cadeia resultante da iteração anterior, e mais uma vez, quando é obtida então uma cadeia de um único "byte".

Os dois "bits" mais à esquerda são desprezados, resultando assim em um número binário de 6 "bits", o que corresponde a um número decimal entre 0 e 63, inclusives.

3.2.3.2. ROTINA PARA IMPRESSÃO DE MENSAGENS DE ERRO - PLAPP01

A função desta rotina é imprimir as mensagens relativas aos erros encontrados no programa fonte durante a compilação.

A impressão da mensagem é precedida por uma linha de asteriscos, contendo um indicador para a posição aproximada do erro no texto do programa e o código do erro.

O código do erro é transmitido para essa rotina através da variável ERRO e a posição aproximada do erro, na variável COLUNA.

As mensagens de erro tem tamanho variável, sendo compostas por blocos de 53 caracteres, armazenados em uma tabela dupla, conforme descrito no tópico 'Estrutura de Dados'.

A tabela de erros de compilação é percorrida a partir da entrada correspondente ao código do erro, seguindo pelos apontadores de continuação até o final da mensagem. Em cada linha são impressos dois blocos do texto da mensagem.

3.2.3.3. ROTINA PARA IMPRESSÃO DA TABELA DE SÍMBOLOS - PLAPP02

Esta rotina imprime a tabela de símbolos para efeito de depuração do compilador.

Recebe em AVAIL um apontador para o início da parte livre da tabela e imprime apenas o conteúdo da parte ocupada tanto da tabela propriamente dita como do vetor cabeça de "hash" (CABEC).

São impressos os conteúdos dos campos: NOME, NIVEL, LINK, IDTIPO, PTCTE e ENDER.

3.2.3.4. RECONHECEDOR DAS OPÇÕES DE COMPILAÇÃO - PLAPP03

Esta função analisa os parâmetros recebidos pelo Módulo Principal do Compilador (PLAPP00), que especificam as opções de compilação desejadas pelo programador (separadas por vírgulas e a última seguida por pelo menos um branco, isto é, um branco indica fim da lista de opções) e efetua as inicializações necessárias. Retorna com '1'B em caso de erro grave e '0'B caso contrário.

Foi implementada apenas a opção TRACE, que ativa os diversos "traces" contidos no compilador para auxiliar na depuração. São disponíveis 16 "traces" diferentes, dos quais 5 são usados, com as seguintes funções:

"trace" 1 - usado para dar o fluxo e imprimir os valores dos parâmetros e das principais variáveis transmitidas entre os módulos do compilador.

"trace" 2 - usado pelas rotinas de manipulação da tabela de símbolos (PLAPP05, PLAPP06 e PLAPP07, respectivamente rotinas de busca, inserção e retirada), para impressão da tabela e do valor da função "hash".

"trace" 3 - usado pelo Reconhecedor de Expressão (PLAPP30) para imprimir o conteúdo da pilha usada para análise de expressões.

"trace" 4 - usado pelo Módulo Principal do Compilador (PLAPP00) para inibir a execução do programa objeto gerado.

"trace" 5 - usado pelo módulo interpretador (PLAPP32) para imprimir o programa depois de carregado na memória (áreas de variáveis, constantes e instruções).

Formatos da opção TRACE:

1) TRACE=n

2) TRACE=(n)

3) TRACE=(n_1,n_2,...,n)

4) TRACE=(n_1-n_2)

onde n, n_1 e n_2 são números entre 1 e 16, inclusives, representando os "traces" que se deseja ativar.

Nos formatos 1 e 2, é ativado o "trace" n. No formato 3 são ativados os "traces" n_1, n_2,..., n e no formato 4 são ativados os "traces" n_1 até n_2, inclusives.

Os formatos 3 e 4 podem ser combinados, por exemplo: TRACE=(n_1,n_2-n_3).

A única restrição é com relação à ordenação nos formatos que indicam intervalo, onde o limite superior tem que ser maior que o inferior.

3.2.3.5. RECONHECEDOR LÉXICO - PLAPP04

As funções desta rotina são: ler e imprimir os cartões do programa fonte e reconhecer os elementos léxicos da linguagem, pulando brancos supérfluos e comentários.

Os elementos léxicos podem ser grupados em: palavras reservadas, identificadores, números inteiros, números reais, literais e símbolos especiais.

Esta rotina devolve, a cada chamada, o próximo elemento léxico existente na sequência de entrada bem como a categoria a que o mesmo pertence.

Ao ser detetado o símbolo "%" todos os caracteres que o seguem até o próximo "%" são considerados comentários, sendo simplesmente impressos.

Fim de cartão não é considerado separador nem delimitador de elemento léxico.

Tratamento dos elementos léxicos:

Literais

Ao ser detetado um delimitador de literais é considerado que o próximo elemento é um literal. Todos os caracteres que o seguem, até o próximo delimitador, ou, na falta dele os seguintes 256 caracteres são lidos e atribuidos ao literal.

No caso de truncamento do literal (mais de 256 caracteres) os próximos caracteres são ignorados até o símbolo ";" e é impressa uma mensagem de erro. Este reconhecedor devolve à rotina que o chamou o literal lido, no mesmo formato, no array STRING, seu tamanho na variável CONT e em TIPO o código correspondente a literal.

Identificadores e Palavras Reservadas

Ao ser detetada uma letra é considerado que o próximo elemento léxico é um identificador. Os próximos caracteres são lidos até ser encontrado um diferente de letra ou dígito, que indica fim do identificador. Não há portanto, identificadores contendo caracteres especiais.

Se o identificador contiver mais de oito caracteres, será truncado para os oito primeiros e uma mensagem de erro será emitida.

Após o reconhecimento do identificador é feita uma busca na tabela de palavras reservadas para verificar se se trata de um identificador ou de uma palavra reservada.

A tabela de palavras reservadas (TABRES) é um array de 59 elementos, de 8 caracteres cada (completados com brancos quando a palavra reservada tem menos de 8 caracteres) que contém as palavras reservadas em ordem alfabética crescente. O código de cada uma é o conteúdo do elemento de mesmo índice do vetor paralelo ATIPO, conforme tabela a seguir:

TABRES	ATIPO	TABRES	ATIPO	TABRES	ATIPO
ABS	99	FORWARD	80	REAL	77
ARCTAN	99	FUNCTION	55	REPEAT	60
ARRAY	57	GOTO	66	ROUND	99
BEGIN	56	IF	68	SIN	99
BOOLEAN	78	IN	41	SQR	99
CASE	64	INTEGER	76	SQRT	99
CHAR	79	LABEL	51	SUCC	99
CHR	99	LN	99	THEN	69
CONST	52	MOD	25	TO	74
COS	99	NEW	65	TRUNC	99
DIV	24	ODD	99	UNTIL	63
DO	72	OF	58	VAR	53
DOWNTO	73	ORD	99	WHILE	71
ELSE	70	PRED	99	WRITE	81
END	67	PROC	54	WRITED	84
EOF	99	PROGRAM	50	WRITELN	82
EOLN	99	READ	91	WRITELND	85
EOPG	99	READD	93	WRITEPG	83
EXP	99	READP	92	WRITEPGD	86
FOR	75	READPD	94		

Observação: o código 99 indica uma função interna ainda não implementada.

Esta rotina devolve o identificador na variável IDENT. Em TIPO retornará o código correspondente a identificador ou à palavra reservada, conforme o caso.

No caso de truncamento do identificador, todos os próximos caracteres são ignorados, até um diferente de letra ou dígito.

Números Inteiros e Reais

Ao ser detetado um dígito é considerado que o próximo elemento é um número. Imediatamente após um ou uma série de dígitos, ao ser detetado um ponto ou a letra "E" é considerado que o número é real.

É fim normal de número real a existência de qualquer caráter pertencente ao conjunto A = { + - < > = ¬)] , ; * / & | % : branco } após os dígitos que sucedem o ponto ou que sucedem a letra "E".

É considerado erro no número real:

- qualquer caráter diferente de dígito imediatamente após o ponto;
- qualquer caráter diferente de +, - e dígito imediatamente após a letra "E";
- qualquer caráter diferente de dígito seguindo o sinal que segue a letra "E" ou
- qualquer caráter não pertencente ao conjunto A definido acima e diferente da letra "E" após o(s) dígito(s) que segue(m) o ponto.

É considerado fim normal de número inteiro a existência de qualquer caráter pertencente ao conjunto A imediatamente após um ou uma série de dígitos.

Em caso de erro no número é emitida uma mensagem e os próximos caracteres são ignorados até ser encontrado um caráter pertencente ao conjunto A.

Esta rotina devolve o número real transformado para o formato ponto flutuante na variável NREAL e o número inteiro na variável NINT, no formato inteiro.

Em caso de erro no número inteiro ou real são assumidos valores válidos para NINT e NREAL, para permitir a continuação da análise sem geração de futuras mensagens de erros redundantes, conforme tabela:

Em caso de	É devolvido
erro no número inteiro	NINT = 1
"overflow" em número inteiro	NINT = 2147483647
erro em número real	NREAL = 1.0
"overflow" em número real	NREAL = 7237008 * 10**69
"underflow" em número real	NREAL = 0.0

Símbolos Especiais

Ao ser detetado um símbolo especial é colocado em TIPO o código do mesmo. Se o símbolo for >, <, : ou ¬, o próximo caráter é analisado para verificar se se trata de um dos operadores duplos >=, <=, ¬= ou :=.

É considerado fim do arquivo que contém o programa fonte a ocorrência do fim físico do arquivo SCARDS ou de um registro com o caráter $ na posição 1, neste arquivo.

Ao ser detetado o fim do arquivo, se já tiver sido iniciado o reconhecimento de algum elemento léxico, a análise é encerrada normalmente. Caso contrário a análise é abortada retornando TIPO com o valor zero. Em ambos os casos a variável EOF retornará com o valor '1'B (indicação de fim de arquivo).

Os códigos dos elementos léxicos estão apresentados na descrição da macro PLAMV01, que define a estrutura AUXSCAN, de variáveis auxiliares para comunicação entre o Reconhecedor Léxico e as demais rotinas.

GETCAR

O Reconhecedor Léxico utiliza a rotina GETCAR, local ao reconhecedor, cujas funções são: extrair o próximo caráter da sequência de entrada, colocando-o na variável CAR; atribuir à variável CLASSE um código de acordo com o caráter lido e posicionar o próximo caráter a ser lido.

A sequência de entrada é considerada como sendo da posição 1 à 72 do registro do arquivo SCARDS, inclusives. Ao ser extraido o último caráter de um registro, o próximo registro é lido e impresso. Ao ser detetado o fim do arquivo, é atribuido à variável EOF o valor '1'B, a variável CLASSE retorna com valor zero e o conteúdo da variável CAR permanece inalterado.

Qualquer caráter inexistente na linguagem é ignorado, não sendo considerado separador, sendo porém impressa uma mensagem de erro.

A rotina GETCAR le inclusive o primeiro registro do arquivo SCARDS. Na primeira chamada a esta rotina, a variável

PTR, que aponta para o próximo caráter a ser lido, deve ter sido inicializada com o valor 72, indicando que um novo registro deve ser lido (no caso o primeiro).

Os códigos dos caracteres são dados na tabela abaixo:

Caráter lido	CLASSE	Caráter lido	CLASSE
branco	1	-	21
letra	2	/	22
dígito	3	*	23
%	4	&	30
.	8	|	31
,	9	¬	32
;	10	>	36
(	11	<	37
)	12	=	38
:	13	"	45
[	14	'	46
]	15	fim de arquivo	0
+	20		

3.2.3.6. ROTINAS PARA MANIPULAÇÃO DA TABELA DE SÍMBOLOS - PLAPP05, PLAPP06 e PLAPP07

Esta rotina tem 3 "entry-points": para busca, inserção e retirada de símbolos da tabela.

A rotina de retirada não precisava ser um "entry-point", podendo ser uma rotina à parte. Está aqui apenas para grupar as rotinas de manipulação da tabela de símbolos.

A rotina de retirada recebe em BLOCO o número do bloco cuja análise está sendo encerrada, indicando que os símbolos definidos neste bloco devem ser retirados da tabela.

As rotinas de busca e inserção recebem em BLOCO o número do bloco corrente e em IDNOME o identificador.

3.2.3.6.1. ROTINA DE BUSCA - PLAPP05

Esta rotina efetua a busca, na tabela de símbolos, do identificador recebido em IDNOME. Para isso é aplicada ao identificador a função "hash", para se determinar o elemento do vetor cabeça de "hash" (CABEC) que aponta para a lista que deverá conter o identificador. Esta lista é então percorrida até ser encontrado um identificador igual ao procurado ou até o fim da lista.

É devolvido na variável ACHOU o valor '1'B ou '0'B conforme o identificador tenha sido encontrado ou não. No primeiro caso é devolvido na variável IND a posição do identificador na tabela. No segundo caso é emitida uma mensagem de erro, o identificador é inserido na tabela sendo devolvido em IND sua posição. A inserção é feita para evitar nova mensagem de erro cada vez que for feita referência ao identificador no programa fonte.

3.2.3.6.2. ROTINA DE INSERÇÃO - PLAPP06

À medida que os identificadores são declarados em cada bloco do programa fonte, esta rotina é chamada para inseri-los na tabela de símbolos com parte das informações necessárias para identificá-los.

Inicialmente é feita uma busca na tabela para verificar se o identificador já foi declarado antes, ou seja, se já se encontra na tabela. Caso seja encontrado, é verificado se foi definido no bloco atual ou num bloco externo. Se foi no bloco atual, é emitida uma mensagem de erro. Se não, é feita a inserção (o ponteiro AVAIL indica o índice do elemento de TABELA que conterá o identificador), sendo preenchidos os campos NOME e NIVEL e atualizados os ponteiros.

No caso de inserção bem sucedida a variável ACHOU retorna com o valor '0'B e a variável IND com a posição em que o identificador foi inserido na tabela. Caso contrário ACHOU retorna com o valor '1'B.

3.2.3.6.3. ROTINA PARA RETIRADA - PLAPP07

Cada vez que é encerrada a análise de um bloco esta rotina é chamada para retirar da tabela de símbolos os símbolos definidos neste bloco.

Para isso todas as listas não vazias são percorridas até ser encontrada em cada lista uma entrada cujo valor do campo NIVEL seja menor que o número do bloco sendo encerrado ou o fim da lista.

Todas as entradas encontradas durante o percurso desta lista são retiradas da tabela. A retirada é feita simplesmente pela atualização dos apontadores, inclusive do apontador AVAIL (para indicar que as entradas retiradas estão agora disponíveis para futuras inserções).

3.2.3.7. ROTINAS PARA MANIPULAÇÃO DAS TABELAS DE CONSTANTES - PLAPP13, PLAPP14 e PLAPP15

Esta rotina tem 3 "entry-points": para inserção na tabela de constantes inteiras, na tabela de constantes reais e na tabela de constantes alfanuméricas.

Estas rotinas poderiam ser independentes umas das outras. Estão grupadas para efeito de documentação.

3.2.3.7.1. ROTINA PARA INSERÇÃO NA TABELA DE CONSTANTES INTEIRAS PLAPP13

Esta rotina efetua a inserção de constantes inteiras na tabela de constantes inteiras (TABINT).

Recebe em CTEINT o valor da constante a inserir. É inicialmente procurada na tabela, de forma sequencial, uma constante igual ou maior do que a que deve ser inserida. Se o fim da lista foi alcançado sem ser encontrado tal valor ou se foi encontrada uma constante maior que CTEINT, é feita a inserção e os ponteiros são atualizados para manter a ordenação.

É devolvida na variável POSCTE a posição da constante na tabela.

Se foi encontrada uma constante igual a CTEINT não é feita a inserção e POSCTE contém, após a busca, a posição da

constante na tabela.

3.2.3.7.2. ROTINA PARA INSERÇÃO NA TABELA DE CONSTANTES REAIS - PLAPP14

Esta rotina efetua a inserção de constantes reais na tabela de constantes reais (TABREAL).

Recebe o valor da constante a ser inserida na variável CTEREAL. É inicialmente procurada na tabela, de forma sequêncial, uma constante igual ou maior que a que deve ser inserida. Se o fim da lista foi alcançado sem ser encontrado tal valor ou se foi encontrada uma constante maior que CTEREAL, é feita a inserção e os ponteiros são atualizados para manter a ordenação.

É devolvida na variável POSCTE a posição da constante na tabela.

Se foi encontrada uma constante igual a CTEREAL, não é feita a inserção e POSCTE contém, após a busca, a posição da constante na tabela.

3.2.3.7.3. ROTINA PARA INSERÇÃO NA TABELA DE CONSTANTES ALFANUMÉRICAS - PLAPP15

Esta rotina efetua a inserção de constantes alfanuméricas na tabela de constantes alfanuméricas (TABALFA).

Recebe o tamanho, em caracteres, do literal a ser inserido na variável TAMLIT e o literal no vetor CTELIT.

O literal é colocado no vetor TEXTO, são atualizados os ponteiros e o campo TAMALFA é preenchido com o tamanho do literal.

É devolvida na variável POSCTE a posição inicial do literal dentro do vetor TEXTO.

3.2.3.8. ROTINA PARA GERAR O PROGRAMA OBJETO EM DISCO - PLAPP31

A função desta rotina é gravar no arquivo OBJ2 as informações necessárias e suficientes para a carga e execução (pelo módulo interpretador - PLAPP32) do programa compilado, incluindo o programa em formato de quádruplas.

Estas informações são:

- o tamanho de cada área para dados a ser alocada para a execução do programa (áreas para constantes, variáveis e temporários - inteiros, reais, booleanos e alfanuméricos) e os valores das constantes. Estas informações são obtidas das tabelas de constantes;
- o tamanho da área para instruções. Obtido da variável CINST da estrutura AUXGERA, definida na macro PLAMV11;
- o tamanho da pilha de registradores base. Obtido da variável BLOCMAX, definida na macro PLAMV02;
- as quádruplas com as instruções. Obtidas do arquivo OBJ1.

Os arquivos OBJ1 e OBJ2 estão descritos no tópico 'Estrutura de Dados'.

3.2.3.9. ROTINA PARA GRAVAR O REGISTRO DE INSTRUÇÕES - PLAPP34

Esta rotina recebe como parâmetros as partes componentes de uma instrução: o código de operação, os tres operandos e os números dos registradores base relativos aos tres operandos.

Sua função é formatar esta instrução dentro do registro do arquivo OBJ1, efetuando a gravação quando o registro estiver completo.

Se for efetuada a gravação, o contador de instruções (CINST) é atualizado.

Em qualquer caso é atualizado o apontador para a próxima quádrupla do registro corrente (LC1).

Como o teste para gravação é efetuado antes da inserção da nova quádrupla, ao final da execução desta rotina o registro corrente conterá sempre pelo menos uma quádrupla.

3.2.4. ROTINAS PARA A FASE DE EXECUÇÃO

3.2.4.1. ROTINA PARA EXECUTAR O PROGRAMA COMPILADO - PLAPP32

Esta rotina simula a máquina virtual para a qual é gerado o código pelo Compilador Didático Pascal.

É chamada pelo Módulo Principal (PLAPP00) apesar

de ser logicamente independente do compilador pois todas as informações necessárias para a execução do programa compilado estão contidas no arquivo OBJ2.

Esta rotina é composta de duas partes. Uma parte inicial de carga, em que é lido o arquivo OBJ2, sendo efetuada a alocação das áreas para constantes, variáveis, temporários e instruções e processada a carga das constantes e instruções.

A segunda parte é o interpretador propriamente dito, que processa a decodificação das quádruplas efetuando a execução das instruções nelas contidas.

3.2.4.2. ROTINA PARA IMPRESSÃO DE MENSAGENS DURANTE A EXECUÇÃO - PLAPP33

A função desta rotina é imprimir as mensagens emitidas durante a execução, pelo módulo interpretador (PLAPP32), do programa compilado.

Recebe como parâmetro o código da mensagem a ser impressa.

As mensagens tem tamanho variável, sendo compostas por blocos de 53 caracteres, armazenados em uma tabela dupla, conforme descrito no tópico 'Estrutura de Dados'.

A tabela é percorrida a partir da entrada correspondente ao código da mensagem, seguindo pelos apontadores de continuação até o final da mensagem. Em cada linha são impressos dois blocos do texto da mensagem.

3.2.5. MACROS

Para declaração das rotinas, variáveis, tabelas e estruturas comuns foram definidas macro-instruções para o pré-processador da linguagem PL1. Cada rotina que necessita de uma dessas declarações efetua a inclusão da macro correspondente, por meio do comando INCLUDE do pré-processador[1].

---

1) Nesta implementação, ao invés do comando INCLUDE do pré-processador PL1 para efetuar a inclusão das macros no texto dos programas fonte do compilador foi utilizado o recurso $CONTINUE WITH do sistema operacional MTS (Michigan Terminal System), por questões de eficiência, sendo porém mantida a mesma estrutura.

As macros de nome PLAMAxx, onde xx é um número de dois algarismos, definem as rotinas em assembler; as de nome PLAMPxx definem as rotinas em PL1 e as de nome PLAMVxx definem as variáveis, tabelas e estruturas. Nos nomes das macros que definem as rotinas o número xx é o mesmo número do nome da rotina.

Como em cada macro sua parte inicial compreende um comentário com sua descrição detalhada e a listagem das mesmas encontra-se em anexo, as descrições das macros foi omitida aqui.

4. CONCLUSÕES E SUGESTÕES PARA UTILIZAÇÃO

Os seguintes aspectos merecem destaque por sua influência no desenvolvimento e na utilização deste trabalho. Note-se que a ordem em que estão apresentados não implica em relação de importância:

- o compilador implementado neste trabalho destina-se a ser utilizado como ferramenta no aprendizado de construção de compiladores e não para compilação de programas;

- a simplificação efetuada na linguagem original não modificou sua estrutura e o subconjunto resultante manteve as características que justificaram a escolha da linguagem PASCAL;

- a grande difusão da linguagem PL1, facilitando o acesso e a compreensão compensa a perda de eficiência por não se tratar de uma linguagem específica para implementação de compiladores;

- a implementação do compilador em um único passo reforça a necessidade, por parte do usuário deste trabalho, da compreensão do conjunto sem entretanto obrigá-lo ao conhecimento de detalhes, senão das partes diretamente envolvidas na alteração desejada;

- a substituição da geração de código para uma máquina real pela definição de uma máquina virtual com características bem próximas das máquinas reais e pela implementação do simulador respectivo confere portabilidade ao Compilador Didático Pascal além de tornar mais fácil a compreensão do código gerado, devido à simplicidade da máquina virtual definida;

Abaixo estão relacionadas algumas sugestões para o uso deste trabalho em exercícios de um curso de construção de compiladores:

- modificação do método de geração de código;
- criação de um passo complementar para otimização do código gerado para a máquina virtual;
- geração de código para uma máquina real;
- substituição de um ou mais reconhecedores sintáticos para avaliação e/ou desenvolvimento de outros métodos de análise;
- estudos de métodos mais completos de recuperação e correção de erros;
- modificações na linguagem fonte, como por exemplo, a reinclusão das partes retiradas da linguagem original.

BIBLIOGRAFIA

1. WIRTH, N. - JENSEN, K. - Pascal - User Manual and Report - Berlin, Springer Verlag, 1974.

2. WIRTH, N. - Algorithms + Data Structure = Program - New Jersey, Prentice-Hall, 1976.

3. HALSTEAD, M. - A Laboratory Manual for Compiler and Operating System Implementation - New York, American Elsevier Publishing Company, Inc, 1974.

4. GRIES, D. - Compiler Construction for Digital Computers - U.S.A., John Wiley & Sons, Inc, 1971.

5. KNUTH, D. - The Art of Computer Programming - Vol. 1 - Fundamentals Algorithms - U.S.A., Addison Wesley, 1973.

6. FOSTER, J.M. - Automatic Syntatic Analysis - London, MacDonald and American Elsevier, 1970.

7. HOPGOOD, F.R.A. - Compiling Techniques - London, MacDonald and American Elsevier, 1971.

8. PRATT, T. - Programming Languages: Design and Implementation - New Jersey, Prentice-Hall, Inc, 1975.

9. HOPCROFT, J.E. - ULLMAN, J.D. - Formal Languages and their Relation to Automata - U.S.A., Addison Wesley Publishing Company, Inc, 1969.

10. MICHIGAN UNIVERSITY - Pascal in MTS.

11. MICHIGAN UNIVERSITY - PL/1 in MTS.

12. MICHIGAN UNIVERSITY - The Michigan Terminal System.

A P Ê N D I C E S

A - BNF da Linguagem Original

```
<program> ::= <program heading> <block> .

<program heading> ::= PROGRAM <identifier> ( <file identifier>
                      { , <file identifier> } ) ;

<file identifier> ::= <identifier>

<identifier> ::= <letter> { <letter or digit> }

<letter or digit> ::= <letter> | <digit>

<block> ::= <label declaration part> <constant definition part>
            <type definition part> <variable declaration part>
            <procedure and function declaration part>
            <statement part>

<label declaration part> ::= <empty> |
                             LABEL <label> { , <label> } ;

<label> ::= <unsigned integer>

<constant definition part> ::= <empty> |
       CONST <constant definition> { ; <constant definition> } ;

<constant definition> ::= <identifier> = <constant>

<constant> ::= <unsigned number> | <sign> <unsigned number> |
          <constant identifier> | <sign> <constant identifier> |
          <string>

<unsigned number> ::= <unsigned integer> | <unsigned real>

<unsigned integer> ::= <digit> { <digit> }

<unsigned real> ::= <unsigned integer> . <digit> { <digit> } |
     <unsigned integer> . <digit> { <digit> } E <scale factor> |
     <unsigned integer> E <scale factor>

<scale factor> ::= <unsigned integer> | <sign> <unsigned integer>

<sign> ::= + | -

<constant identifier> ::= <identifier>
```

```
<string> ::= ' <character> { <character> } '

<type definition part> ::= <empty> |
                TYPE <type definition> { ; <type definition> } ;

<type definition> ::= <identifier> = <type>

<type> ::= <simple type> | <structured type> | <pointer type>

<simple type> ::= <scalar type> | <subrange type> |
                  <type identifier>

<scalar type> ::= ( <identifier> { , <identifier> } )

<subrange type> ::= <constant> .. <constant>

<type identifier> ::= <identifier>

<structured type> ::= <unpacked structured type> |
                      PACKED <unpacked structured type>

<unpacked structured type> ::= <array type> | <record type> |
                               <set type> | <file type>

<array type> ::= ARRAY [ <index type> { , <index type> } ] OF
                 <component type>

<index type> ::= <simple type>

<component type> ::= <type>

<record type> ::= RECORD <field list> END

<field list> ::= <fixed part> | <fixed part> ; <variant part> |
                 <variant part>

<fixed part> ::= <record section> { ; <record section> }

<record section> ::= <field identifier> { , <field identifier> }
                     : <type> | <empty>

<variant part> ::= CASE <tag field> <type identifier> OF
                   <variant> { ; <variant> }
```

```
<tag field> ::= <field identifier> : | <empty>

<variant> ::= <case label list> : ( <field list> ) | <empty>

<case label list> ::= <case label> { , <case label> }

<case label> ::= <constant>

<set type> ::= SET OF <base type>

<base type> ::= <simple type>

<file type> ::= FILE OF <type>

<pointer type> ::= ↑ <type identifier>

<variable declaration part> ::= <empty> |
      VAR <variable declaration> { ; <variable declaration> } ;

<variable declaration> ::= <identifier> { , <identifier> } :
                           <type>

<procedure and function declaration part> ::=
                      { <procedure or function declaration> ; }

<procedure or function declaration> ::= <procedure declaration> |
                                        <function declaration>

<procedure declaration> ::= <procedure heading> <block>

<procedure heading> ::= PROCEDURE <identifier> ; |
            PROCEDURE <identifier> ( <formal parameter section>
            { ; <formal parameter section> } ) ;

<formal parameter section> ::= <parameter group> |
           VAR <parameter group> | FUNCTION <parameter group> |
           PROCEDURE <identifier> { , <identifier> }

<parameter group> ::= <identifier> { , <identifier> } :
                      <type identifier>

<function declaration> ::= <function heading> <block>
```

```
<function heading> ::= FUNCTION <identifier> : <result type> ; |
            FUNCTION <identifier> ( <formal parameter section>
            { ; <formal parameter section> } ) : <result type> ;

<result type> ::= <type identifier>

<statement part> ::= <compound statement>

<statement> ::= <unlabelled statement> |
                <label> : <unlabelled statement>

<unlabelled statement> ::= <simple statement> |
                           <structured statement>

<simple statement> ::= <assignment statement> |
                       <procedure statement> |
                       <go to statement> | <empty statement>

<assignment statement> ::= <variable> := <expression> |
                           <function identifier> := <expression>

<variable> ::= <entire variable> | <component variable> |
               <referenced variable>

<entire variable> ::= <variable identifier>

<variable identifier> ::= <identifier>

<component variable> ::= <indexed variable> |
                         <field designator> | <file buffer>

<indexed variable> ::= <array variable> [ <expression>
                       { , <expression> } ]

<array variable> ::= <variable>

<field designator> ::= <record variable> . <field identifier>

<record variable> ::= <variable>

<field identifier> ::= <identifier>

<file buffer> ::= <file variable> ↑
```

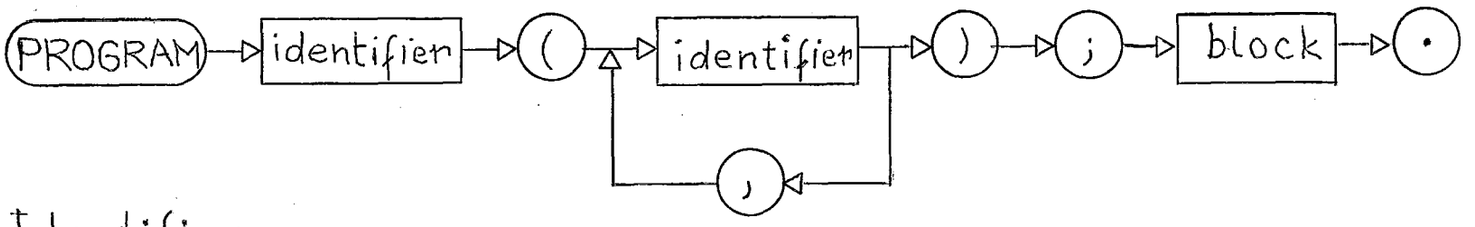
```
<file variable> ::= <variable>

<referenced variable> ::= <pointer variable> ↑

<pointer variable> ::= <variable>

<expression> ::= <simple expression> | <simple expression>
                 <relational operator> <simple expression>

<relational operator> ::= = | <> | < | <= | >= | > | IN

<simple expression> ::= <term> | <sign> <term> |
                        <simple expression> <adding operator> <term>

<adding operator> ::= + | - | OR

<term> ::= <factor> | <term> <multiplying operator> <factor>

<multiplying operator> ::= * | / | DIV | MOD | AND

<factor> ::= <variable> | <unsigned constant> |
             ( <expression> ) | <function designator> | <set> |
             NOT <factor>

<unsigned constant> ::= <unsigned number> | <string> |
                        <constant identifier> | NIL

<function designator> ::= <function identifier> |
                          <function identifier> ( <actual parameter>
                          { , <actual parameter> } )

<function identifier> ::= <identifier>

<set> ::= [  <element list>  ]

<element list> ::= <element> { , <element> } | <empty>

<element> ::= <expression> | <expression> .. <expression>

<procedure statement> ::= <procedure identifier> |
                          <procedure identifier> ( <actual parameter>
                          { , <actual parameter> } )

<procedure identifier> ::= <identifier>
```

```
<actual parameter> ::= <expression> | <variable> |
                       <procedure identifier> | <function identifier>

<go to statement> ::= GOTO <label>

<empty statement> ::= <empty>

<empty> ::=

<structured statement> ::= <compound statement> |
                  <conditional statement> | <repetitive statement> |
                  <with statement>

<compound statement> ::= BEGIN <statement> { ; <statement> } END

<conditional statement> ::= <if statement> | <case statement>

<if statement> ::= IF <expression> THEN <statement> |
                   IF <expression> THEN <statement> ELSE <statement>

<case statement> ::= CASE <expression> OF <case list element>
                        { ; <case list element> } END

<case list element> ::= <case label list> : <statement> | <empty>

<case label list> ::= <case label> { , <case label> }

<repetitive statement> ::= <while statement> |
                               <repeat statement> | <for statement>

<while statement> ::= WHILE <expression> DO <statement>

<repeat statement> ::= REPEAT <statement> { ; <statement> }
                          UNTIL <expression>

<for statement> ::= FOR <control variable> := <for list> DO
                        <statement>

<for list> ::= <initial value> TO <final value> |
                  <initial value> DOWNTO <final value>

<control variable> ::= <identifier>

<initial value> ::= <expression>
```

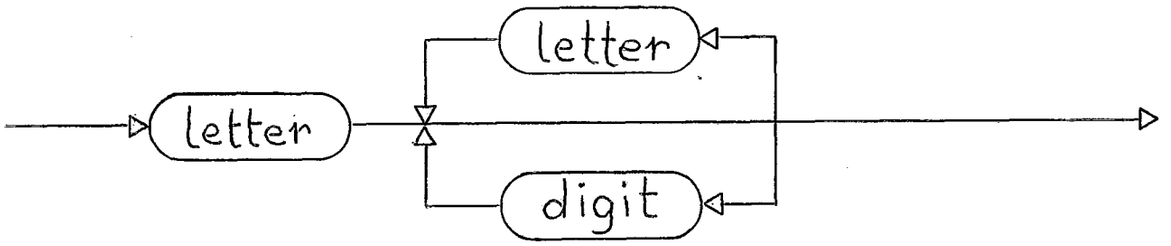
```
<final value> ::= <expression>

<with statement> ::= WITH <record variable list> DO <statement>

<record variable list> ::= <record variable>
                           { , <record variable> }
```

B - Diagramas da Linguagem Original

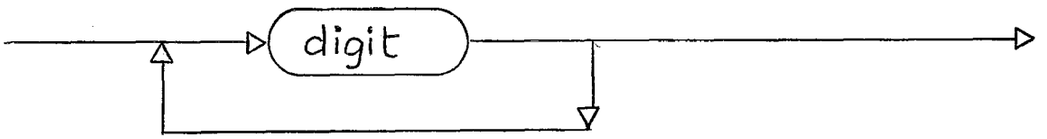
Program



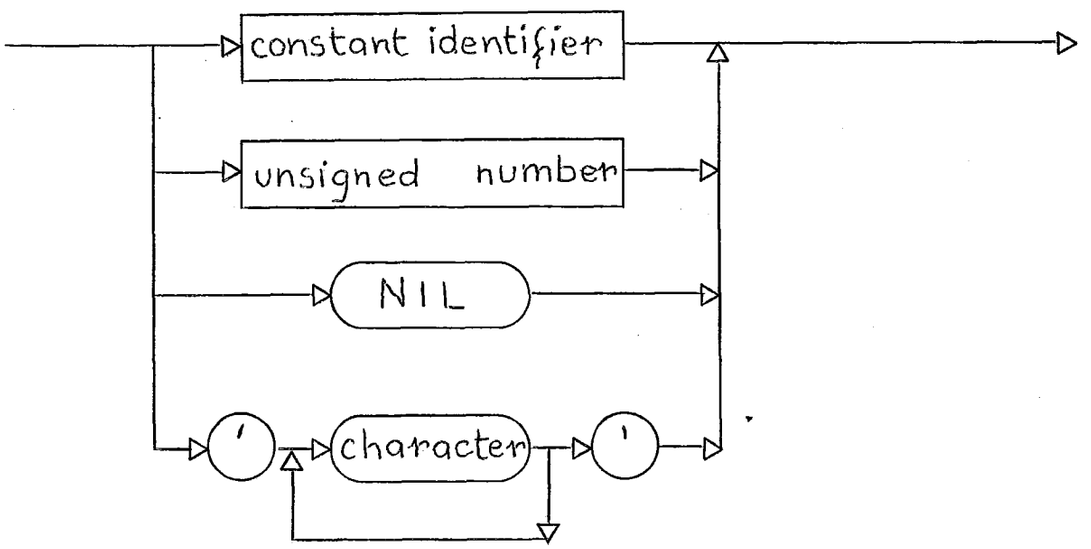
Identifier



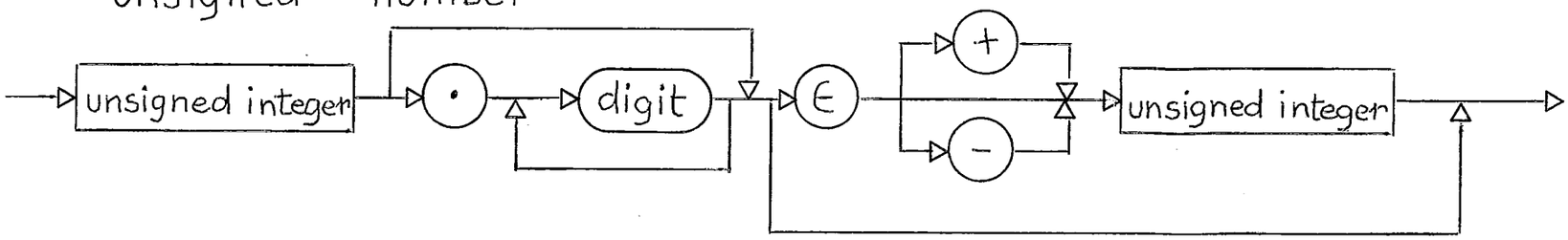
Unsigned integer



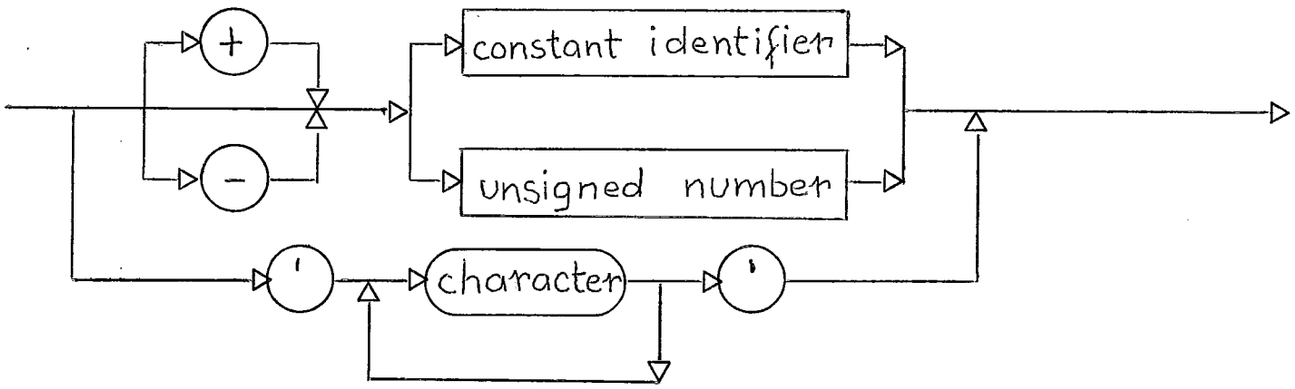
Unsigned constant



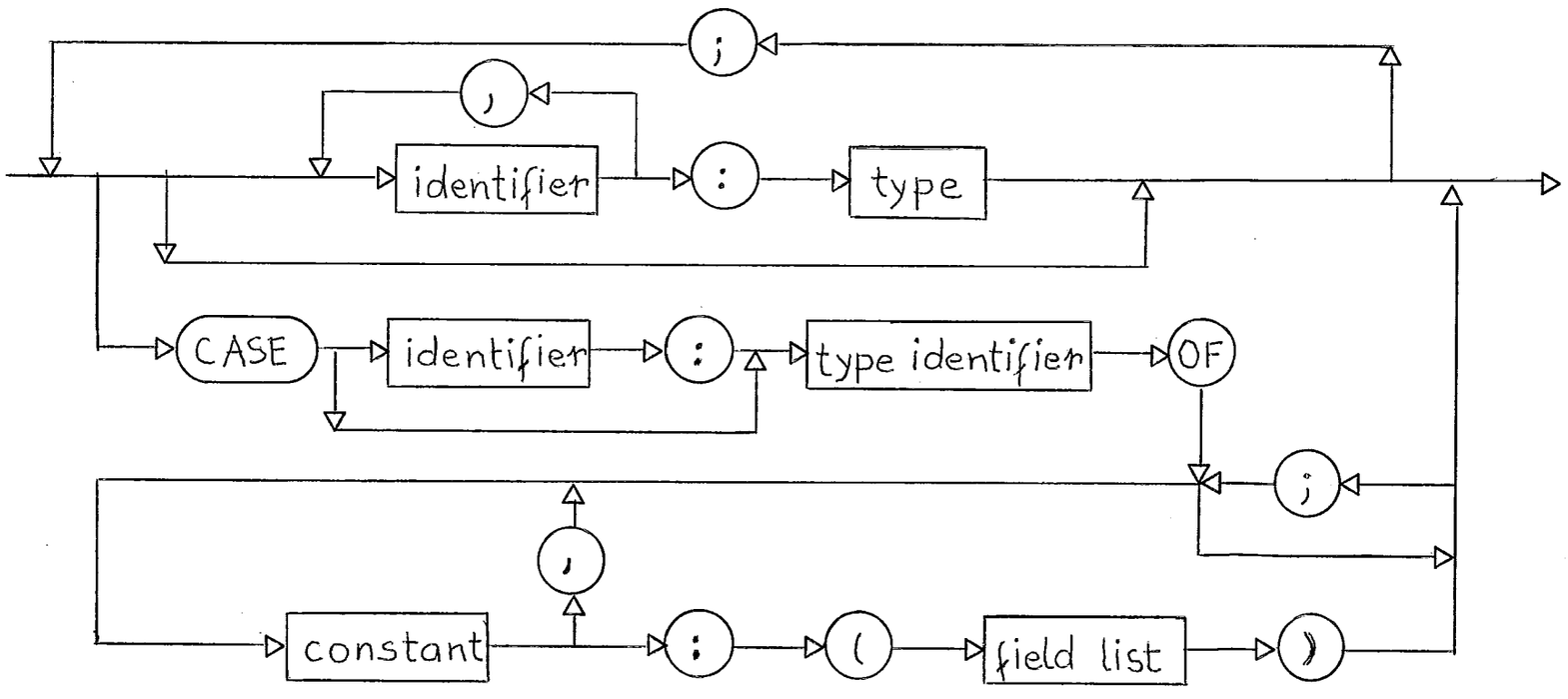
Unsigned number



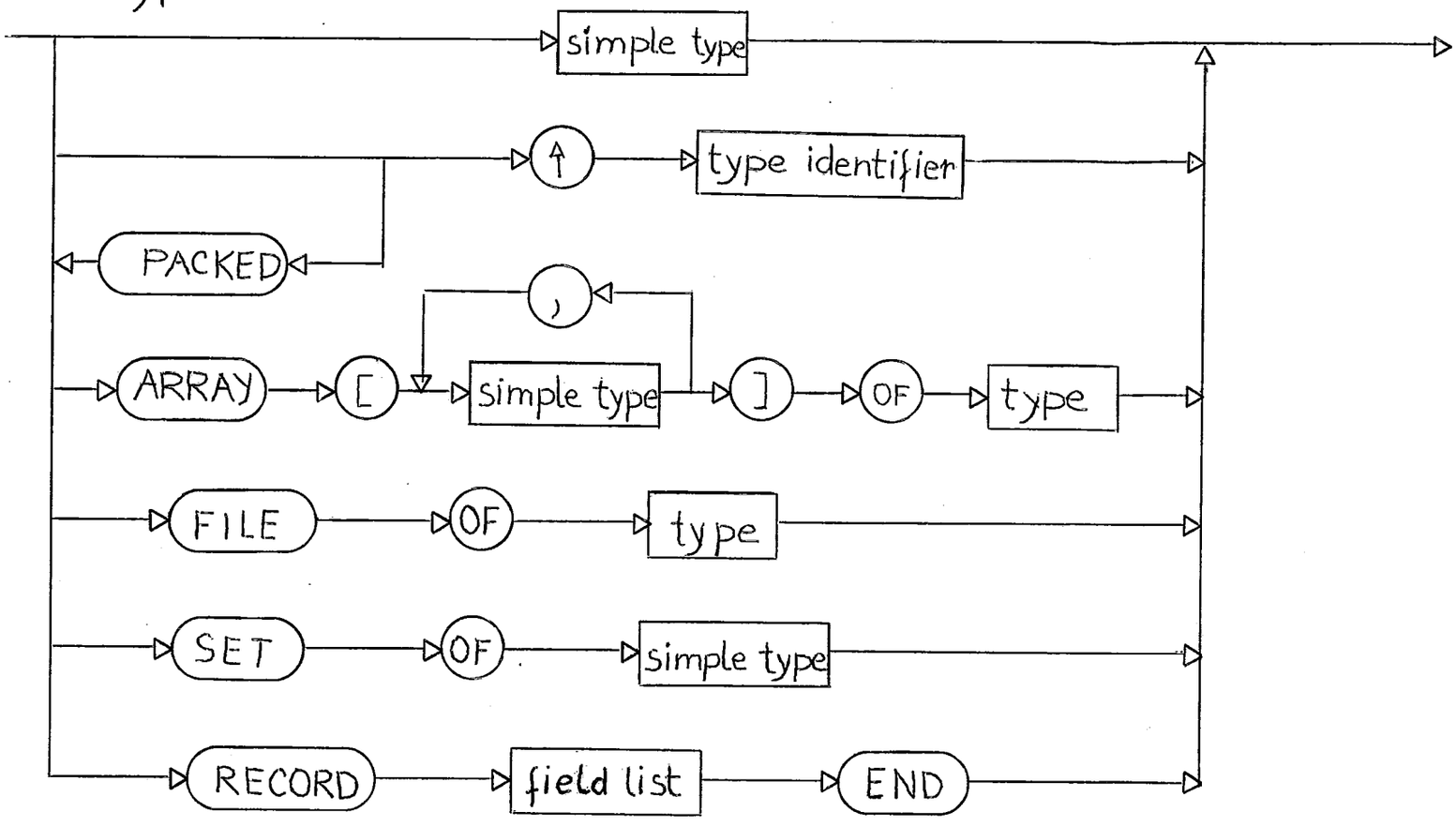
Constant



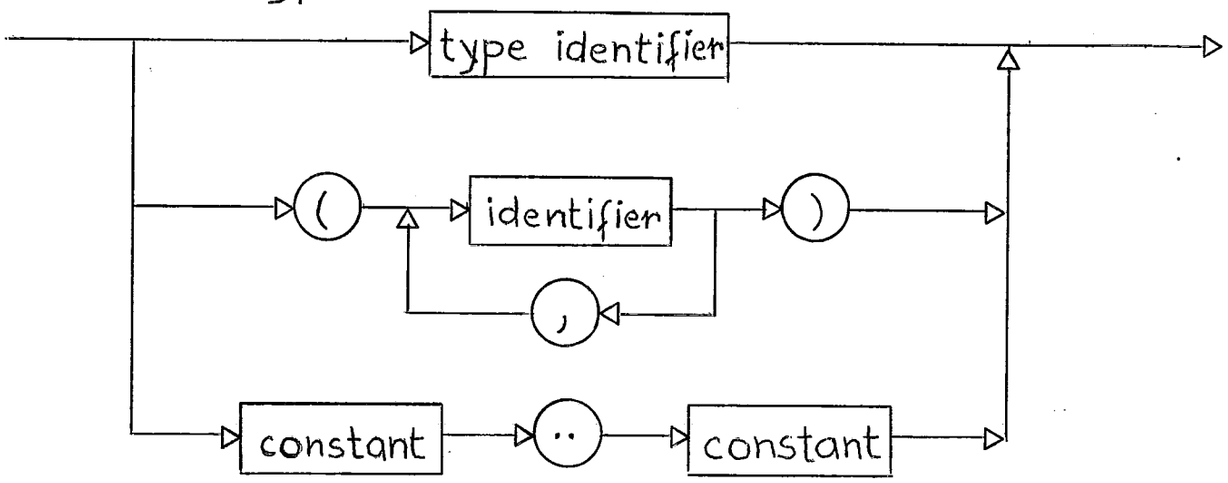
Field list



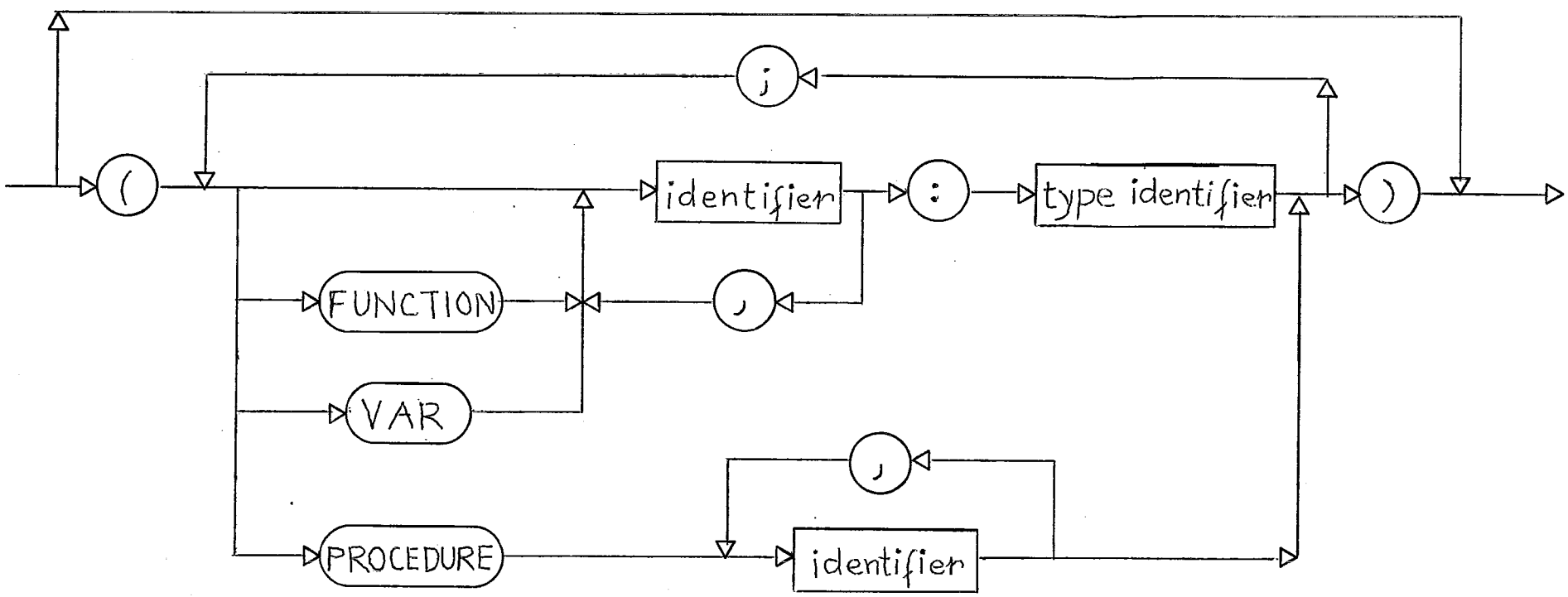
Type



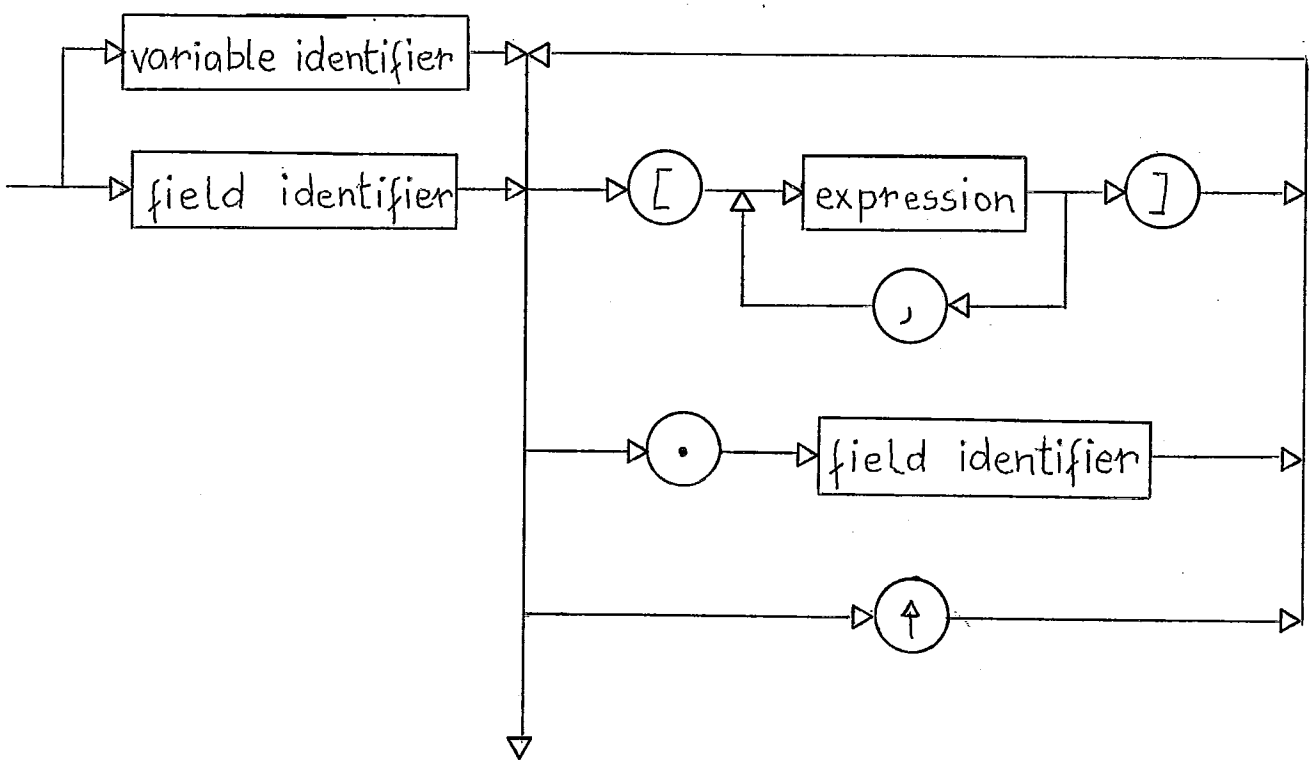
Simple type



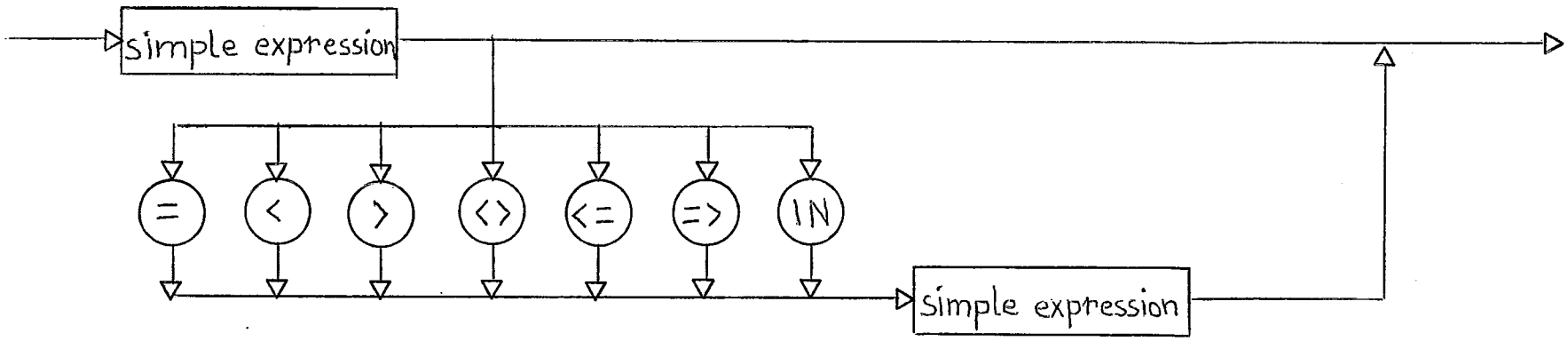
Parameter List



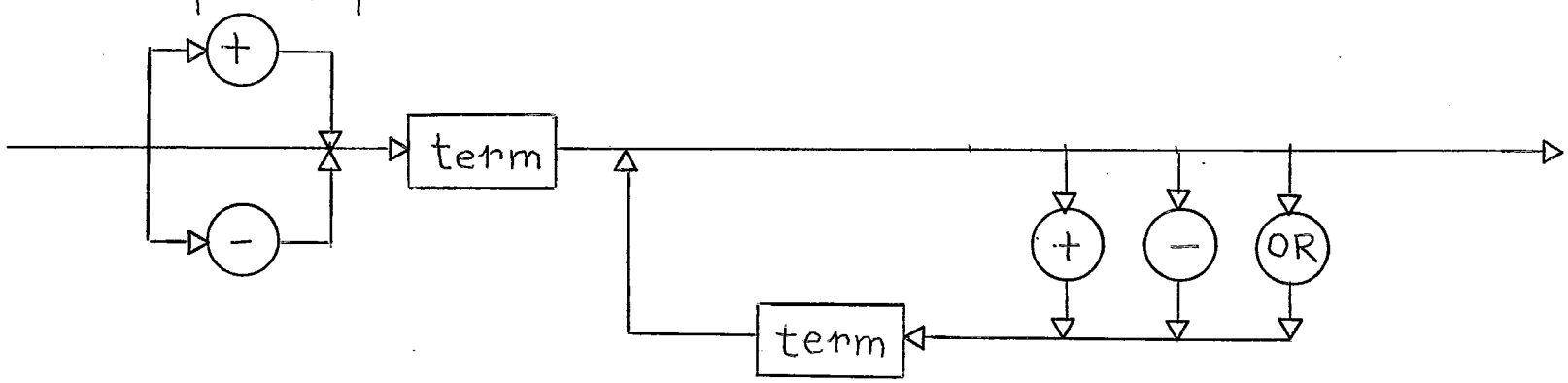
Variable



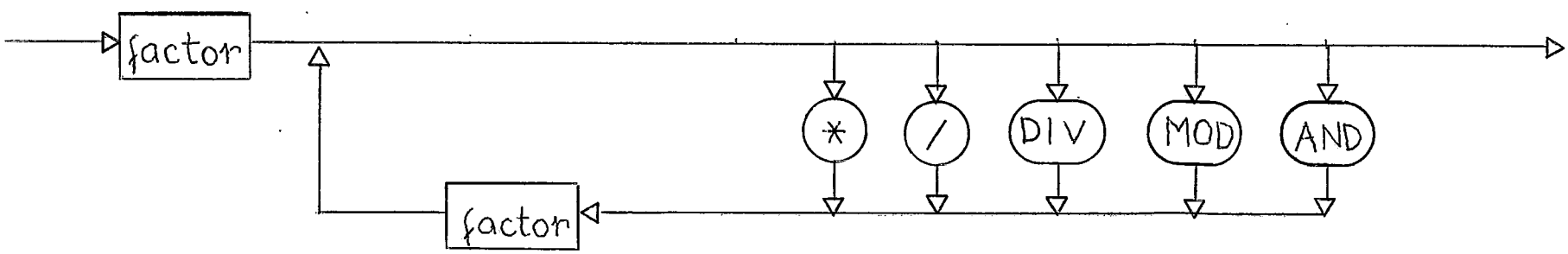
Expression



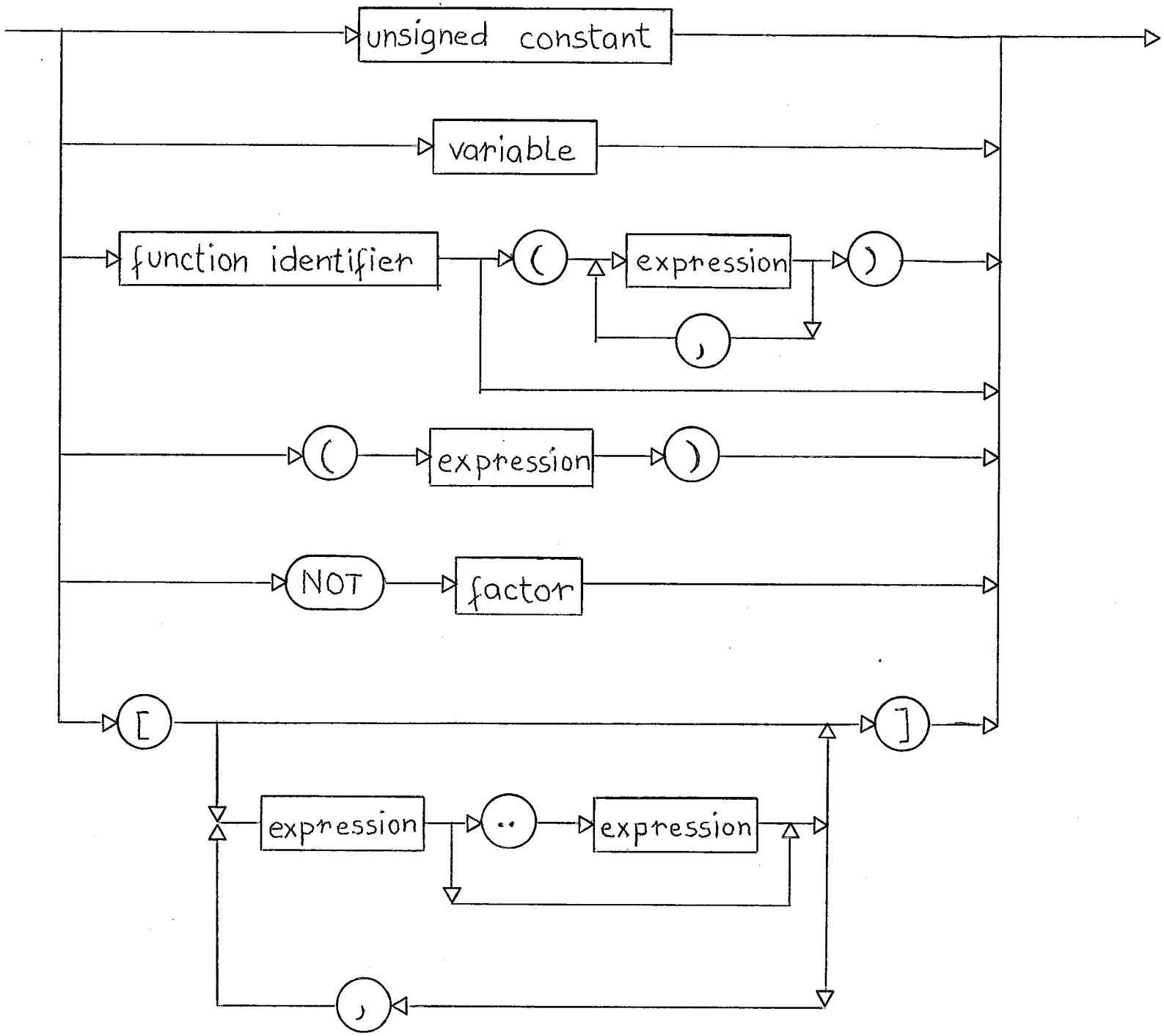
Simple expression



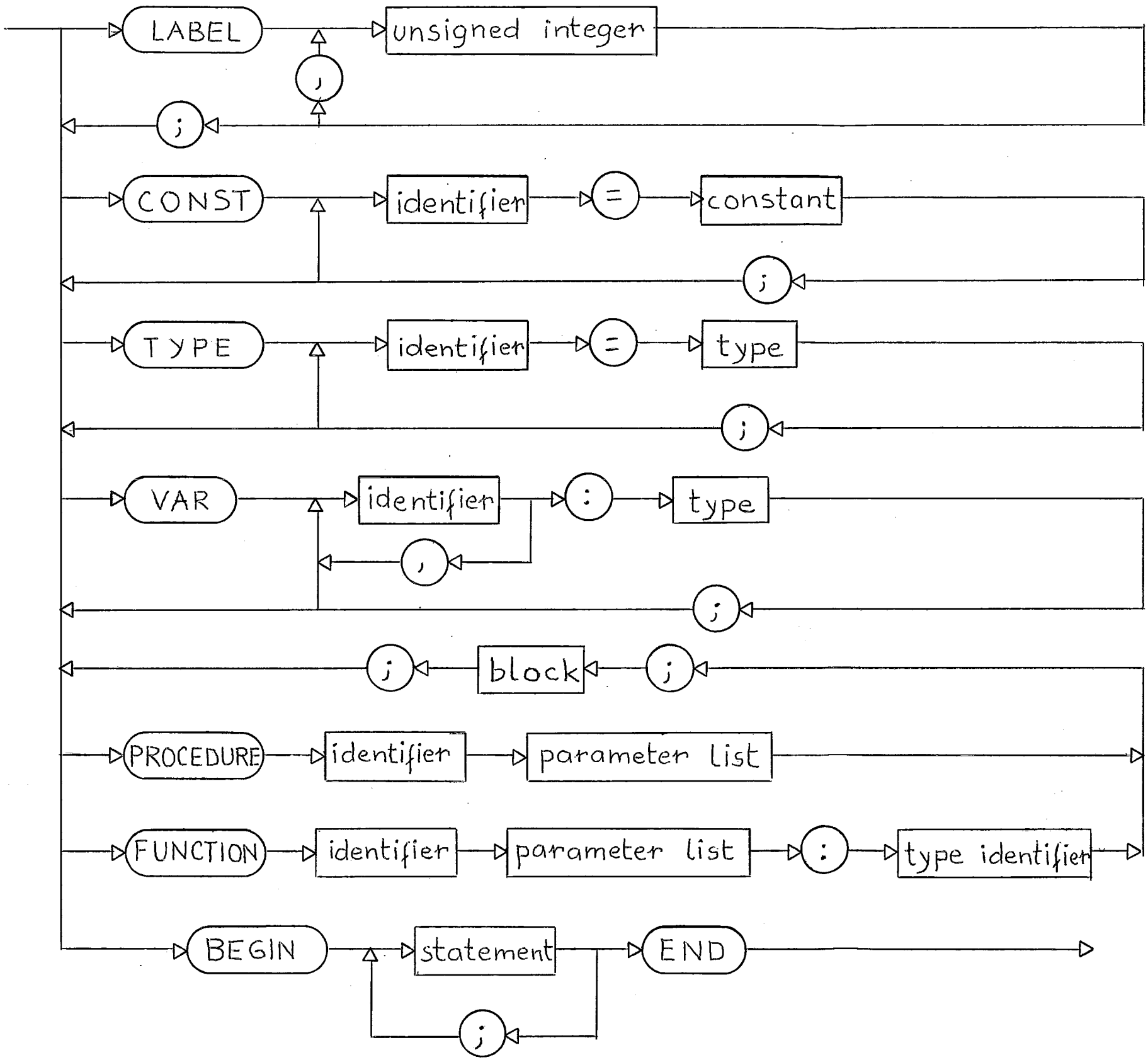
Term



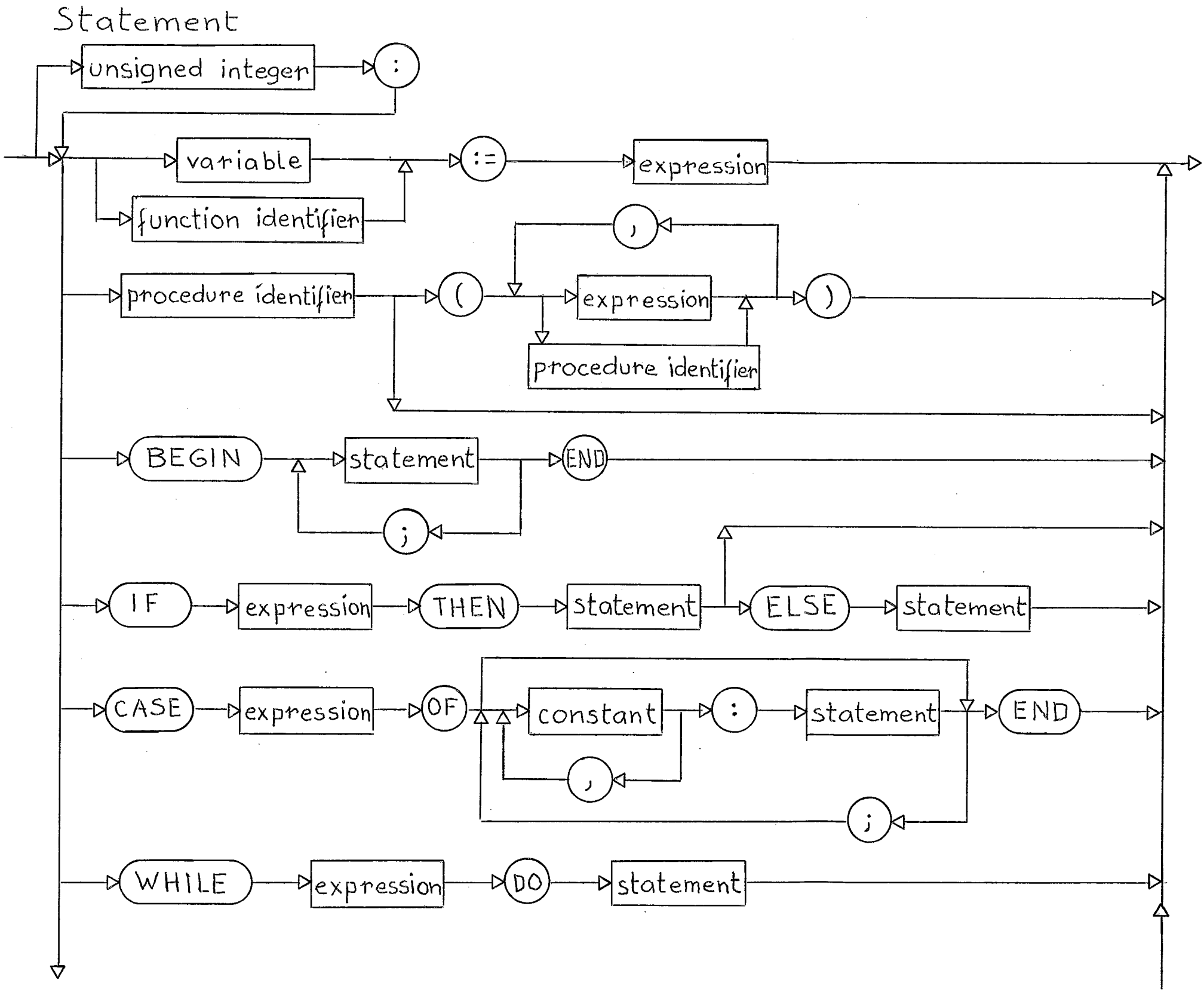
Factor


unsigned constant
variable
function identifier
(
expression
)
,
(
expression
)
NOT
factor
[
]
expression
..
expression
,

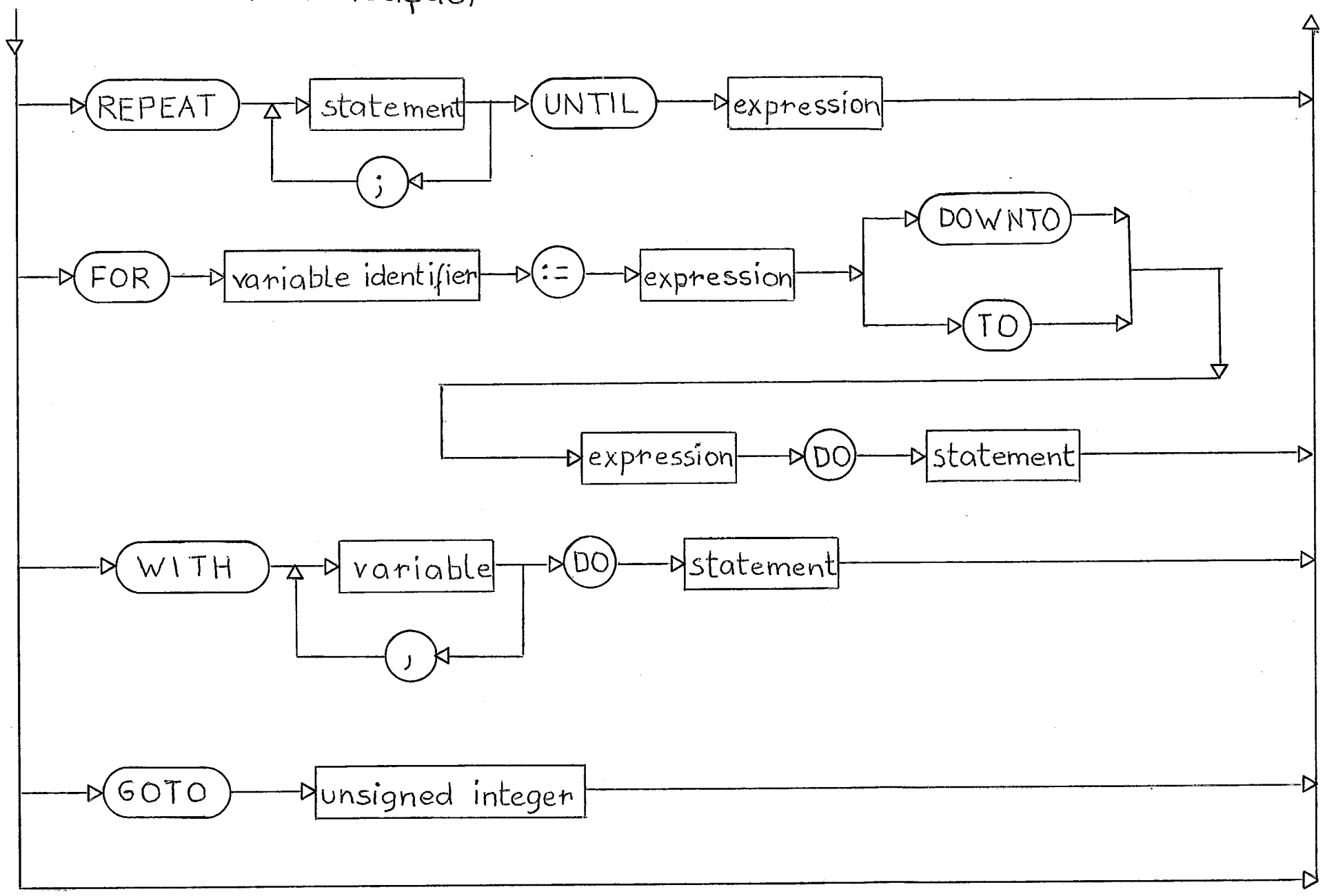
Block


LABEL
unsigned integer
,
;
CONST
identifier
=
constant
;
TYPE
identifier
=
type
;
VAR
identifier
:
type
,
;
;
block
;
PROCEDURE
identifier
parameter list
FUNCTION
identifier
parameter list
:
type identifier
BEGIN
statement
END
;

Statement


unsigned integer
:
variable
:=
expression
function identifier
,
procedure identifier
(
expression
)
procedure identifier
BEGIN
statement
END
;
IF
expression
THEN
statement
ELSE
statement
CASE
expression
OF
constant
:
statement
END
,
;
WHILE
expression
DO
statement

Statement (continuação)



C - BNF do Subconjunto do PASCAL Utilizado

```
<programa> ::= <cabeçalho de programa> <bloco> .

<cabeçalho de programa> ::= PROGRAM <identificador> ;

<identificador> ::= <letra> { <letra ou dígito> }

<letra> ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
            N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<letra ou dígito> ::= <letra> | <dígito>

<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<bloco> ::= <declaração de rótulos > <declaração de constantes>
            <declaração de variáveis>
            <declaração de procedimentos e funções>
            <comando composto>

<declaração de rótulos > ::= <vazio> | LABEL <rótulo>
                             { , <rótulo> } ;

<vazio> ::=

<rótulo> ::= <identificador>

<declaração de constantes> ::= <vazio> |
                               CONST <definição de constante>
                               { , <definição de constante> } ;

<definição de constante> ::= <identificador>
                             { , <identificador> } = <constante>

<constante> ::= <número sem sinal> |
                <sinal> <número sem sinal> |
                <identificador de constante> |
                <sinal> <identificador de constante> |
                <literal>

<número sem sinal> ::= <inteiro sem sinal> | <real sem sinal>

<inteiro sem sinal> ::= <dígito> { <dígito> }
```

```
<real sem sinal> ::= <inteiro sem sinal> . <inteiro sem sinal> |
          <inteiro sem sinal> . <inteiro sem sinal> <expoente> |
          <inteiro sem sinal> <expoente>

<expoente> ::= E <inteiro sem sinal> |
               E <sinal> <inteiro sem sinal>

<sinal> ::= + | -

<identificador de constante> ::= <identificador>

<literal> ::= " <sequência sem aspas> " |
              ' <sequência sem apóstrofe> '

<sequência sem aspas> ::= ' | <caráter> |
                          ' <sequência sem aspas> |
                          <caráter> <sequência sem aspas>

<caráter> ::= <letra ou dígito> | <símbolo especial>

<símbolo especial> ::= . | , | ; | ( | ) | : | [ | ] | + | - |
                       / | * | & | OR[1] | < | > | = | ¬ | ∅ | %

<sequência sem apóstrofe> ::= " | <caráter> |
                              " <sequência sem apóstrofe> |
                              <caráter> <sequência sem apóstrofe>

<declaração de variáveis> ::= <vazio> |
                               VAR <definição de variáveis>
                               { , <definição de variáveis> } ;

<definição de variáveis> ::= <identificador>
                             { , <identificador> } : <tipo> |
                             <identificador>
                             { , <identificador> } :
                             <tipo de array>

<tipo> ::= INTEGER | REAL | BOOLEAN | CHAR
```

---

1) para evitar a ambiguidade, o símbolo | da linguagem foi substituido por OR na BNF, bem como o espaço que foi substituido por ∅.

```
<tipo de array> ::= ARRAY [ <dimensão> { , <dimensão> } ] OF
                    <tipo>

<dimensão> ::= <expressão tipo 1> : <expressão tipo 1> |
               <expressão tipo 1>

<expressão tipo 1> ::= <termo 1> | <sinal> <termo 1> |
                       <expressão tipo 1> + <termo 1> |
                       <expressão tipo 1> - <termo 1>

<termo 1> ::= <fator 1> | <termo 1> * <fator 1> |
              <termo 1> / <fator 1> | <termo 1> DIV <fator 1> |
              <termo 1> MOD <fator 1>

<fator 1> ::= <número sem sinal> | ( <expressão tipo 1> ) |
              <identificador de constante>

<declaração de procedimentos e funções> ::=
                        { <definição de procedimento ou função> ; }

<definição de procedimento ou função< ::=
                 <definição de procedimento> | <definição de função>

<definição de procedimento> ::= <cabeçalho de procedure> <bloco>

<cabeçalho de procedimento> ::= PROC <identificador> ; |
                                PROC <identificador>
                                ( <seção de parâmetros formais> ) ;

<seção de parâmetros formais> ::= <lista de parâmetros formais>
                                  { ; <lista de parâmetros formais> }

<lista de parâmetros formais> ::= <grupo de parâmetros> |
       FUNCTION <identificador> { , <identificador> } : <tipo> |
       PROC <identificador> { , <identificador> } |
       VAR <grupo de parâmetros>

<grupo de parâmetros> ::= <identificador> { , <identificador> }
                          : <tipo> |
                          <identificador> { , <identificador> }
                          : <tipo de array>
```

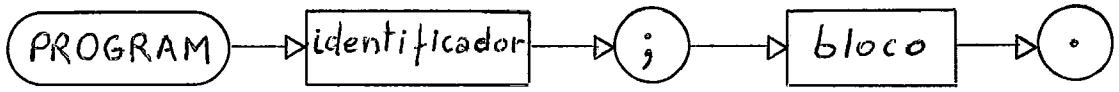
```
<definição de função> ::= <cabeçalho de função> <bloco>

<cabeçalho de função> ::= FUNCTION <identificador> : <tipo> ; |
                          FUNCTION <identificador>
                          ( <seção de parâmetros formais> ) :
                          <tipo> ;

<comando composto> ::= BEGIN <comando> { ; <comando> } END

<comando> ::= <comando sem rótulo>  |
              < rótulo> : <comando sem rótulo>

<comando sem rótulo>  ::= <comando simples> |
                          <comando estruturado> |
                          <comando de entrada/saida>

<comando simples> ::= <comando de atribuição> |
                      <comando de ativação de procedimento> |
                      <comando goto> | <vazio>

<comando de atribuição> ::= <variável> := <expressão> |
                            <identificador de função> := <expressão>

<variável> ::= <variável simples> | <variável indexada>

<variável simples> ::= <identificador de variável simples>

<identificador de variável simples> ::= <identificador>

<variável indexada> ::= <identificador de array>
                        [  <expressão> { , <expressão> }  ] |
                        <identificador de array>

<identificador de array> ::= <identificador>

<expressão> ::= <expressão simples> | <expressão simples>
                <operador relacional> <expressão simples>

<expressão simples> ::= <termo> | <sinal> <termo> |
                <expressão simples> <operador de adição> <termo>
```

```
<termo> ::= <termo> <operador de multiplicação> <fator> |
            <fator>

<fator> ::= <variável> | <constante sem sinal> |
            ( <expressão> ) | <designador de função> | ¬ <fator>

<constante sem sinal> ::= <número sem sinal> | <literal> |
                          <identificador de constante>

<designador de função> ::= <identificador de função> |
                           <identificador de função>
                           ( <parâmetro real>
                           { , <parâmetro real> } )

<identificador de função> ::= <identificador>

<parâmetro real> ::= <expressão> | <identificador de função> |
                     <identificador de procedimento>

<identificador de procedimento>::= <identificador>

<operador de multiplicação> ::= * | / | DIV | MOD | &

<operador de adição> ::= + | - | OR

<operador relacional> ::= = | ¬= | < | > | <= | >=

<comando de ativação de procedimento> ::=
                     <identificador de procedimento> |
                     <identificador de procedimento>
                     ( <parâmetro real> { , <parâmetro real> } )

<comando goto> ::= GOTO <rótulo>

<comando estruturado> ::= <comando condicionante> |
                          <comando iterativo>     |
                          <comando composto>

<comando condicionante> ::= <comando if> | <comando case>

<comando if> ::= IF <expressão> THEN <comando> |
                 IF <expressão> THEN <comando> ELSE <comando>
```

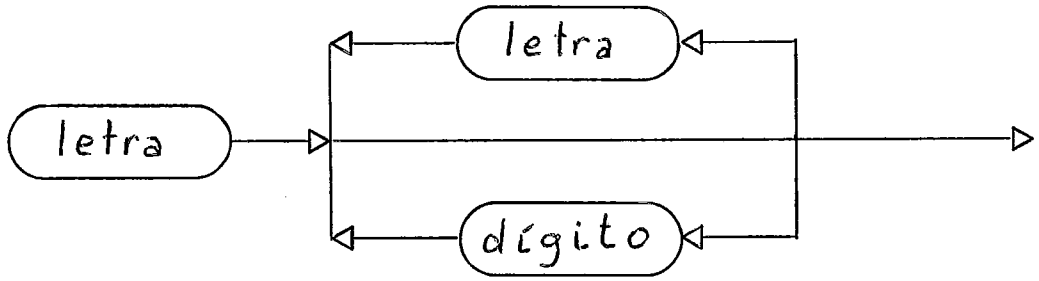
```
<comando case> ::= CASE <expressão> OF <elemento de case>
                   { ; <elemento de case> } END

<elemento de case> ::= <vazio> | <constante> { , <constante> } :
                       <comando>

<comando iterativo> ::= <comando while> | <comando for> |
                        <comando repeat>

<comando while> ::= WHILE <expressão> DO <comando>

<comando for> ::= FOR <variável simples> := <lista for> DO
                  <comando>

<lista for> ::= <expressão> TO <expressão> |
                <expressão> DOWNTO <expressão>

<comando repeat> ::= REPEAT <comando> { ; <comando> } UNTIL
                     <expressão>

<comando de entrada/saida> ::= <comando read> | <comando write>

<comando read> ::= <tipo read> ( <variável> { , <variável> } )

<tipo read> ::= READ | READD | READP | READPD

<comando write> ::= <tipo write>
                    ( <expressão> { , <expressão> } )

<tipo write> ::= WRITE | WRITED | WRITELN | WRITELND | WRITEPG |
                 WRITEPGD
```

D - Diagramas do Subconjunto do PASCAL Utilizado

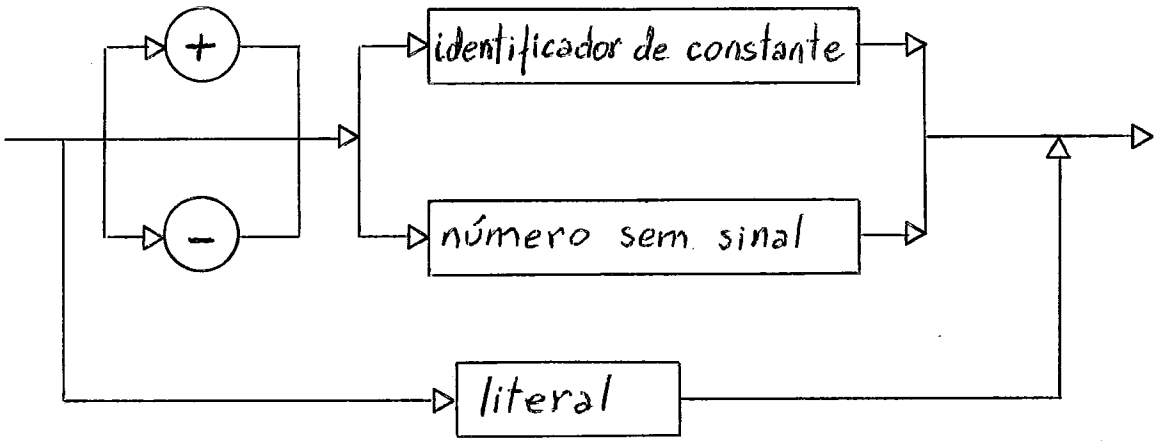
programa



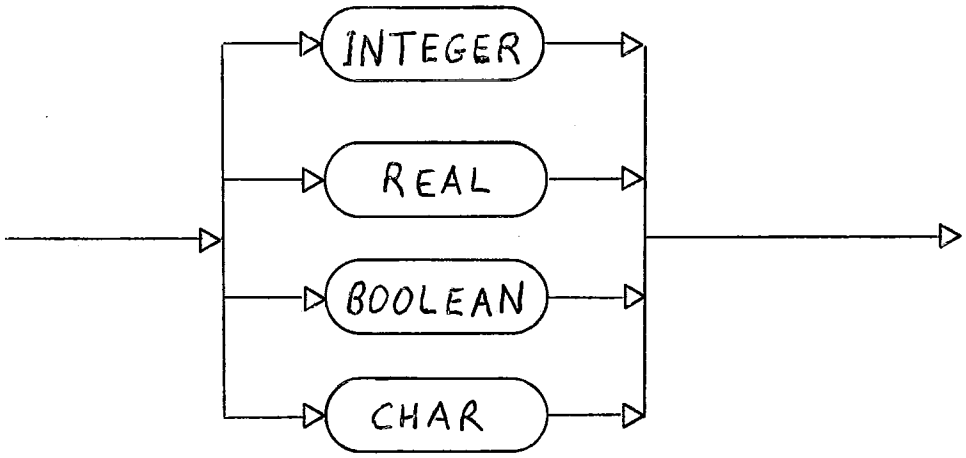
identificador



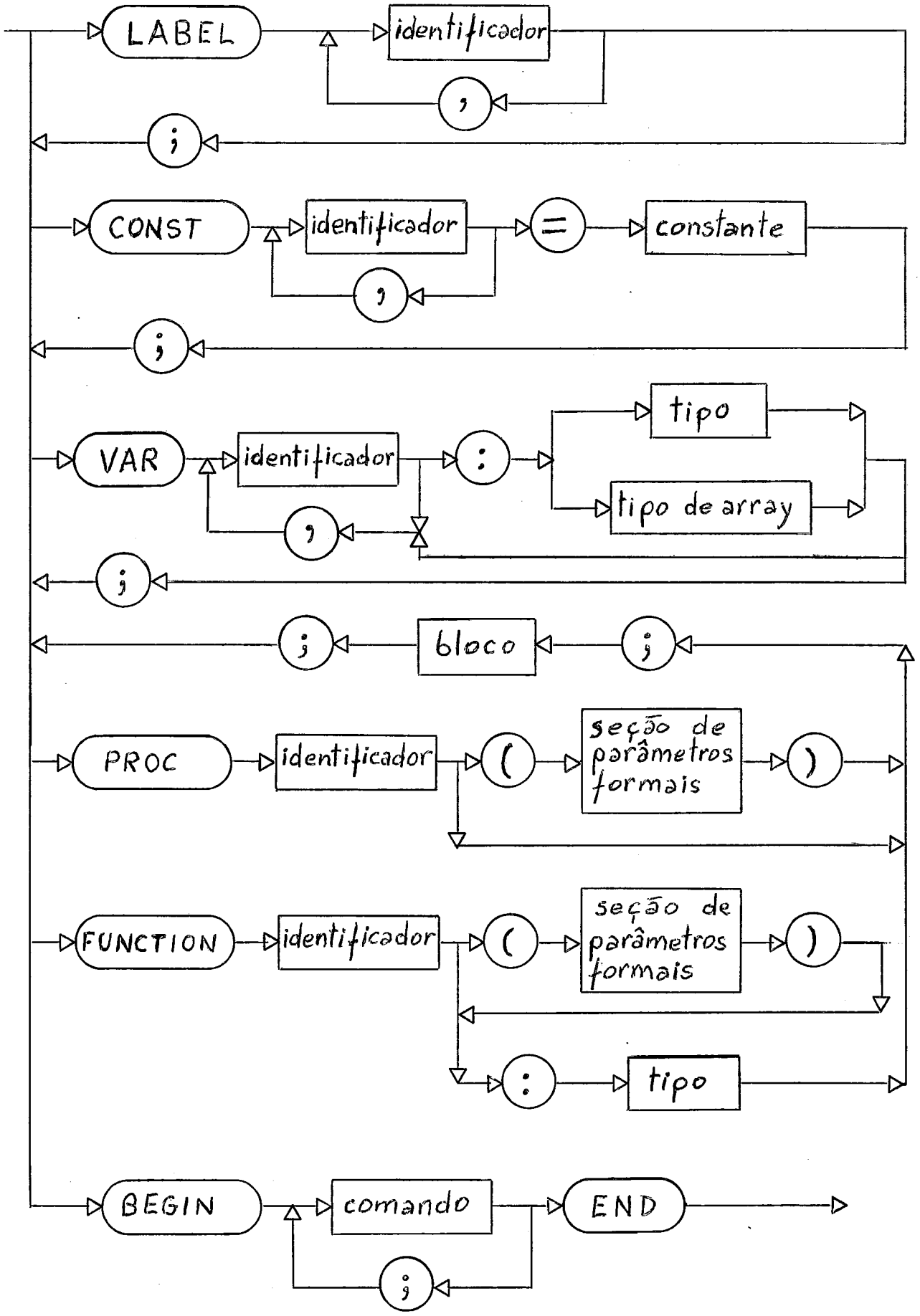
constante



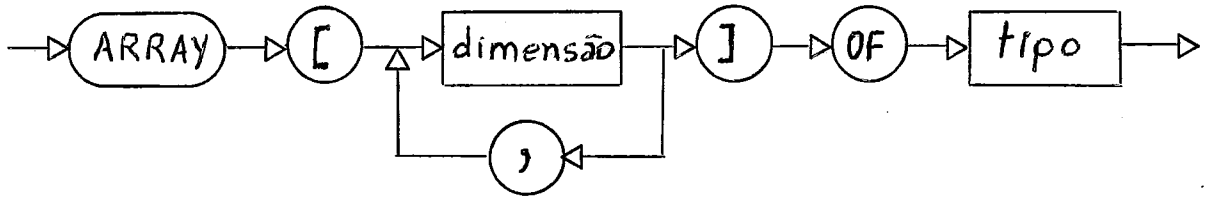
tipo



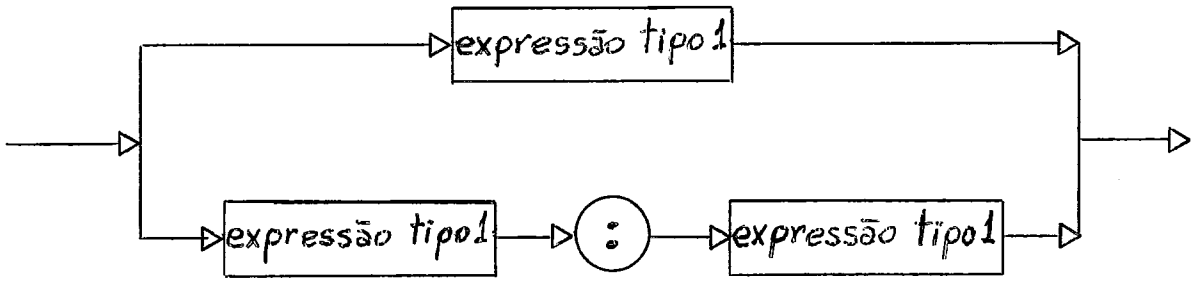
bloco


LABEL
identificador
,
;
CONST
identificador
=
constante
,
;
VAR
identificador
:
tipo
tipo de array
,
;
;
bloco
;
PROC
identificador
(
seção de parâmetros formais
)
FUNCTION
identificador
(
seção de parâmetros formais
)
:
tipo
BEGIN
comando
END
;

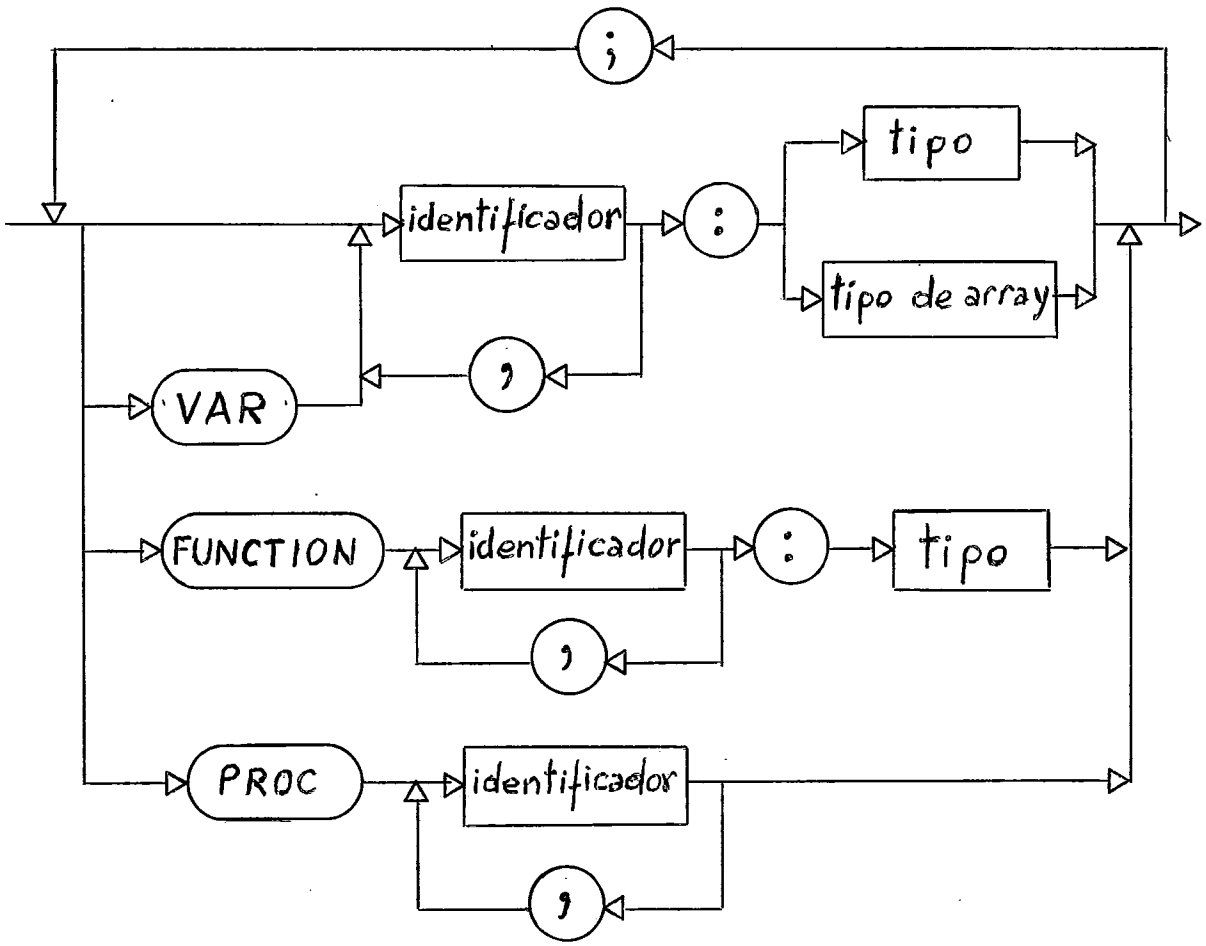
tipo de array



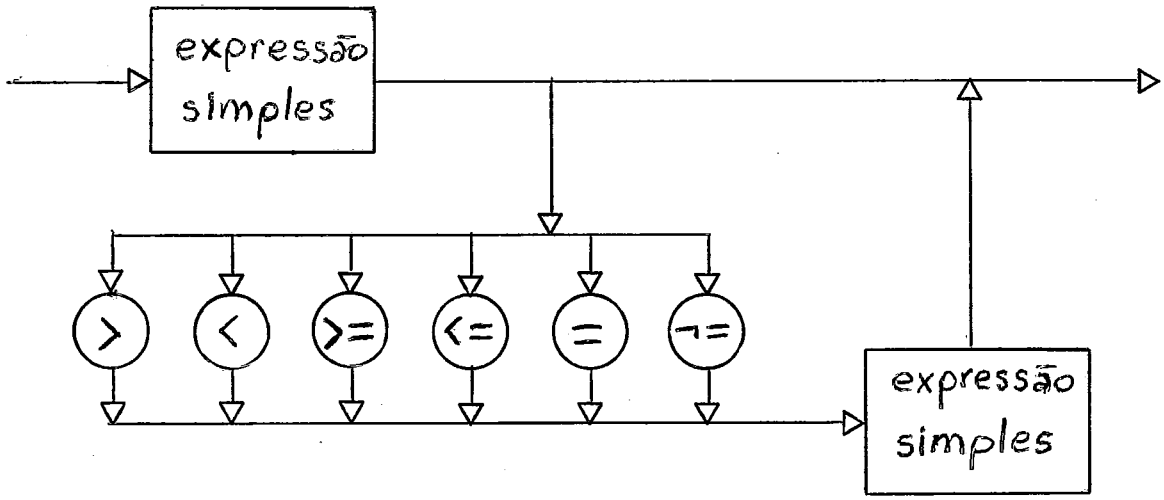
dimensão



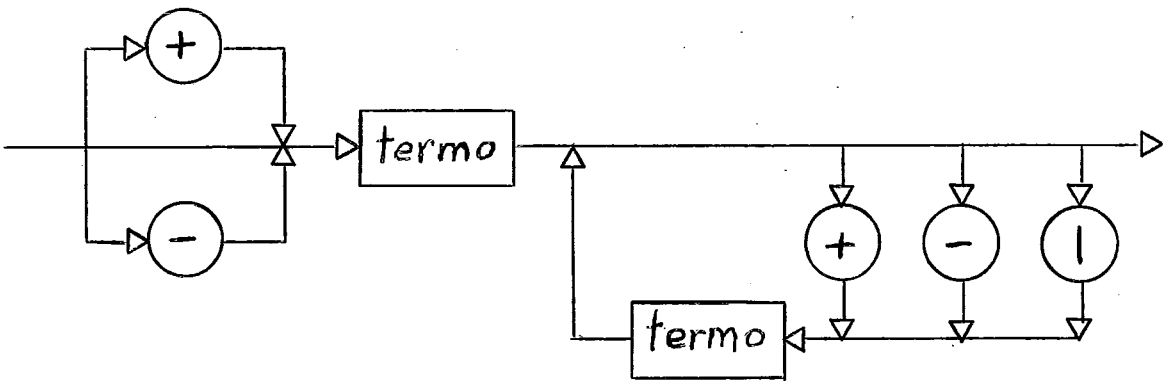
seção de parâmetros formais



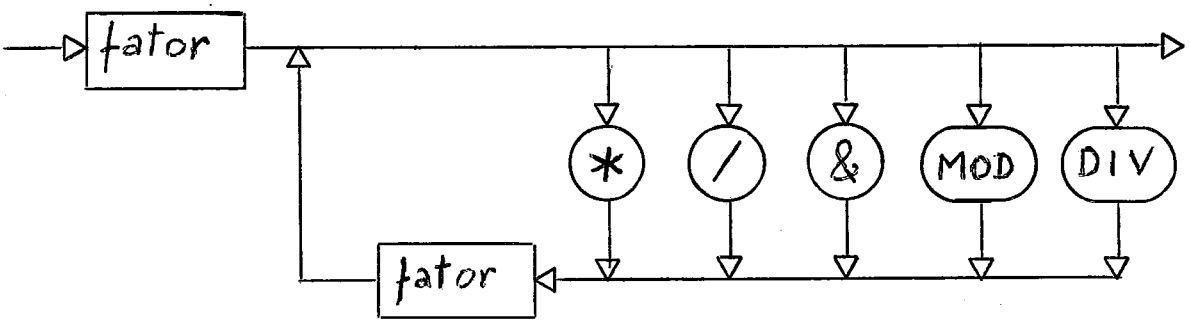
expressão



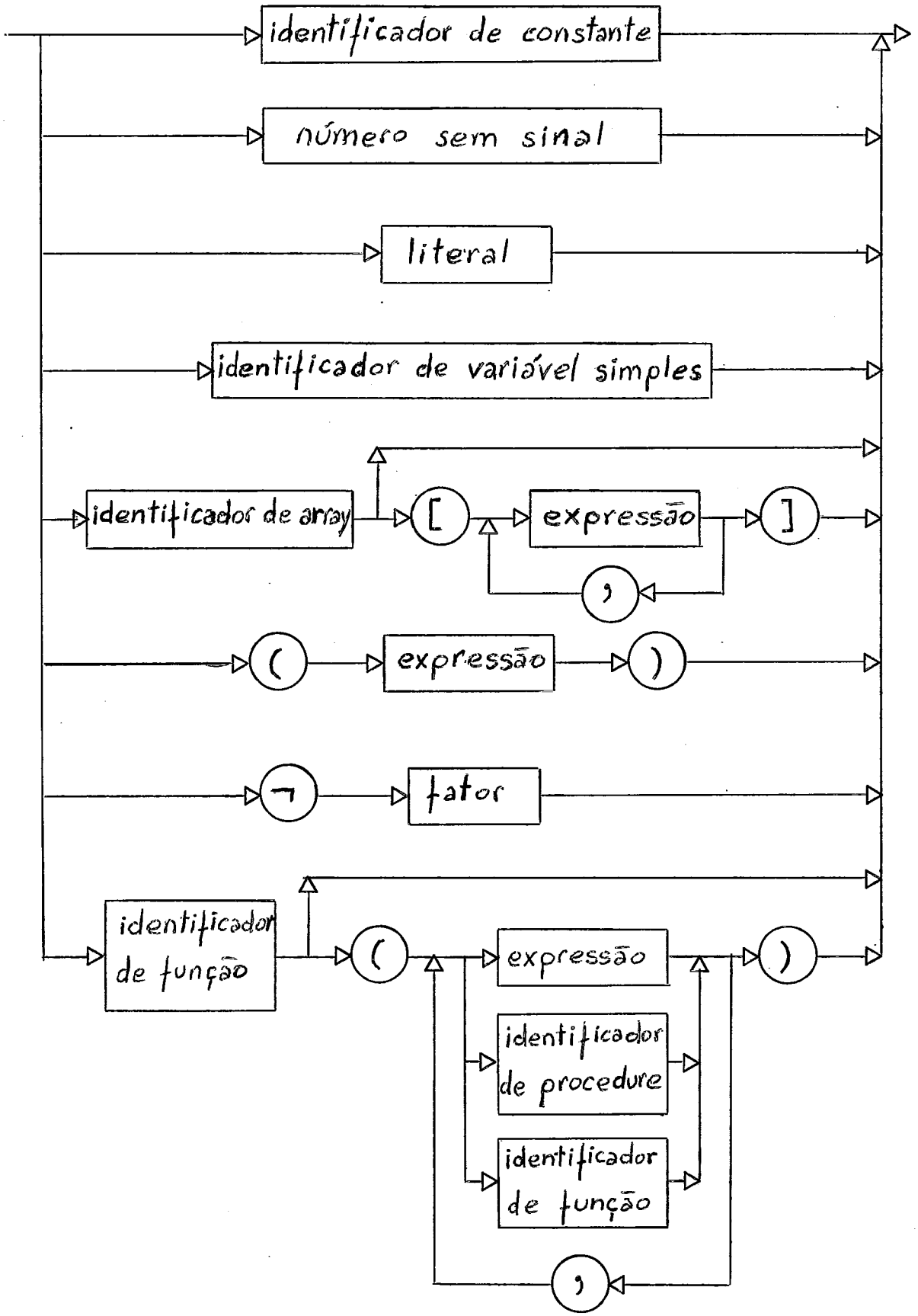
expressão simples



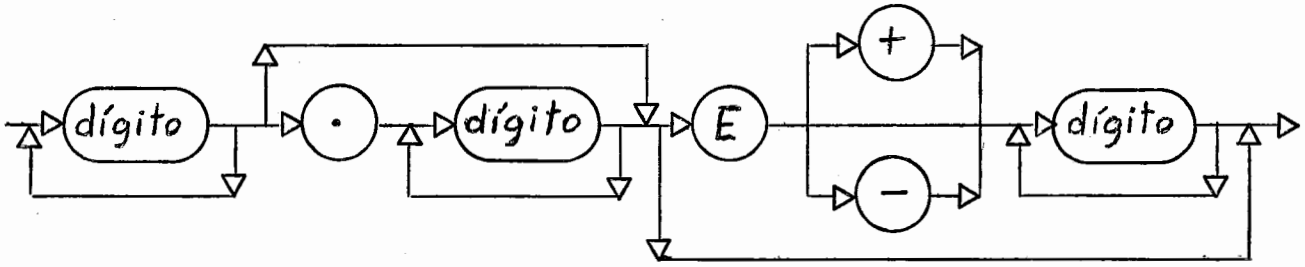
termo



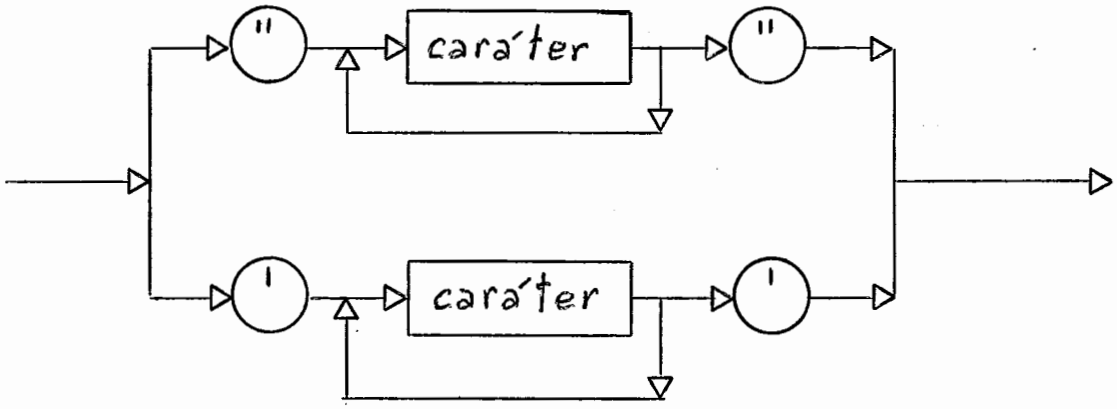
fator


identificador de constante
número sem sinal
literal
identificador de variável simples
identificador de array
[
expressão
]
,
(
expressão
)
¬
fator
identificador de função
(
expressão
identificador de procedure
identificador de função
)
,

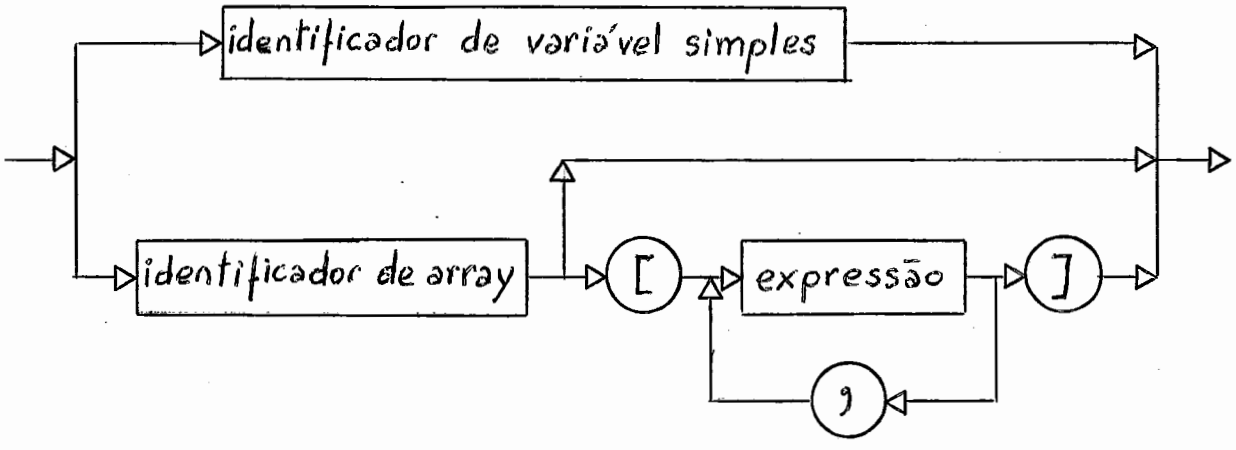
número sem sinal



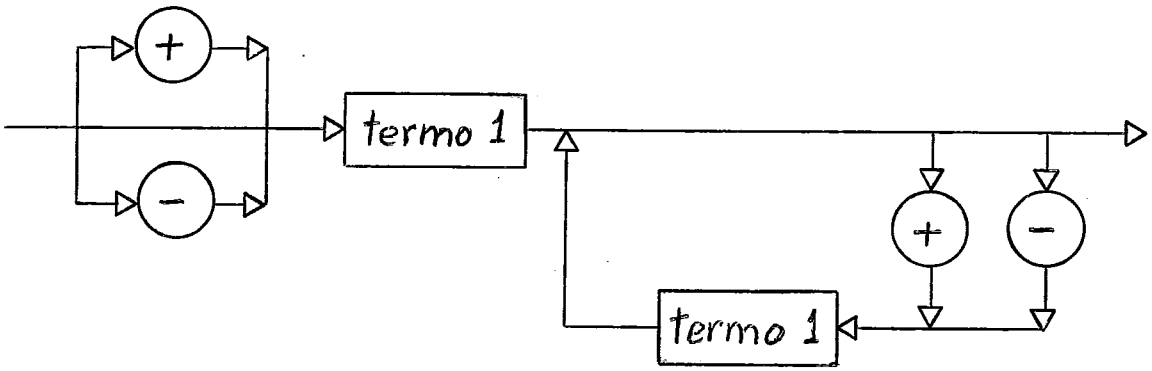
literal



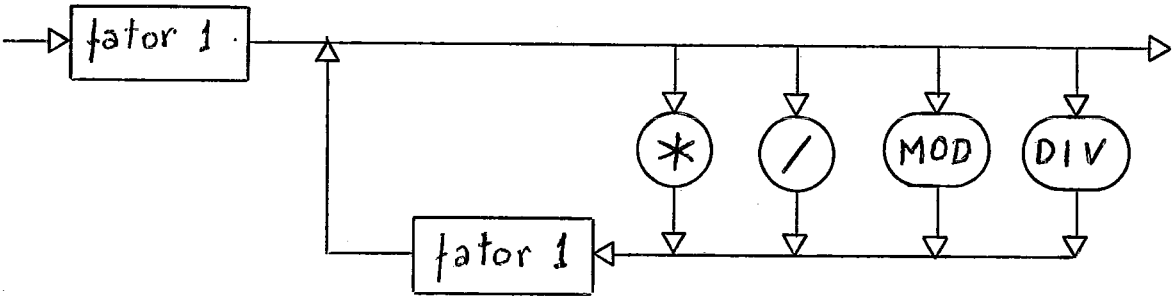
variável



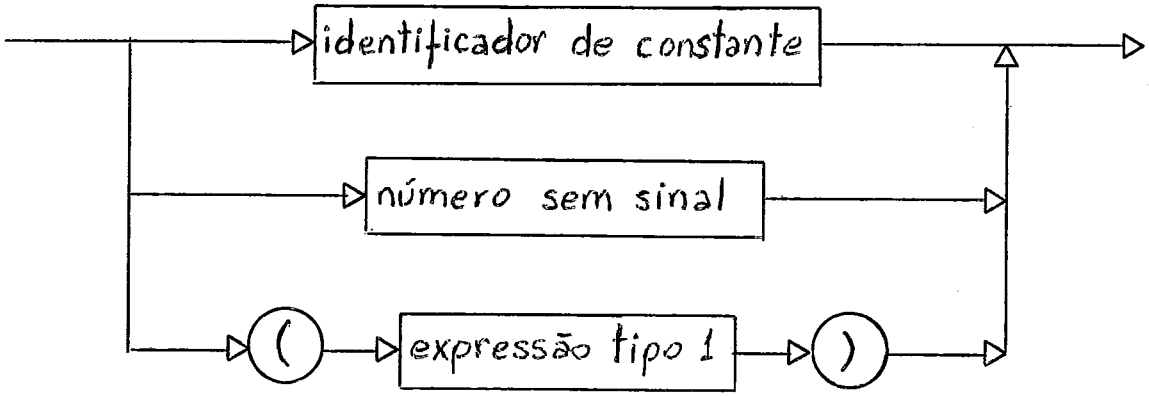
expressão tipo 1



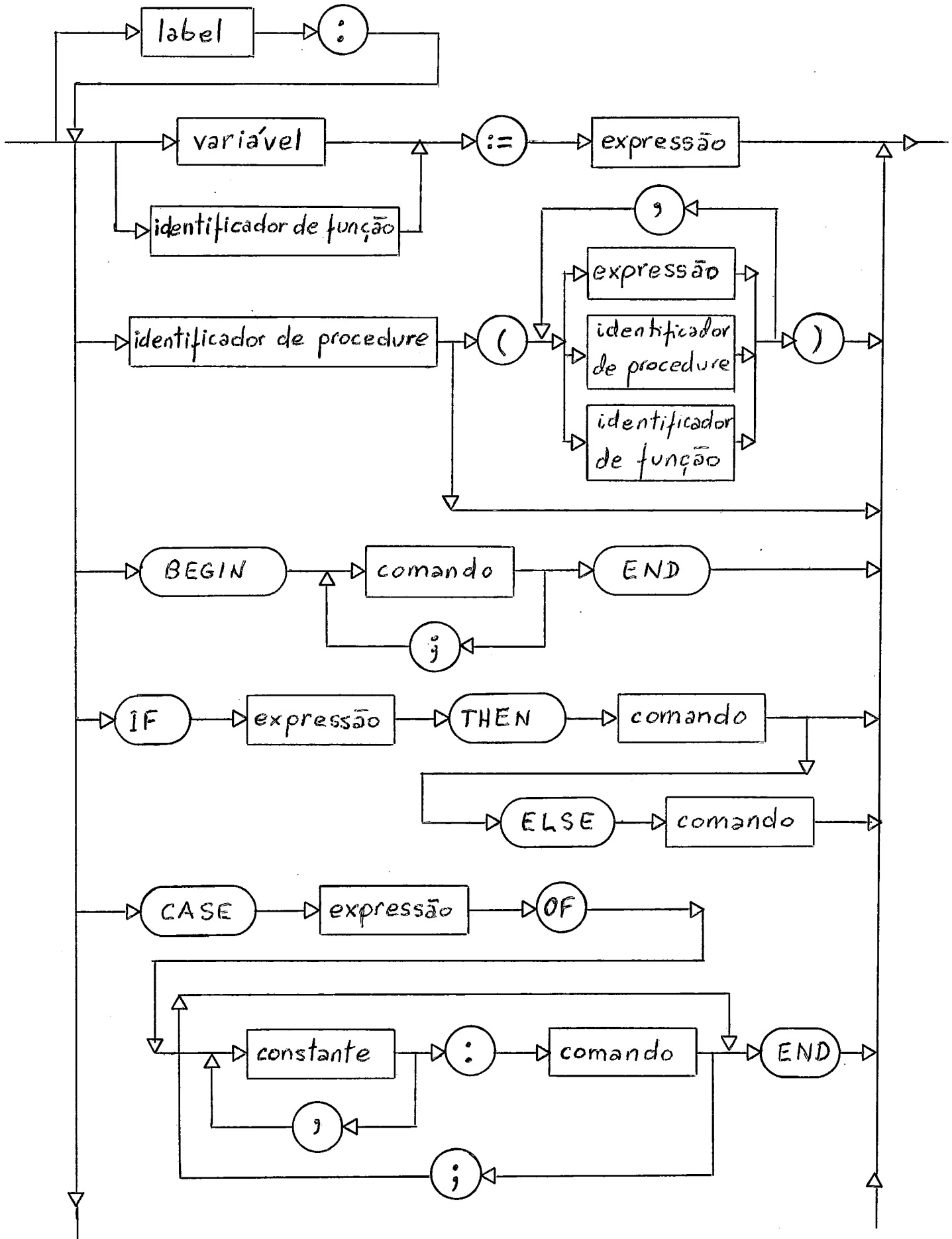
termo 1



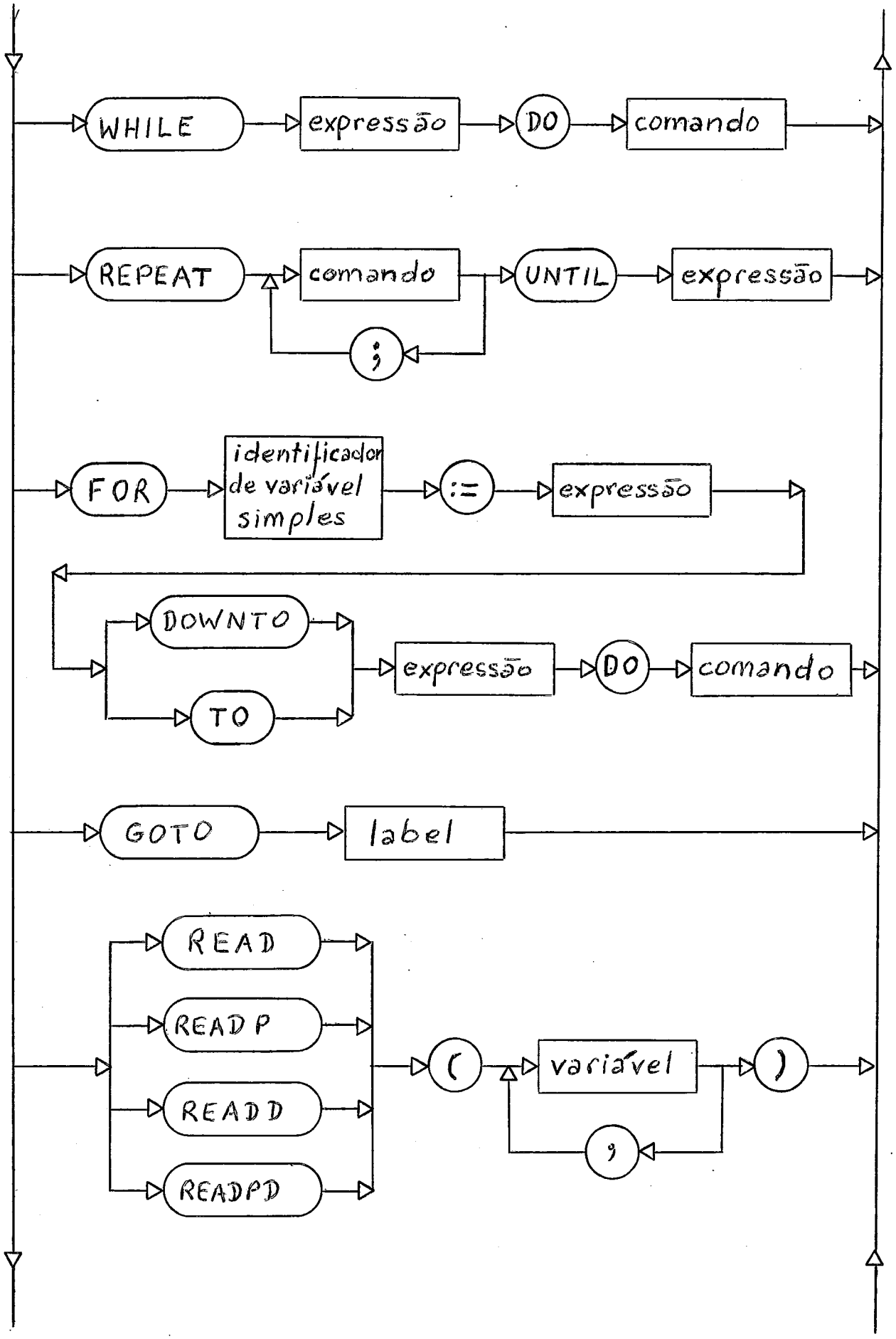
fator 1



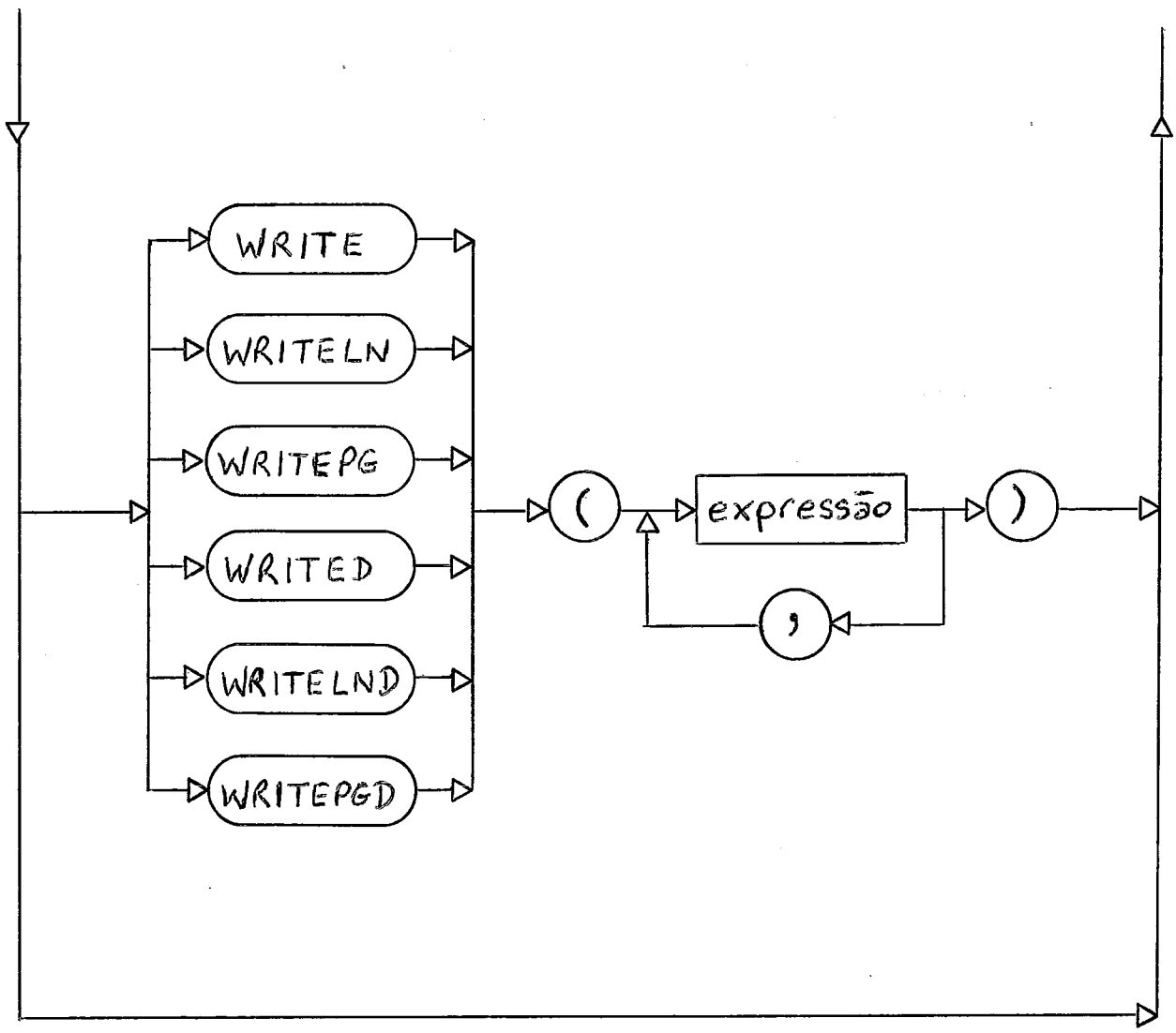
comando


label
:
variável
:=
expressão
identificador de função
identificador de procedure
(
,
expressão
identificador de procedure
identificador de função
)
BEGIN
comando
END
;
IF
expressão
THEN
comando
ELSE
comando
CASE
expressão
OF
constante
:
comando
END
,
;

comando (continuação)



comando (continuação)



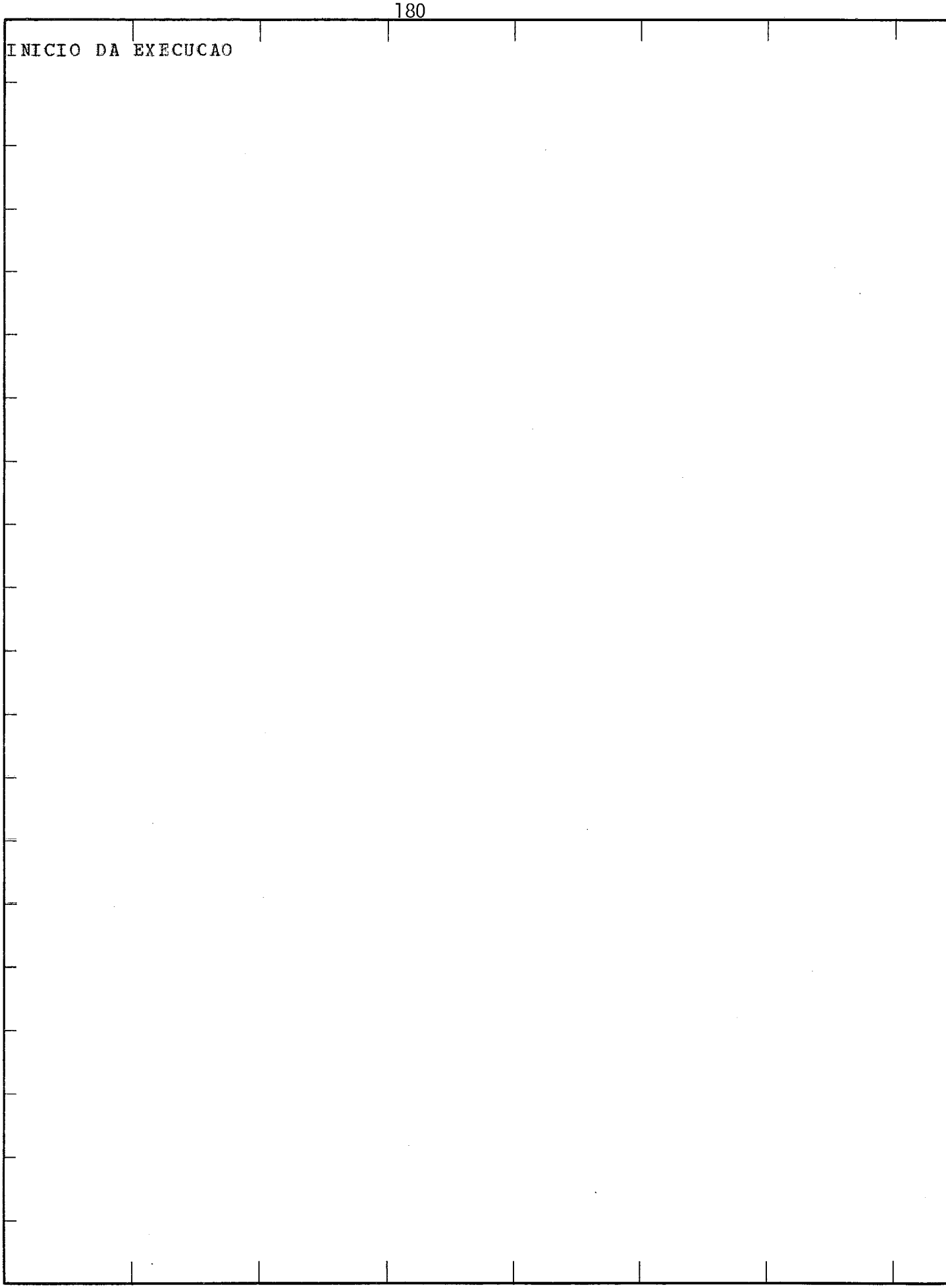
E - Exemplos de Programas PASCAL Compilados sob o CDP

Para cada exemplo a seguir são apresentados: a listagem dos dados (programa fonte e os dados para execução, se houver) e a listagem da execução.

```
PROGRAM POSTFIX ;
% ESTE PROGRAMA LE EXPRESSOES ARITMETICAS CORRETAS E AS %
% IMPRIME NA FORMA POS-FIXADA. OS OPERANDOS PODEM SER   %
% CONSTANTES INTEIRAS OU VARIAVEIS. EM AMBOS OS CASOS   %
% DEVEM SER COMPOSTAS DE APENAS UM CARATER. OS OPERADO- %
% RES SAO: +, - E *, E AS EXPRESSOES PODEM CONTER       %
% PARENTESES.                                           %
VAR CH : CHAR ;
  PROC FIND ;
  % ROTINA PARA PULAR BRANCOS %
    BEGIN
      REPEAT READD(CH) UNTIL(CH ¬= ' ')
    END ; % FIND %
  PROC EXPRE ;
  % ROTINA PARA ANALISAR EXPRESSAO %
    VAR OP : CHAR ;
    PROC TERM ;
    % ROTINA PARA ANALISAR TERMO %
      PROC FATOR ;
      % ROTINA PARA ANALISAR FATOR %
        BEGIN
          IF CH = '('
          THEN BEGIN
                 % ANALISA EXPRESSAO ENTRE PARENTESES %
                 FIND ;
                 EXPRE ;
               END
          ELSE WRITED(CH) ; % IMPRIME O OPERANDO %
          FIND
        END ; % FATOR %
      BEGIN
        FATOR ;
        WHILE CH = '*' DO
          BEGIN
            FIND ;
            FATOR ;
            WRITED ('*') ;
          END
      END ; % TERM %
    BEGIN
      TERM ;
      WHILE(CH = '+') | (CH = '-') DO
        BEGIN
          OP := CH ;
          FIND ;
          TERM ;
          WRITED(OP)  % IMPRIME OPERADOR %
        END
    END ; % EXPRE %
    BEGIN
      FIND ;
      REPEAT
        % COMECA NOVA EXPRESSAO %
        WRITED(' ') ;
        EXPRE ;
        WRITELN(' ') ;
      UNTIL CH = '.'
    END.
$
( A +   B) *  (C-   D  )
```

(A + 3 * B - (C + E) - 5) -

```
PROGRAM POSTFIX ;
% ESTE PROGRAMA LE EXPRESSOES ARITMETICAS CORRETAS E AS %
% IMPRIME NA FORMA POS-FIXADA. OS OPERANDOS PODEM SER   %
% CONSTANTES INTEIRAS OU VARIAVEIS. EM AMBOS OS CASOS   %
% DEVEM SER COMPOSTAS DE APENAS UM CARATER. OS OPERADO- %
% RES SAO: +, - E *, E AS EXPRESSOES PODEM CONTER       %
% PARENTESES.                                           %
VAR CH : CHAR ;
  PROC FIND ;
  % ROTINA PARA PULAR BRANCOS %
    BEGIN
      REPEAT READD(CH) UNTIL(CH ¬= ' ')
    END ; % FIND %
  PROC EXPRE ;
  % ROTINA PARA ANALISAR EXPRESSAO %
    VAR OP : CHAR ;
    PROC TERM ;
    % ROTINA PARA ANALISAR TERMO %
      PROC FATOR ;
      % ROTINA PARA ANALISAR FATOR %
        BEGIN
          IF CH = '('
          THEN BEGIN
                 % ANALISA EXPRESSAO ENTRE PARENTESES %
                 FIND ;
                 EXPRE ;
               END
          ELSE WRITED(CH) ; % IMPRIME O OPERANDO %
          FIND
        END ; % FATOR %
      BEGIN
        FATOR ;
        WHILE CH = '*' DO
          BEGIN
            FIND ;
            FATOR ;
            WRITED ('*') ;
          END
      END ; % TERM %
    BEGIN
      TERM ;
      WHILE(CH = '+') | (CH = '-') DO
        BEGIN
          OP := CH ;
          FIND ;
          TERM ;
          WRITED(OP)  % IMPRIME OPERADOR %
        END
    END ; % EXPRE %
    BEGIN
      FIND ;
      REPEAT
        % COMECA NOVA EXPRESSAO %
        WRITED(' ') ;
        EXPRE ;
        WRITELN(' ') ;
      UNTIL CH = '.'
    END.
```



```
AB+CD-*
 A3B*+CE+-5-

FIM NORMAL DE PROCESSAMENTO.
```

```
PROGRAM ARVORE ;
% ESTE PROGRAMA LE OS DADOS RELATIVOS A ARVORES BINARIAS %
% DE 10 NOS E EFETUA A IMPRESSAO DOS CONTEUDOS DOS NOS,  %
% PERCORRENDO INICIALMENTE A SUB-ARVORE DA ESQUERDA, EM  %
% SEGUIDA A SUB-ARVORE DA DIREITA E FINALMENTE A RAIZ.   %
% PARA CADA ARVORE SAO LIDOS: O APONTADOR PARA O NO      %
% RAIZ; OS CONTEUDOS DOS NOS; OS VALORES DOS APONTADORES %
% PARA ESQUERDA E OS VALORES DOS APONTADORES PARA        %
% DIREITA. O VALOR ZERO PARA O APONTADOR PARA O NO RAIZ  %
% INDICA FIM DE DADOS.                                   %
  VAR NOME : ARRAY#10! OF CHAR ,
      PE , PD : ARRAY#10! OF INTEGER ,
      CABEC : INTEGER ;
  PROC IMPRIME(RAIZ : INTEGER) ;
  % ROTINA PARA PERCURSO DA ARVORE E IMPRESSAO DOS NOS. %
  % ESTA ROTINA RECEBE COMO PARAMETRO O APONTADOR PARA  %
  % O NO RAIZ.                                          %
  BEGIN
    IF PE#RAIZ! ¬= 0
    THEN IMPRIME(PE#RAIZ!) ; % PERCORRE SUB-ARVORE DA ESQUERDA %
    IF PD#RAIZ! ¬= 0
    THEN IMPRIME(PD#RAIZ!) ; % PERCORRE SUB-ARVORE DA DIREITA %
    WRITED(NOME#RAIZ!)  % IMPRIME RAIZ %
  END ;
BEGIN
  READD(CABEC , NOME , PE , PD) ;
  WHILE CABEC ¬= 0 DO
    BEGIN
      WRITELN('NOME LIDO = ') ;
      IMPRIME(CABEC) ;
      READD(CABEC , NOME , PE , PD)
    END
END.
$
8  AAAL.NM.DE  0  5  4  0  0  10  0  9  2  0   0  7  6  1  0  0  0  3  0  0
8  AAAL.NM.DE  0  5  4  0  0  10  0  9  2  0   0  7  6  1  0  0  0  3  0  0
0  XXXXXXXXX   0  0  0  0  0  0   0  0  0  0   0  0  0  0  0  0  0  0  0  0
```

```
PROGRAM ARVORE ;
% ESTE PROGRAMA LE OS DADOS RELATIVOS A ARVORES BINARIAS %
% DE 10 NOS E EFETUA A IMPRESSAO DOS CONTEUDOS DOS NOS,  %
% PERCORRENDO INICIALMENTE A SUB-ARVORE DA ESQUERDA, EM  %
% SEGUIDA A SUB-ARVORE DA DIREITA E FINALMENTE A RAIZ.   %
% PARA CADA ARVORE SAO LIDOS: O APONTADOR PARA O NO      %
% RAIZ; OS CONTEUDOS DOS NOS; OS VALORES DOS APONTADORES %
% PARA ESQUERDA E OS VALORES DOS APONTADORES PARA        %
% DIREITA. O VALOR ZERO PARA O APONTADOR PARA O NO RAIZ  %
% INDICA FIM DE DADOS.                                   %
  VAR NOME : ARRAY#10! OF CHAR ,
      PE , PD : ARRAY#10! OF INTEGER ,
      CABEC : INTEGER ;
  PROC IMPRIME(RAIZ : INTEGER) ;
  % ROTINA PARA PERCURSO DA ARVORE E IMPRESSAO DOS NOS. %
  % ESTA ROTINA RECEBE COMO PARAMETRO O APONTADOR PARA  %
  % O NO RAIZ.                                          %
  BEGIN
    IF PE#RAIZ! ¬= 0
    THEN IMPRIME(PE#RAIZ!) ; % PERCORRE SUB-ARVORE DA ESQUERDA %
    IF PD#RAIZ! ¬= 0
    THEN IMPRIME(PD#RAIZ!) ; % PERCORRE SUB-ARVORE DA DIREITA %
    WRITED(NOME#RAIZ!)  % IMPRIME RAIZ %
  END ;
BEGIN
  READD(CABEC , NOME , PE , PD) ;
  WHILE CABEC ¬= 0 DO
    BEGIN
      WRITELN('NOME LIDO = ') ;
      IMPRIME(CABEC) ;
      READD(CABEC , NOME , PE , PD)
    END
END.
```

INICIO DA EXECUCAO

```
NOME LIDO = .MADALENA.
NOME LIDO = .MADALENA.
FIM NORMAL DE PROCESSAMENTO.
```

```
PROGRAM POSTFIX ;
% EXEMPLO DE PROGRAMA COM ERRO.                                    %
% ESTE PROGRAMA LE EXPRESSOES ARITMETICAS CORRETAS E AS            %
% IMPRIME NA FORMA POS-FIXADA. OS OPERANDOS PODEM SER              %
% CONSTANTES INTEIRAS OU VARIAVEIS. EM AMBOS OS CASOS              %
% DEVEM SER COMPOSTAS DE APENAS UM CARATER. OS OPERADO-            %
% RES SAO: +, - E *, E AS EXPRESSOES PODEM CONTER                  %
% PARENTESES.                                                      %
CONST VALOR = 3.  ;
VAR CH : CHAR ;
  PROC FIND ;
  % ROTINA PARA PULAR BRANCOS %
    BEGIN
      REPEAT READD(CH) UNTIL(CH ¬= ' ')
    END ; % FIND %
  PROC EXPRE ;
  % ROTINA PARA ANALISAR EXPRESSAO %
    VAR OP : CHAR ;
    PROC TERM ;
    % ROTINA PARA ANALISAR TERMO %
      PROC FATOR ;
      % ROTINA PARA ANALISAR FATOR %
        BEGIN
          IF CX = '('
          THEN BEGIN
                 % ANALISA EXPRESSAO ENTRE PARENTESES %
                 FIND ;
                 EXPRE ;
               END
          ELSE WRITED(CX) ; % IMPRIME O OPERANDO %
          FIND
        END ; % FATOR %
      BEGIN
        FATOR ;
        WHILE CH = '*' DO
          BEGIN
            FIMD ;
            FATOR ;
            WRITED '*') ;
          END
      END ; % TERM %
    BEGIN
      TERM ;
      WHILE(CH = '+') | (CH = '-') DO
        BEGIN
          OP := CH ;
          FIND
          TERM ;
          WRITED(OP)   % IMPRIME OPERADOR %
        END
    END ; % EXPRE %
    BEGIN
      FIND ;
      REPEAT
        % COMECA NOVA EXPRESSAO %
        WRITED(' ') ;
        EXPRE ;
        WRITELN(' ') ;
      UNTIL CH = '.'
    END.
```

```
                    PROGRAM POSTFIX ;
                    % EXEMPLO DE PROGRAMA COM ERRO.                          %
                    % ESTE PROGRAMA LE EXPRESSOES ARITMETICAS CORRETAS E AS %
                    % IMPRIME NA FORMA POS-FIXADA. OS OPERANDOS PODEM SER   %
                    % CONSTANTES INTEIRAS OU VARIAVEIS. EM AMBOS OS CASOS   %
                    % DEVEM SER COMPOSTAS DE APENAS UM CARATER. OS OPERADO- %
                    % RES SAO: +, - E *, E AS EXPRESSOES PODEM CONTER       %
                    % PARENTESES.                                           %
                    CONST VALOR = 3.  ;
***** ERRO  21 ********************|
      FALTA PARTE FRACIONARIA EM CONSTANTE REAL. CONSIDERADA APENAS A PARTE INTEIRA.
                    VAR CH : CHAR ;
                      PROC FIND ;
                      % ROTINA PARA PULAR BRANCOS %
                        BEGIN
                          REPEAT READD(CH)  UNTIL(CH ¬= ' ')
                        END ; % FIND %
                      PROC EXPRE ;
                      % ROTINA PARA ANALISAR EXPRESSAO %
                        VAR OP : CHAR ;
                        PROC TERM ;
                        % ROTINA PARA ANALISAR TERMO %
                          PROC FATOR ;
                          % ROTINA PARA ANALISAR FATOR %
                            BEGIN
                              IF CX = '('
***** ERRO  48 ********************|
      IDENTIFICADOR NAO DECLARADO.
                              THEN BEGIN
                                     % ANALISA EXPRESSAO ENTRE PARENTESES %
                                     FIND ;
                                     EXPRE ;
                                   END
                              ELSE WRITED(CX) ; % IMPRIME O OPERANDO %
                              FIND
                            END ; % FATOR %
                          BEGIN
                            FATOR ;
                            WHILE CH = '*' DO
                              BEGIN
                                FIMD ;
***** ERRO  48 ********************|
      IDENTIFICADOR NAO DECLARADO.
***** ERRO  94 ********************|
      IDENTIFICADOR INVALIDO PARA INICIO DE COMANDO.
***** ERRO  14 ********************|
      SIMBOLOS ANTERIORES, A PARTIR DA ULTIMA MENSAGEM DE ERRO, FORAM IGNORADOS.
                                FATOR ;
                                WRITED '*') ;
***** ERRO  92 ************************|
      ESPERADO "(". INSERIDO.
                              END
                          END ; % TERM %
                        BEGIN
                          TERM ;
                          WHILE(CH = '+') | (CH = '-') DO
                            BEGIN
                              OP := CH ;
                              FIND
                              TERM ;
```

```
***** ERRO  12 ***********
      FALTA ";". FOI ASSUMIDO.
                              WRITED(OP)   % IMPRIME OPERADOR %
                            END
                        END ; % EXPRE %
                        BEGIN
                          FIND ;
                          REPEAT
                            % COMECA NOVA EXPRESSAO %
                            WRITED(' ') ;
                            EXPRE ;
                            WRITELN(' ') ;
                          UNTIL CH = '.'
                        END.
```

```
PROGRAM MEDIA ;
  % EXEMPLO DE PROGRAMA COM ERRO %
  VAR SOMA , CONT : INTEGER , NUM : INTEGER ,
      MED : REAL ;
  BEGIN
    SOMA := 0 ;
    CONT := 0 ;
    NUM := 3 ;
    WHILE NUM < 98 DO
    BEGIN
      SOMA := SOMA + NUM
      CONT := CONT + 1 ;
      NUM := NUM + + 2 ;
    END ;
    MED := SOMA / DONT ;
    WRITELN ('MEDIA = ' , MED) ;
  END .
```

```
                    PROGRAM MEDIA ;
                      % EXEMPLO DE PROGRAMA COM ERRO %
                      VAR SOMA , CONT : INTEGER , NUM : INTEGER ,
                          MED : REAL ;
                      BEGIN
                        SOMA := 0 ;
                        CONT := 0 ;
                        NUM := 3 ;
                        WHILE NUM < 98 DO
                        BEGIN
                          SOMA := SOMA + NUM
                          CONT := CONT + 1 ;
***** ERRO  12 **************|
      FALTA ";". FOI ASSUMIDO.
                          NUM := NUM + + 2 ;
***** ERRO 116 ************************|
      FALTA OPERANDO.
                        END ;
                        MED := SOMA / DONT ;
***** ERRO  48 ************************|
      IDENTIFICADOR NAO DECLARADO.
                        WRITELN ('MEDIA = ' , MED) ;
                      END .
```

```
PROGRAM FIB ;
  % EXEMPLO DE PROGRAMA COM ERRO %
  VAR TERM :ARRAY # 10 ! OF INTEGER ;
  BEGIN
    TER #1! := 0 ;
    TERM #2! := 1 ;
    WRITELN ( TERM#1!,TERM#2!  ;
    FOR I := 3 TO 10 DO
       BEGIN
         TERM #I! := TERM#I-1! + TERM#I-2! ;
         WRITE (TERM I!) ;
       END ;
  END .
```

```
                    PROGRAM FIB ;
                      % EXEMPLO DE PROGRAMA COM ERRO %
                      VAR TERM :ARRAY # 10 ! OF INTEGER ;
                      BEGIN
                        TER #1! := 0 ;
***** ERRO  48 **********|
      IDENTIFICADOR NAO DECLARADO.
***** ERRO  94 **********|
      IDENTIFICADOR INVALIDO PARA INICIO DE COMANDO.
***** ERRO  14 **********************|
      SIMBOLOS ANTERIORES, A PARTIR DA ULTIMA MENSAGEM DE ERRO, FORAM IGNORADOS.
                        TERM #2! := 1 ;
                        WRITELN ( TERM#1!,TERM#2!  ;
***** ERRO 102 ********************************|
      SIMBOLO INVALIDO. "," OU ")" ESPERADO.
***** ERRO  14 ********************************|
      SIMBOLOS ANTERIORES, A PARTIR DA ULTIMA MENSAGEM DE ERRO, FORAM IGNORADOS.
***** ERRO 103 ********************************|
      FALTA ")". INSERIDO.
                        FOR I := 3 TO 10 DO
***** ERRO  48 ************|
      IDENTIFICADOR NAO DECLARADO.
                          BEGIN
                            TERM #I! := TERM#I-1! + TERM#I-2! ;
                            WRITE (TERM I!) ;
***** ERRO 102 **************************|
      SIMBOLO INVALIDO. "," OU ")" ESPERADO.
***** ERRO  14 **************************|
      SIMBOLOS ANTERIORES, A PARTIR DA ULTIMA MENSAGEM DE ERRO, FORAM IGNORADOS.
                          END ;
                    END .
```

F - Listagens das Macros

```
/*****************************************************************/
/* MACRO PLAMA03                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPA03" :                  */
/*   ESTA FUNCAO ASSEMBLER CALCULA O HOME-ADDRESS DO IDENTI-     */
/*   FICADOR PARA A BUSCA NA TABELA DE SIMBOLOS.                 */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.   */
/*   PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIM-    */
/*             BOLOS - ENTRY-POINT DA PLAPP05.                   */
/*****************************************************************/

DCL PLAPA03      ENTRY(CHAR(8)) EXTERNAL RETURNS(BIN FIXED(15,0)) ;

/*****************************************************************/
/* MACRO PLAMP01                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP01" :                  */
/*   ESTA PROCEDURE IMPRIME MENSAGENS DE ERROS.                  */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                   */
/*   PLAPP03 - RECONHECEDOR DAS OPCOES DE COMPILACAO.            */
/*   PLAPP04 - RECONHECEDOR LEXICO DO COMPILADOR.                */
/*   PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.   */
/*   PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIMBO-  */
/*             LOS - ENTRY-POINT DA PLAPP05.                     */
/*   PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES  */
/*             DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO       */
/*             ENCERRADA - ENTRY-POINT DA PLAPP05.               */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                            */
/*   PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.         */
/*   PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.          */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.           */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.              */
/*   PLAPP13 - INSERE CONSTANTE NA TABELA DE CONSTANTES INTEIRAS */
/*   PLAPP14 - INSERE CONSTANTE NA TABELA DE CONSTANTES REAIS -  */
/*             ENTRY-POINT DA PLAPP13.                           */
/*   PLAPP15 - INSERE CONSTANTE NA TABELA DE CONSTANTES ALFANU-  */
/*             MERICAS - ENTRY-POINT DA PLAPP13.                 */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS                          */
/*   PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS.                */
/*   PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1.                 */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                 */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                  */
/*   PLAPP23 - RECONHECEDOR DE COMANDO "IF".                     */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                   */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                    */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                   */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                  */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.        */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS.                */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                        */
/*   PLAPP34 - GRAVA REGISTRO DE INSTRUCOES EM "OBJ1".           */
/*                                                               */
/*****************************************************************/
```

```
DCL PLAPP01         ENTRY EXTERNAL ;


/*****************************************************************/
/* MACRO PLAMP02                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP02" :                  */
/*   ESTA PROCEDURE IMPRIME A TABELA DE SIMBOLOS DO COMPILADOR   */
/*   PARA FINS DE DEPURACAO.                                     */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.   */
/*   PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIMBO-  */
/*             LOS - ENTRY-POINT DA PLAPP05.                     */
/*   PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES  */
/*             DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO       */
/*             ENCERRADA - ENTRY-POINT DA PLAPP05.               */
/*****************************************************************/


DCL PLAPP02         ENTRY(BIN FIXED(15,0)) EXTERNAL ;


/*****************************************************************/
/* MACRO PLAMP03                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP03" :                  */
/*   ESTA FUNCAO DECODIFICA O PARM RECEBIDO PELO MODULO PRIN-    */
/*   PAL, RECONHECENDO AS OPCOES DE COMPILACAO DESEJADAS E       */
/*   EFETUANDO AS INICIALIZACOES NECESSARIAS.                    */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                   */
/*****************************************************************/


DCL PLAPP03         ENTRY(CHAR(100)) EXTERNAL RETURNS(BIT(1)) ;


/*****************************************************************/
/* MACRO PLAMP04                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP04" :                  */
/*   ESTA PROCEDURE E' O RECONHECEDOR LEXICO DO COMPILADOR.      */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                   */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                            */
/*   PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.         */
/*   PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.          */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.           */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.              */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                         */
/*   PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS.                */
/*   PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1.                 */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                 */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                  */
/*   PLAPP23 - RECONHECEDOR DE COMANDO "IF".                     */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                   */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                    */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                   */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                  */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.        */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS.                */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                        */
```

```
/*                                                                    */
/**********************************************************************/

DCL PLAPP04          ENTRY EXTERNAL ;

/**********************************************************************/
/* MACRO PLAMP05                                                      */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "PLAPP05" , "PLAPP06" E        */
/*   "PLAPP07" :                                                      */
/*   PLAPP05 , PLAPP06 E PLAPP07 SAO OS 3 ENTRY-POINTS DA             */
/*   PROCEDURE PIAPP05, UTILIZADA PARA MANIPULAR COM A TABELA         */
/*   DE SIMBOLOS DO COMPILADOR.                                       */
/*   PLAPP05 E' UTILIZADA PARA PROCURAR UM IDENTIFICADOR NA           */
/*           TABELA.                                                  */
/*   PLAPP06 E' UTILIZADA PARA INSERIR NA TABELA UM NOVO              */
/*           IDENTIFICADOR.                                           */
/*   PLAPP07 E' UTILIZADA PARA ELIMINAR DA TABELA OS IDENTIFI-        */
/*           CADORES DECLARADOS NO BLOCO CUJA ANALISE ESTA'           */
/*           SENDO ENCERRADA.                                         */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                           */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                        */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                                 */
/*   PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.              */
/*   PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.               */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.                */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.                   */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                              */
/*   PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1.                      */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.           */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                        */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                         */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                        */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS .            */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                             */
/**********************************************************************/

DCL PLAPP05          ENTRY EXTERNAL ;
DCL PLAPP06          ENTRY EXTERNAL ;
DCL PLAPP07          ENTRY EXTERNAL ;

/**********************************************************************/
/* MACRO PLAMP06                                                      */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP06" :                       */
/*   ESTA PROCEDURE E' ENTRY-POINT DA PLAPP05.                        */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                           */
/*           - POR SER ENTRY-POINT ESTA MACRO NAO E' UTILIZADA.       */
/**********************************************************************/


/**********************************************************************/
/* MACRO PLAMP07                                                      */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP07" :                       */
/*   ESTA PROCEDURE E' ENTRY-POINT DA PLAPP05.                        */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                           */
/*           - POR SER ENTRY-POINT ESTA MACRO NAO E' UTILIZADA.       */
/**********************************************************************/
```

```
/*********************************************************************/
/* MACRO PLAMP08                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP08" :                      */
/*    ESTA PROCEDURE (RECURSIVA) E' O RECONHECEDOR DE BLOCO DO       */
/*    COMPILADOR.                                                    */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                      */
/*********************************************************************/

DCL PLAPP08          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP09                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP09" :                      */
/*    ESTA PROCEDURE RECONHECE DECLARACAO DE CONSTANTE.              */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                               */
/*********************************************************************/

DCL PLAPP09          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP10                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP10" :                      */
/*    ESTA PROCEDURE RECONHECE DECLARACAO DE VARIAVEIS.              */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                               */
/*********************************************************************/

DCL PLAPP10          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP11                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP11" :                      */
/*    ESTA PROCEDURE RECONHECE CABECALHO DE PROCEDURE.               */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                               */
/*********************************************************************/

DCL PLAPP11          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP12                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP12" :                      */
/*    ESTA PROCEDURE RECONHECE CABECALHO DE FUNCAO.                  */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                               */
/*********************************************************************/

DCL PLAPP12          ENTRY EXTERNAL ;
```

```
/*********************************************************************/
/* MACRO PLAMP13                                                     */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "PLAPP13", "PLAPP14" E        */
/*    "PLAPP15" :                                                    */
/*    PLAPP13, PLAPP14 E PLAPP15 SAO OS 3 ENTRY-POINTS DA            */
/*    PROCEDURE PLAPP13, UTILIZADA PARA MANIPULAR COM AS TABELAS     */
/*    DE CONSTANTES DO PROGRAMA DO USUARIO.                          */
/*    PLAPP13 E' UTILIZADA PARA INSERIR UMA CONSTANTE INTEIRA NA     */
/*    TABELA DE CONSTANTES INTEIRAS (TABINT).                        */
/*    PLAPP14 E' UTILIZADA PARA INSERIR UMA CONSTANTE REAL NA TA-    */
/*    BELA DE CONSTANTES REAIS (TABREAL).                            */
/*    PLAPP15 E' UTILIZADA PARA INSERIR UMA CONSTANTE ALFANUMERI-    */
/*    CA NA TABELA DE CONSTANTES ALFANUMERICAS (TABALFA).            */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                               */
/*    PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.            */
/*    PLAPP17 - RECCNHECEDOR DE COMANDOS.                            */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.         */
/*    PLAPP24 - RECCNHECEDOR DE COMANDO "CASE".                      */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                       */
/*    PLAPP26 - RECCNHECEDOR DE COMANDO "READ".                      */
/*    PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                     */
/*    PLAPP28 - RECCNHECEDOR DE SECAO DE PARAMETROS REAIS.           */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY                     */
/*    PLAPP30 - RECCNHECEDOR DE EXPRESSAO.                           */
/*********************************************************************/

DCL PLAPP13         ENTRY EXTERNAL ;
DCL PLAPP14         ENTRY EXTERNAL ;
DCL PLAPP15         ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP14                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP14" :                      */
/*   ESTA PROCEDURE E' ENTRY-POINT DA PLAPP13.                       */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*           - POR SER ENTRY-POINT ESTA MACRO NAO E' UTILIZADA.      */
/*********************************************************************/

/*********************************************************************/
/* MACRO PLAMP15                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP15" :                      */
/*   ESTA PROCEDURE E' ENTRY-POINT DA PLAPP13.                       */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*           - POR SER ENTRY-POINT ESTA MACRO NAO E' UTILIZADA.      */
/*********************************************************************/

/*********************************************************************/
/* MACRO PLAMP17                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP17" :                      */
/*   ESTA PROCEDURE RECONHECE COMANDOS.                              */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                                */
/*   PLAPP21 - RECCNHECEDOR DE COMANDO "REPEAT".                     */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                      */
/*   PLAPP23 - RECCNHECEDOR DE COMANDO "IF".                         */
```

```
/*    PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                    */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                     */
/*****************************************************************/

DCL PLAPP17          ENTRY EXTERNAL ;

/*****************************************************************/
/* MACRO PLAMP18                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP18" :                  */
/*    ESTA PROCEDURE E' O RECONHECEDOR DE TIPO DE VARIAVEIS.     */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*    PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.         */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.     */
/*****************************************************************/

DCL PLAPP18          ENTRY(BIN FIXED(15,0),BIN FIXED(15,0)) EXTERNAL;

/*****************************************************************/
/* MACRO PLAMP19                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP19" :                  */
/*    ESTA FUNCAO E' O RECONHECEDOR DE EXPRESSAO DO TIPO 1.      */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*    PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS.               */
/*****************************************************************/

DCL PLAPP19          ENTRY(BIN FIXED(15,0)) EXTERNAL RETURNS(BIT(1));

/*****************************************************************/
/* MACRO PLAMP20                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP20" :                  */
/*    ESTA FUNCAO E' O RECONHECEDOR DE SECAO DE PARAMETROS       */
/*    FORMAIS.                                                   */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*    PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.          */
/*    PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.             */
/*****************************************************************/

DCL PLAPP20          ENTRY(BIN FIXED(15,0),BIN FIXED(15,0))
                     EXTERNAL RETURNS(BIN FIXED(15,0)) ;

/*****************************************************************/
/* MACRO PLAMP21                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP21"                    */
/*    ESTA PROCEDURE RECONHECE O COMANDO "REPEAT".               */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS                         */
/*****************************************************************/

DCL PLAPP21          ENTRY EXTERNAL ;
```

```
/*********************************************************************/
/* MACRO PLAMP22                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP22"                        */
/*    ESTA PROCEDURE RECONHECE O COMANDO "WHILE".                    */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS                             */
/*********************************************************************/

DCL PLAPP22          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP23                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP23"                        */
/*    ESTA PROCEDURE RECONHECE O COMANDO "IF.                        */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS                             */
/*********************************************************************/

DCL PLAPP23          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP24                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP24"                        */
/*    ESTA PROCEDURE RECONHECE O COMANDO "CASE".                     */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS                             */
/*********************************************************************/

DCL PLAPP24          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP25                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP25"                        */
/*    ESTA PROCEDURE RECONHECE O COMANDO "FOR".                      */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS                             */
/*********************************************************************/

DCL PLAPP25          ENTRY EXTERNAL ;

/*********************************************************************/
/* MACRO PLAMP26                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP26"                        */
/*    ESTA PROCEDURE RECONHECE O COMANDO "READ".                     */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS                             */
/*********************************************************************/

DCL PLAPP26          ENTRY EXTERNAL ;
```

```
/*********************************************************************/
/* MACRO PLAMP27                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP27"                        */
/*   ESTA PROCEDURE RECONHECE O COMANDO "WRITE".                     */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS                              */
/*********************************************************************/


DCL PLAPP27         ENTRY EXTERNAL ;


/*********************************************************************/
/* MACRO PLAMP28                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP28"                        */
/*   ESTA PROCEDURE RECONHECE SECAO DE PARAMETROS REAIS DE           */
/*   PROCEDURES E FUNCOES.                                           */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                             */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                            */
/*********************************************************************/


DCL PLAPP28         ENTRY (BIN FIXED(15,0),BIN FIXED(15,0)) EXTERNAL;


/*********************************************************************/
/* MACRO PLAMP29                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP29".                       */
/*   ESTA PROCEDURE RECONHECE INDICES DE ARRAYS.                     */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                             */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                       */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                            */
/*********************************************************************/


DCL PLAPP29         ENTRY (BIN FIXED(15,0) , BIN FIXED(15,0) ,
                           BIN FIXED(15,0) , BIN FIXED(15,0))
                    EXTERNAL ;


/*********************************************************************/
/* MACRO PLAMP30                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP30"                        */
/*   ESTA PROCEDURE RECONHECE EXPRESSOES.                            */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS                              */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                     */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                      */
/*   PLAPP23 - RECONHECEDOR DE COMANDO "IF".                         */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                       */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                        */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                      */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.            */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS.                    */
/*********************************************************************/


DCL PLAPP30         ENTRY EXTERNAL ;
```

```
/*****************************************************************/
/* MACRO PLAMP31                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP31"                    */
/*   ESTA PROCEDURE GERA PROGRAMA OBJETO EM DISCO.               */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - PROGRAMA PRINCIPAL.                               */
/*****************************************************************/

DCL PLAPP31         ENTRY EXTERNAL ;

/*****************************************************************/
/* MACRO PLAMP32                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP32"                    */
/*   ESTA PROCEDURE EXECUTA O PROGRAMA COMPILADO.                */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - PROGRAMA PRINCIPAL.                               */
/*****************************************************************/

DCL PLAPP32         ENTRY EXTERNAL ;

/*****************************************************************/
/* MACRO PLAMP33                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP33".                   */
/*   ESTA PROCEDURE IMPRIME MENSAGENS DURANTE EXECUCAO DO        */
/* PROGRAMA COMPILADO.                                           */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - PROGRAMA PRINCIPAL                                */
/*   PLAPP32 - EXECUTA O PROGRAMA COMPILADO.                     */
/*****************************************************************/

DCL PLAPP33         ENTRY (BIN FIXED(15,0)) EXTERNAL ;

/*****************************************************************/
/* MACRO PLAMP34                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "PLAPP34".                   */
/*   ESTA PROCEDURE VERIFICA SE O REGISTRO DE INSTRUCOES         */
/* (QUADRUPLAS) GERADAS JA' ESTA' COMPLETO, GRAVANDO-O EM CASO   */
/* POSITIVO. ALEM DISSO, ATUALIZA O CONTADOR DE INSTRUCOES       */
/* ('CINST') E O PONTEIRO PARA AS QUADRUPLAS NO REGISTRO COR-    */
/* RENTE ('LC1').                                                */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                   */
/*   PLAPP08 - RECONHECECOR DE BLOCO.                            */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.           */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.              */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                         */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                 */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                  */
/*   PLAPP23 - RECONHECEDOR DE COMANDO "IF".                     */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                   */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                    */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                   */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                  */
```

```
/*    PLAPP28 - RECONHECEDOR DE PARAMETROS REAIS.                    */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                    */
/*    PLAPP30 - RECONHECEDOR DE EXPRESSAO.                           */
/*****************************************************************/


DCL PLAPP34 ENTRY(BIN FIXED(15,0) , BIN FIXED(15,0) ,
                  BIN FIXED(31,0) , BIN FIXED(15,0) ,
                  BIN FIXED(31,0) , BIN FIXED(15,0) ,
                  BIN FIXED(31,0)) EXTERNAL ;


/*****************************************************************/
/* MACRO PLAMV01                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "AUXSCAN" :                  */
/*   ESTA ESTRUTURA CONTEM VARIAVEIS AUXILIARES PARA COMUNICA-   */
/*   CAO ENTRE A PROCEDURE 'PLAPP04' E DEMAIS PARTES DO COMPI-   */
/*   LADOR SE O SIMBOLO RECONHECIDO FOR IDENTIFICADOR, NUMERO    */
/*   INTEIRO, NUMERO REAL OU LITERAL, O SIMBOLO E' DEVOLVIDO NA  */
/*   VARIAVEL "IDENT", "NINT", "NREAL" OU "STRING", RESPECTIVA-  */
/*   MENTE. NO ULTIMO CASO, "TAM" CONTERA' O TAMANHO DO LITERAL. */
/*   "PTR" E' O PONTEIRO QUE PERCORRE O PROGRAMA FONTE E "TIPO"  */
/*   E' UTILIZADO PARA DEVOLVER O CODIGO DO SIMBOLO RECONHECIDO, */
/*   DE ACORDO COM A TABELA ABAIXO:                              */
/* TIPO ELEMENTO          TIPO ELEMENTO          TIPO ELEMENTO  */
/*    1 IDENTIFICADOR      13 :                   31 |           */
/*    2 NUMERO INTEIRO     14 ABRE COLCHETE  #    32 ¬           */
/*    3 NUMERO REAL        15 FECHA COLCHETE      35 ¬=          */
/*    4 LITERAL            16 :=                  36 >           */
/*    8 .                  20 +                   37 <           */
/*    9 ,                  21 -                   38 =           */
/*   10 ;                  22 /                   39 >=          */
/*   11 (                  23 *                   40 <=          */
/*   12 )                  30 &                    0 FIM DE      */
/*                                                  ARQUIVO      */
/* ALEM DESTES TEMOS TAMBEM AS PALAVRAS RESERVADAS QUE SERAO     */
/* ENUMERADAS A SEGUIR, COM OS RESPECTIVOS TIPOS :               */
/* TIPO PALAVRA    TIPO PALAVRA    TIPO PALAVRA    TIPO PALAVRA */
/*  99  ABS         70  ELSE        76  INTEGER     99  ROUND    */
/*  99  ARCTAN      67  END         51  LABEL       99  SIN      */
/*  57  ARRAY       99  EOF         99  LN          99  SQR      */
/*  56  BEGIN       99  EOLN        25  MOD         99  SQRT     */
/*  78  BOOLEAN     99  EOPG        65  NEW         99  SUCC     */
/*  64  CASE        99  EXP         99  ODD         69  THEN     */
/*  79  CHAR        58  OF          74  TO          99  CHR      */
/*  75  FOR         99  ORD         52  CONST       80  FORWARD  */
/*  99  PRED        99  TRUNC       99  COS         55  FUNCTION */
/*  54  PROC        63  UNTIL       24  DIV         66  GOTO     */
/*  50  PROGRAM     53  VAR         72  DO          68  IF       */
/*  77  REAL        71  WHILE       73  DOWNTO      91  READ     */
/*  92  READP       93  READD       94  READPD      41  IN       */
/*  60  REPEAT      81  WRITE       82  WRITELN     83  WRITEPG  */
/*  84  WRITED      85  WRITELND    86  WRITEPGD                 */
/* OBS. - NA TABELA ACIMA, TIPO=99 INDICA FUNCAO INTERNA QUE     */
/*        AINDA NAO FOI IMPLEMENTADA.                            */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                   */
/*   PLAPP04 - RECONHECEDOR LEXICO DO COMPILADOR.                */
/*   PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.   */
/*   PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIM-    */
```

```
/*          BOLOS - ENTRY-POINT DA PLAPP05.                          */
/*    PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES     */
/*              DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO          */
/*              ENCERRADA - ENTRY-POINT DA PLAPP05.                  */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                               */
/*    PLAPP09 - RECCNHECEDOR DE DECLARACAO DE CONSTANTES.            */
/*    PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.             */
/*    PLAPP11 - RECCNHECEDOR DE CABECALHO DE PROCEDURE.              */
/*    PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.                 */
/*    PLAPP13 - INSERE CONSTANTE NA TABELA DE CONSTANTES INTEIRAS    */
/*    PLAPP14 - INSERE CONSTANTE NA TABELA DE CONSTANTES REAIS -     */
/*              ENTRY-POINT DA PLAPP13.                              */
/*    PLAPP15 - INSERE CONSTANTE NA TABELA DE CONSTANTES ALFA-       */
/*              NUMERICAS - ENTRY-POINT DA PLAPP13.                  */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS.                            */
/*    PLAPP18 - RECCNHECEDOR DE TIPO DE VARIAVEIS.                   */
/*    PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1.                    */
/*    PLAPP20 - RECCNHECEDOR DE SECAO DE PARAMETROS FORMAIS.         */
/*    PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                    */
/*    PLAPP22 - RECCNHECEDOR DE COMANDO "WHILE".                     */
/*    PLAPP23 - RECONHECEDOR DE COMANDO "IF".                        */
/*    PLAPP24 - RECCNHECEDOR DE COMANDO "CASE".                      */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                       */
/*    PLAPP26 - RECCNHECEDOR DE COMANDO "READ".                      */
/*    PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                     */
/*    PLAPP28 - RECCNHECEDOR DE SECAO DE PARAMETROS REAIS.           */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS.                   */
/*    PLAPP30 - RECCNHECEDOR DE EXPRESSAO.                           */
/*    PLAPP34 - GRAVA REGISTRO DE INSTRUCOES EM "OBJ1".              */
/*******************************************************************/


DCL 1 AUXSCAN       EXTERNAL ,
      2 IDENT         CHAR(8) ,
      2 NINT          BIN FIXED(31,0) ,
      2 NREAL         BIN FLOAT(21) ,
      2 STRING(256)   CHAR(1) ,
      2 TAM           BIN FIXED(15,0) ,
      2 PTR           BIN FIXED(15,0) INIT(72) ,
      2 TIPO          BIN FIXED(15,0) ;


/*******************************************************************/
/* MACRO PLAMV02                                                   */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "BLOCO" E "BLOCMAX" :       */
/*   "BLOCO" E' A VARIAVEL QUE CONTEM O NUMERO DO BLOCO QUE ES-    */
/*   TA' SENDO ANALISADO E "BLOCMAX" INDICA O VALOR MAXIMO AL-     */
/*   CANCADO POR "BLOCO" (NUMERO MAXIMO DE PROCEDURES EMBUTI-      */
/*   DAS).                                                         */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                        */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                     */
/*   PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.     */
/*   PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIMBO-    */
/*             LOS - ENTRY-POINT DA PLAPP05.                       */
/*   PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES    */
/*             DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO EN-     */
/*             CERRADA - ENTRY-POINT DA PLAPP05.                   */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                              */
/*   PLAPP11 - RECCNHECEDOR DE CABECALHO DE PROCEDURE.             */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.                */
```

```
/*    PLAPP17 - RECONHECEDOR DE COMANDOS.                              */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.           */
/*    PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                        */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                         */
/*    PLAPP26 - RECONHECEDOR DE COMANDO "READ".                        */
/*    PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                       */
/*    PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.             */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                      */
/*    PLAPP30 - RECONHECEDOR DE EXPRESSOES.                            */
/*    PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                         */
/********************************************************************/

DCL BLOCO           EXTERNAL BIN FIXED(15,0) INIT(0) ;
DCL BLOCMAX         EXTERNAL BIN FIXED(15,0) INIT(0) ;


/********************************************************************/
/* MACRO PLAMV03                                                    */
/* DECLARACAO DA REFERENCIA EXTERNA "EOF" :                         */
/*   VARIAVEL USADA PARA INDICAR O ESTADO DO ARQUIVO QUE CONTEM     */
/*  O  PROGRAMA DO USUARIO. PODE ASSUMIR 2 VALORES :                */
/*     '0'B - ARQUIVO AINDA NAO TERMINOU                            */
/*     '1'B - ARQUIVO JA' TERMINOU                                  */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                         */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                      */
/*   PLAPP04 - RECONHECEDOR LEXICO DO COMPILADOR.                   */
/********************************************************************/


DCL EOF             EXTERNAL BIT(1) INIT('0'B) ;


/********************************************************************/
/* MACRO PLAMV04                                                    */
/* DECLARACAO DA REFERENCIA EXTERNA "TABSIMB" :                     */
/*   ESTA ESTRUTURA CONTEM A TABELA DE SIMBOLOS E VARIAVEIS         */
/*   AUXILIARES UTILIZADAS PARA SUA MANIPULACAO.                    */
/*   "CABEC" E' O VETOR CABECA DE HASH.                             */
/*   "TABELA" E' A TABELA DE SIMBOLOS PROPRIAMENTE DITA:            */
/*     "NOME" CONTEM O IDENTIFICADOR;                               */
/*     "IDTIPO" - TIPO DO IDENTIFICADOR, CONFORME TABELA ABAIXO:    */
/*  IDTIPO      TIPO DE IDENTIFICADOR          PTCTE APONTA PARA:   */
/*  -1       IDENTIFICADOR NAO DECLARADO                            */
/*   0       CONSTANTE INDEFINIDA                                   */
/*   1       CONSTANTE INTEIRA                    TABINT            */
/*   2       CONSTANTE REAL                       TABREAL           */
/*   3       CONSTANTE BOOLEANA                VALOR DA CTE         */
/*   4       LITERAL                              TABALFA           */
/*  10       PROCEDURE                            TABPARM           */
/*  20       FUNCAO INDEFINIDA                    TABPARM           */
/*  21       FUNCAO INTEIRA                       TABPARM           */
/*  22       FUNCAO REAL                          TABPARM           */
/*  23       FUNCAO BOOLEANA                      TABPARM           */
/*  24       FUNCAO CHAR                          TABPARM           */
/*  30       LABEL                                                  */
/*  40       VARIAVEL SIMPLES INDEFINIDA                            */
/*  41       VARIAVEL SIMPLES INTEIRA                               */
/*  42       VARIAVEL SIMPLES REAL                                  */
/*  43       VARIAVEL SIMPLES BOOLEANA                              */
```

```
/*  44      VARIAVEL SIMPLES CHAR                                   */
/*  50      VARIAVEL INDEXADA INDEFINIDA                TABDIM      */
/*  51      VARIAVEL INDEXADA INTEIRA                   TABDIM      */
/*  52      VARIAVEL INDEXADA REAL                      TABDIM      */
/*  53      VARIAVEL INDEXADA BOOLEANA                  TABDIM      */
/*  54      VARIAVEL INDEXADA CHAR                      TABDIM      */
/*  60      PARAMETRO PROCEDURE                                     */
/*  70      PARAMETRO FUNCAO INDEFINIDA                             */
/*  71      PARAMETRO FUNCAO INTEIRA                                */
/*  72      PARAMETRO FUNCAO REAL                                   */
/*  73      PARAMETRO FUNCAO BOOLEANA                               */
/*  74      PARAMETRO FUNCAO CHAR                                   */
/*  80      PARAMETRO VAR. SIMPLES INDEFINIDA                       */
/*  81      PARAMETRO VAR. SIMPLES INTEIRA                          */
/*  82      PARAMETRO VAR. SIMPLES REAL                             */
/*  83      PARAMETRC VAR. SIMPLES BOOLEANA                         */
/*  84      PARAMETRO VAR. SIMPLES CHAR                             */
/*  90      PAR. VAR. SIMPLES INDEFINIDA POR VALOR                  */
/*  91      PAR. VAR. SIMPLES INTEIRA POR VALOR                     */
/*  92      PAR. VAR. SIMPLES REAL POR VALOR                        */
/*  93      PAR. VAR. SIMPLES BOOLEANA POR VALOR                    */
/*  94      PAR. VAR. SIMPLES CHAR POR VALOR                        */
/*  100     PAR. VAR. INDEXADA INDEFINIDA               TABDIM      */
/*  101     PAR. VAR. INDEXADA INTEIRA                  TABDIM      */
/*  102     PAR. VAR. INDEXADA REAL                     TABDIM      */
/*  103     PAR. VAR. INDEXADA BOOLEANA                 TABDIM      */
/*  104     PAR. VAR. INDEXADA CHAR                     TABDIM      */
/*  110     PAR. VAR. INDEXADA INDEFINIDA POR VALOR     TABDIM      */
/*  111     PAR. VAR. INDEXADA INTEIRA POR VALOR        TABDIM      */
/*  112     PAR. VAR. INDEXADA REAL POR VALOR           TABDIM      */
/*  113     PAR. VAR. INDEXADA BOOLEANA POR VALOR       TABDIM      */
/*  114     PAR. VAR. INDEXADA CHAR POR VALOR           TABDIM      */
/*     "PTCTE" - APONTADOR PARA TABELAS: DE CONSTANTES, DE DI-       */
/*               MENSOES DE ARRAY OU DE PARAMETROS FORMAIS. NO       */
/*               CASO DO IDENTIFICADOR SER UM LABEL, ESTE CAMPO      */
/*               E' UTILIZADO PARA DETETAR SUA EXISTENCIA COMO       */
/*               LABEL DE MAIS DE UM COMANDO NO MESMO BLOCO E,       */
/*               DEVIDO A EXISTENCIA DE DESVIO A FRENTE (GOTO        */
/*               PARA COMANDO A FRENTE), PARA INDICAR REFEREN-       */
/*               CIA AO LABEL ANTES DE SUA DEFINICAO.                */
/*               INICIALMENTE CONTEM 0, E' FEITO IGUAL A -1 NA       */
/*               PRIMEIRA REFERENCIA ANTERIOR A SUA DEFINICAO,       */
/*               ISTO E', ANTES DE APARECER COMO LABEL DE UM         */
/*               COMANDO, E E' FEITO IGUAL A 1 NA PRIMEIRA           */
/*               UTILIZACAO DO IDENTIFICADOR. NO CASO DO IDEN-       */
/*               TIFICADOR SER UMA CONSTANTE BOOLEANA, CONTEM        */
/*               O VALOR DA CONSTANTE.                               */
/*     "NIVEL" - NUMERO DO BLOCO CORRENTE QUANDO DA INSERCAO DO      */
/*               IDENTIFICADOR;                                      */
/*     "LINK" - APONTADOR PARA ENTRADA ANTERIOR COM O MESMO          */
/*              HOME-ADDRESS.                                        */
/*     "ENDER"  - ENDERECO DO IDENTIFICADOR.                         */
/*   "IDNOME" - IDENTIFICADOR A SER INSERIDO OU PROCURADO.           */
/*   "IND" - POSICAO DO IDENTIFICADOR (INSERIDO OU ENCONTRADO)       */
/*           NA TABELA.                                              */
/*   "ACHOU" - INDICA SE O IDENTIFICADOR EXISTIA ('1'B) OU NAO       */
/*             ('0'B) NA TABELA ANTES DA BUSCA OU INSERCAO.          */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                       */
/*   PLAPP02 - IMPRIME TABELA DE SIMBOLOS, PARA DEPURACAO.           */
```

```
/*    PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.   */
/*    PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIMBO- */
/*              LOS - ENTRY-POINT DA PLAPP05.                    */
/*    PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES */
/*              DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO EN-  */
/*              CERRADA - ENTRY-POINT DA PLAPP05.                */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                           */
/*    PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.        */
/*    PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.         */
/*    PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.          */
/*    PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.             */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS.                        */
/*    PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1.                */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.     */
/*    PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                  */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                   */
/*    PLAPP26 - RECONHECEDOR DE COMANDO "READ".                  */
/*    PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.       */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                */
/*    PLAPP30 - RECONHECEDOR DE EXPRESSAO.                       */
/*****************************************************************/


DCL 1 TABSIMB      EXTERNAL ,
      2 CABEC(0:63)        BIN FIXED(15,0) INIT((64) 0) ,
      2 TABELA(300) ,
        3 NOME               CHAR(8) ,
        3 IDTIPO             BIN FIXED(15,0) ,
        3 PTCTE              BIN FIXED(15,0) ,
        3 NIVEL              BIN FIXED(15,0) ,
        3 LINK               BIN FIXED(15,0) ,
        3 ENDER              BIN FIXED(15,0) ,
      2 IDNOME             CHAR(8) ,          /*  VARIAVEIS    */
      2 IND                BIN FIXED(15,0) , /*               */
      2 ACHOU              BIT(1) ;           /*  AUXILIARES   */


/*****************************************************************/
/* MACRO PLAMV05                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA "TRACE" :                    */
/*   VETOR DE FLAGS UTILIZADOS PARA DEPURACAO.                   */
/*     TRACE(1)   - USADO PARA DAR O FLUXO E IMPRESSAO DE PARA-  */
/*                  METROS.                                      */
/*     TRACE(2)   - USADO PELA PROCEDURE PLAPP05 (BUSCA) E SEUS  */
/*                  ENTRY-POINT'S PARA IMPRIMIR A TABELA DE SIM- */
/*                  BOLOS E O VALOR DA FUNCAO HASH.              */
/*     TRACE(3)   - USADO POR PLAPP30 (REC.EXPRESSAO) PARA IM-   */
/*                  PRIMIR A PILHA USADA NA ANALISE DA EXPRESSAO.*/
/*     TRACE(4)   - USADO POR PLAPP00 (PROGRAMA PRINCIPAL) PARA  */
/*                  INIBIR A EXECUCAO DO PROGRAMA COMPILADO.     */
/*     TRACE(5)   - USADO POR PLAPP32 (EXECUTA PROGRAMA COMPI-   */
/*                  LADO) PARA IMPRIMIR O PROGRAMA DEPOIS DE     */
/*                  CARREGADO NA MEMORIA (AREAS DE VARIAVEIS E   */
/*                  CONSTANTES E INSTRUCOES).                    */
/*     TRACE(6)   - AINDA NAO ESTA' SENDO UTILIZADO.             */
/*     TRACE(7)   - AINDA NAO ESTA' SENDO UTILIZADO.             */
/*     TRACE(8)   - AINDA NAO ESTA' SENDO UTILIZADO.             */
/*     TRACE(9)   - AINDA NAO ESTA' SENDO UTILIZADO.             */
/*     TRACE(10)  - AINDA NAO ESTA' SENDO UTILIZADO.             */
/*     TRACE(11)  - AINDA NAO ESTA' SENDO UTILIZADO.             */
```

```
/*      TRACE(12) - AINDA NAO ESTA' SENDO UTILIZADO.                    */
/*      TRACE(13) - AINDA NAO ESTA' SENDO UTILIZADO.                    */
/*      TRACE(14) - AINDA NAO ESTA' SENDO UTILIZADO.                    */
/*      TRACE(15) - AINDA NAO ESTA' SENDO UTILIZADO.                    */
/*      TRACE(16) - AINDA NAO ESTA' SENDO UTILIZADO.                    */
/*  ESTA MACRO E' UTILIZADA NAS PROCEDURES :                            */
/*    PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                         */
/*    PLAPP03 - RECONHECEDOR DAS OPCOES DE COMPILACAO.                  */
/*    PLAPP04 - RECONHECEDOR LEXICO DO COMPILADOR.                      */
/*    PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.         */
/*    PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIM-          */
/*              BOLOS - ENTRY-POINT DA PLAPP05.                         */
/*    PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES        */
/*              DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO             */
/*              ENCERRADA - ENTRY-POINT DA PLAPP05.                     */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                                  */
/*    PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.               */
/*    PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.                */
/*    PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.                 */
/*    PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.                    */
/*    PLAPP13 - INSERE CONSTANTE NA TABELA DE CONSTANTES INTEIRAS       */
/*    PLAPP14 - INSERE CONSTANTE NA TABELA DE CONSTANTES REAIS -        */
/*              ENTRY-POINT DA PLAPP13.                                 */
/*    PLAPP15 - INSERE CONSTANTE NA TABELA DE CONSTANTES ALFANU-        */
/*              MERICAS - ENTRY-POINT DA PLAPP13.                       */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS.                               */
/*    PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS.                      */
/*    PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1.                       */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.            */
/*    PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                       */
/*    PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                        */
/*    PLAPP23 - RECONHECEDOR DE COMANDO "IF".                           */
/*    PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                         */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                          */
/*    PLAPP26 - RECONHECEDOR DE COMANDO "READ".                         */
/*    PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                        */
/*    PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.              */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS.                      */
/*    PLAPP30 - RECONHECEDOR DE EXPRESSAO.                              */
/*    PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                          */
/*    PLAPP32 - EXECUTA PROGRAMA COMPILADO.                             */
/*********************************************************************/


DCL TRACE(16)      BIT(1) EXTERNAL INIT((16) (1) '0'B) ;


/*********************************************************************/
/* MACRO PLAMV06                                                     */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "ERRO" E "COLUNA" :           */
/*   ESTAS VARIAVEIS SAO UTILIZADAS PARA COMUNICACAO ENTRE AS        */
/*   DIVERSAS PARTES DO COMPILADOR E A PROCEDURE PLAPP01.            */
/*     ERRO - CONTEM O CODIGO DO ERRO                                */
/*     COLUNA - CONTEM A COLUNA (APROXIMADA) DO CARTAO ONDE FOI      */
/*              DETETADO O ERRO                                      */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                       */
/*   PLAPP01 - ROTINA DE IMPRESSAO DE ERROS DE COMPILACAO.           */
/*   PLAPP03 - RECONHECEDOR DAS OPCOES DE COMPILACAO.                */
/*   PLAPP04 - RECONHECEDOR LEXICO DO COMPILADOR.                    */
```

```
/*    PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.   */
/*    PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TABELA DE SIM-    */
/*              BOLOS  - ENTRY-POINT DA PLAPP05.                  */
/*    PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES  */
/*              DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO       */
/*              ENCERRADA - ENTRY-POINT DA PLAPP05.               */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                            */
/*    PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.         */
/*    PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.          */
/*    PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.           */
/*    PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.              */
/*    PLAPP13 - INSERE CONSTANTE NA TABELA DE CONSTANTES INTEIRAS */
/*    PLAPP14 - INSERE CONSTANTE NA TABELA DE CONSTANTES REAIS -  */
/*              ENTRY-POINT DA PLAPP13.                           */
/*    PLAPP15 - INSERE CONSTANTE NA TABELA DE CONSTANTES ALFA-    */
/*              NUMERICAS - ENTRY-POINT DA PLAPP13.               */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS.                         */
/*    PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS.                */
/*    PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1.                 */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*    PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                 */
/*    PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                  */
/*    PLAPP23 - RECONHECEDOR DE COMANDO "IF".                     */
/*    PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                   */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                    */
/*    PLAPP26 - RECONHECEDOR DE COMANDO "READ".                   */
/*    PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                  */
/*    PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.        */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS.                */
/*    PLAPP30 - RECONHECEDOR DE EXPRESSAO.                        */
/*    PLAPP34 - GRAVA REGISTRO DE INSTRUCOES EM "OBJ1".           */
/*****************************************************************/

DCL ERRO            BIN FIXED(15,0)  EXTERNAL ;
DCL COLUNA          BIN FIXED(15,0)  EXTERNAL ;


/*****************************************************************/
/* MACRO PLAMV07                                                 */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "TABINT", "TABREAL",      */
/*   "TABALFA", "ENDZERO", "ENDUM", "EFALSE", "ETRUE", "ENDCTE"  */
/*   E "POSCTE".                                                 */
/*   TABINT, TABREAL E TABALFA SAO AS TABELAS UTILIZADAS PARA    */
/*   ARMAZENAR AS CONSTANTES (INTEIRAS, REAIS E ALFANUMERICAS,   */
/*   RESPECTIVAMENTE) DO PROGRAMA DO USUARIO.                    */
/*   NAS RESPECTIVAS TABELAS:                                    */
/*     - "VALINT", "VALREAL" E "TEXTO" CONTEM A CONSTANTE;       */
/*     - "PTINT" E "PTREAL" SAO APONTADORES PARA MANTER A        */
/*       ORDENACAO ASCENDENTE (PTINT(0) E PTREAL(0) SAO OS       */
/*       CABECA DE LISTA);                                       */
/*     - "PROXINT" E "PROXRE" APONTAM PARA A PROXIMA ENTRADA     */
/*       DISPONIVEL, E "ULTALFA" PARA A ULTIMA OCUPADA;          */
/*     - AS VARIAVEIS PREFIXADAS POR "LSK" CONTEM O LIMITE COR-  */
/*       RENTE INFERIOR DE CADA TABELA. OS SUFIXOS "INT", "RE"   */
/*       E "CH" INDICAM RESPECTIVAMENTE TABELA DE CONSTANTES     */
/*       INTEIRAS, REAIS E ALFANUMERICAS. A VARIAVEL "LSKCHT"    */
/*       REFERE-SE AO VETOR "TEXTO", DE "TABALFA".               */
/*     - "CTEINT", "CTEREAL" E "CTELIT" CONTEM A CONSTANTE A     */
/*       SER PROCURADA OU INSERIDA. E NO ULTIMO CASO, "TAMLIT"   */
```

```
/*       CONTEM O TAMANHO DO LITERAL;                                  */
/*    "PTALFA" INDICA A POSICAO INICIAL DO LITERAL DENTRO DE           */
/*    "TEXTO" E "TAMALFA" DA' O TAMANHO DO LITERAL.                    */
/*    ALEM DA POSICAO 0 DE "TABINT" QUE SERVE DE CABECA DE LISTA,      */
/*    AS POSICOES DE 1 A 10 TAMBEM TEM SIGNIFICADO ESPECIAL.           */
/*    AS POSICOES 1 E 2 CONTEM AS CONSTANTES 0 E 1, RESPECTIVA-        */
/*    MENTE, QUE JA' SE ENCONTRAM EM "TABINT" DEVIDO A UTILIZA-        */
/*    CAO FREQUENTE.                                                   */
/*    AS POSICOES 3 A 6 SAO RESERVADAS PARA O VALOR A SER USADO,       */
/*    EM FASE DE EXECUCAO, COMO DESLOCAMENTO DAS AREAS DE VARIA-       */
/*    VEIS (INTEIRAS, REAIS, BOOLEANAS E CHAR, RESPECTIVAMENTE);       */
/*    AS POSICOES DE 7 A 10 CORRESPONDEM AOS DESLOCAMENTOS DAS         */
/*    AREAS DE TEMPORARIOS. ESTAS 8 POSICOES NAO FAZEM PARTE DO        */
/*    ENCADEAMENTO DA LISTA, POIS AS CONSTANTES PROPRIAMENTE DI-       */
/*    TAS SO' SAO GERADAS AO FINAL DA COMPILACAO. PORISSO TAMBEM,      */
/*    "PROXINT" COMECA COM VALOR 11.                                   */
/*    "ENDZERO" E' O ENDERECO DA CONSTANTE INTEIRA ZERO, E             */
/*    "ENDUM" O ENDERECO DA CONSTANTE INTEIRA UM.                      */
/*    "EFALSE" E' O ENDERECO DA CONSTANTE BOOLEANA "FALSE".            */
/*    "ETRUE" E' O ENDERECO DA CONSTANTE BOOLEANA "TRUE".              */
/*    "ENDCTE" E' O ENDERECO DA CONSTANTE QUE FOI INSERIDA.            */
/*    EM TODOS OS CASOS, "POSCTE" RETORNA COM A POSICAO DA             */
/*    CONSTANTE NA TABELA.                                             */
/*    OBS: AO AUMENTAR O TAMANHO DESTAS TABELAS DEVEM-SE ATUALI-       */
/*    ZAR TAMBEM AS VARIAVEIS "LSKINT" , "LSKRE" , "LSKCH" E           */
/*    "LSKCHT", E SEUS VALORES NAO PODEM ULTRAPASSAR A 999 SEM         */
/*    ALTERACAO DO VALOR INICIAL USADO PARA GERACAO DE ENDERECO        */
/*    PARA VARIAVEL (VER MACRO "PLAMV11").                             */
/*  ESTA MACRO E' UTILIZADA NAS PROCEDURES :                           */
/*    PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                        */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                                 */
/*    PLAPP09 - RECCNHECEDOR DE DECLARACAO DE CONSTANTES.              */
/*    PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.                */
/*    PLAPP12 - RECCNHECEDOR DE CABECALHO DE FUNCAO.                   */
/*    PLAPP13 - INSERE CONSTANTE NA TABELA DE CTES INTEIRAS.           */
/*    PLAPP14 - INSERE CONSTANTE NA TABELA DE CONSTANTES REAIS -       */
/*              ENTRY-POINT DA PLAPP13.                                */
/*    PLAPP15 - INSERE CONSTANTE NA TABELA DE CONSTANTES ALFA-         */
/*              NUMERICAS - ENTRY-POINT DA PLAPP13.                    */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS.                              */
/*    PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEL.                      */
/*    PLAPP19 - RECCNHECEDOR DE EXPRESSAO TIPO 1.                      */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.           */
/*    PLAPP21 - RECCNHECEDOR DE COMANDO "REPEAT".                      */
/*    PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                       */
/*    PLAPP23 - RECCNHECEDOR DE COMANDO "IF".                          */
/*    PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                        */
/*    PLAPP25 - RECCNHECEDOR DE COMANDO "FOR".                         */
/*    PLAPP26 - RECONHECEDOR DE COMANDO "READ".                        */
/*    PLAPP27 - RECCNHECEDOR DE COMANDO "WRITE".                       */
/*    PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.             */
/*    PLAPP29 - RECCNHECEDOR DE INDICES DE ARRAY.                      */
/*    PLAPP30 - RECONHECEDOR DE EXPRESSAO.                             */
/*    PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                         */
/*******************************************************************/
DCL 1 TABINT          EXTERNAL ,
      2 TABIN(0 : 400) ,
        3 VALINT          BIN FIXED(31,0) INIT(0,0,1) ,
        3 PTINT           BIN FIXED(15,0) INIT(1,2,0) ,
      2 PROXINT         BIN FIXED(15,0) INIT(11) ,
```

```
      2 LSKINT          BIN FIXED(15,0) INIT(400) ,
      2 CTEINT          BIN FIXED(31,0) ;
DCL 1 TABREAL         EXTERNAL ,
      2 TABRE(0 : 64) ,
        3 VALREAL         BIN FLOAT(21) ,
        3 PTREAL          BIN FIXED(15,0) INIT(0) ,
      2 PROXRE          BIN FIXED(15,0) INIT(1) ,
      2 LSKRE           BIN FIXED(15,0) INIT(64) ,
      2 CTEREAL         BIN FLOAT(21) ;
DCL 1 TABALFA         EXTERNAL ,
      2 TABALF(0 : 100) ,
        3 PTALFA          BIN FIXED(15,0) INIT(1) ,
        3 TAMALFA         BIN FIXED(15,0) INIT(0) ,
      2 TEXTO(1 : 999)      CHAR(1) ,
      2 ULTALFA         BIN FIXED(15,0) INIT(0) ,
      2 LSKCH           BIN FIXED(15,0) INIT(100) ,
      2 LSKCHT          BIN FIXED(15,0) INIT(999) ,
      2 CTELIT(256)     CHAR(1) ,
      2 TAMLIT          BIN FIXED(15,0) ;
DCL ENDZERO         BIN FIXED(15,0) EXTERNAL INIT(1) ;
DCL ENDUM           BIN FIXED(15,0) EXTERNAL INIT(2) ;
DCL EFALSE          BIN FIXED(15,0) EXTERNAL INIT(1) ;
DCL ETRUE           BIN FIXED(15,0) EXTERNAL INIT(2) ;
DCL ENDCTE          BIN FIXED(15,0) EXTERNAL ;
DCL POSCTE          BIN FIXED(15,0) EXTERNAL ;


/*****************************************************************/
/* MACRO PLAMV08                                                 */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "TABDIM" E "PROXDIM" :    */
/*   "TABDIM" E' O VETOR UTILIZADO PARA GUARDAR AS DIMENSOES     */
/*   DAS VARIAVEIS INDEXADAS DECLARADAS NO PROGRAMA DO USUARIO.  */
/*   PARA CADA ARRAY DO PROGRAMA FONTE, DURANTE A ANALISE DA     */
/*   DECLARACAO, SAO COLOCADOS NESTA TABELA: O NUMERO DE DIMEN-  */
/*   SOES DO ARRAY; O NUMERO DE ELEMENTOS; UM PONTEIRO PARA A    */
/*   TABELA DE CONSTANTES INTEIRAS ("TABINT", ONDE ESTAO GUAR-   */
/*   DADOS, ALEM DO NUMERO DE ELEMENTOS DO ARRAY, AS CONSTANTES  */
/*   NECESSARIAS PARA CALCULO DO ENDERECO DE CADA ELEMENTO, EM   */
/*   FASE DE EXECUCAO DO PROGRAMA DO USUARIO, QUANDO  HOUVER     */
/*   REFERENCIA AO ELEMENTO) E OS LIMITES INFERIOR E SUPERIOR    */
/*   DE CADA DIMENSAO DO ARRAY. PORTANTO, PARA CADA ARRAY SAO    */
/*   OCUPADAS 3 + 2*(NUM. DE DIMENSOES) ENTRADAS NA TABELA, NA   */
/*   ORDEM CITADA ACIMA.                                         */
/*   "MAXDIM" E' O NUMERO MAXIMO DE ENTRADAS EM "TABDIM".        */
/*   "PROXDIM" APONTA PARA A PROXIMA ENTRADA DISPONIVEL DE       */
/*   "TABDIM".                                                   */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                            */
/*   PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.          */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                         */
/*   PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS.                */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                   */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                  */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                 */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                        */
/*****************************************************************/


DCL TABDIM(128)     BIN FIXED(15,0) EXTERNAL ;
```

```
DCL MAXDIM          BIN FIXED(15,0) EXTERNAL INIT(128) ;
DCL PROXDIM         BIN FIXED(15,0) EXTERNAL INIT(1) ;

/*****************************************************************/
/* MACRO PLAMV09                                                 */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "TABPARM" E "PRXPARM" :   */
/*   "TABPARM" E' O VETOR UTILIZADO PARA GUARDAR A LISTA DE PA-  */
/*   RAMETROS FORMAIS DAS PROCEDURES E FUNCOES DECLARADAS NO     */
/*   PROGRAMA DO USUARIO. PARA CADA PROCEDURE OU FUNCAO TEMOS    */
/*   EM "TABPARM", NESTA ORDEM: O ENDERECO DA AREA DE LIGACAO,   */
/*   O NUMERO DE PARAMETROS, E O TIPO DE CADA PARAMETRO. NO CA-  */
/*   SO DE UM PARAMETRO ARRAY, TEMOS,ALEM DO TIPO, O NUMERO DE   */
/*   DIMENSOES, O NUMERO DE ELEMENTOS E OS LIMITES INFERIOR E    */
/*   SUPERIOR DE CADA DIMENSAO DO ARRAY.                         */
/*                                                               */
/*   CODIGO DOS PARAMETROS:                                      */
/*    TABPARM       TIPO DO PARAMETRO                          */
/*      60        PARAMETRO PROCEDURE                          */
/*      70        PARAMETRO FUNCAO INDEFINIDA                  */
/*      71        PARAMETRO FUNCAO INTEIRA                     */
/*      72        PARAMETRO FUNCAO REAL                        */
/*      73        PARAMETRO FUNCAO BOOLEANA                    */
/*      74        PARAMETRO FUNCAO CHAR                        */
/*      80        PARAMETRO VAR. SIMPLES INDEFINIDA            */
/*      81        PARAMETRO VAR. SIMPLES INTEIRA               */
/*      82        PARAMETRO VAR. SIMPLES REAL                  */
/*      83        PARAMETRO VAR. SIMPLES BOOLEANA              */
/*      84        PARAMETRO VAR. SIMPLES CHAR                  */
/*      90        PAR. VAR. SIMPLES INDEFINIDA POR VALOR       */
/*      91        PAR. VAR. SIMPLES INTEIRA POR VALOR          */
/*      92        PAR. VAR. SIMPLES REAL POR VALOR             */
/*      93        PAR. VAR. SIMPLES BOOLEANA POR VALOR         */
/*      94        PAR. VAR. SIMPLES CHAR POR VALOR             */
/*     100        PAR. VAR. INDEXADA INDEFINIDA                */
/*     101        PAR. VAR. INDEXADA INTEIRA                   */
/*     102        PAR. VAR. INDEXADA REAL                      */
/*     103        PAR. VAR. INDEXADA BOOLEANA                  */
/*     104        PAR. VAR. INDEXADA CHAR                      */
/*     110        PAR. VAR. INDEXADA INDEFINIDA POR VALOR      */
/*     111        PAR. VAR. INDEXADA INTEIRA POR VALOR         */
/*     112        PAR. VAR. INDEXADA REAL POR VALOR            */
/*     113        PAR. VAR. INDEXADA BOOLEANA POR VALOR        */
/*     114        PAR. VAR. INDEXADA CHAR POR VALOR            */
/*   "LIMPARM" E' O NUMERO MAXIMO DE ENTRADAS EM "TABPARM".     */
/*   "PRXPARM" APONTA PARA A PROXIMA ENTRADA DISPONIVEL DE      */
/*   "TABPARM".                                                  */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                            */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.           */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.              */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                         */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.        */
/*****************************************************************/

DCL TABPARM(128)    BIN FIXED(15,0) EXTERNAL ;
DCL LIMPARM         BIN FIXED(15,0) EXTERNAL INIT (128) ;
DCL PRXPARM         BIN FIXED(15,0) EXTERNAL INIT(1) ;
```

```
/*****************************************************************/
/* MACRO PLAMV10                                                 */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "TIPEXP", "PTMAT" E       */
/*   "TAMCHAR".                                                  */
/*                                                               */
/*   "TIPEXP" E' USADA PARA O RECONHECEDOR DE EXPRESSOES         */
/*   (PLAPP30), DEVOLVER O CODIGO DO TIPO DA EXPRESSAO ANALI-    */
/*   SADA, CONFORME TABELA ABAIXO.                               */
/*   SE "TIPEXP" >= 100 (EXPRESSAO MATRICIAL), "PTMAT" APONTA-   */
/*   RA' PARA A DESCRICAO DAS DIMENSOES DE ARRAY EM "TABDIM".    */
/*   "PTMAT" E' USADO NO RECONHECEDOR DE COMANDO DE ATRIBUICAO    */
/*   (NO PLAPP17), PARA VERIFICAR A COERENCIA DAS DIMENSOES      */
/*   DOS ARRAYS.                                                 */
/*   ALEM DISSO, SE "TIPEXP" = 4 OU "TIPEXP" = 104 (EXPRESSAO    */
/*   MATRICIAL OU NAO, DO TIPO CHAR), O RECONHECEDOR DE EXPRES-  */
/*   SOES (PLAPP30) DEVOLVERA' EM "TAMCHAR" O NUMERO DE CARAC-   */
/*   TERES DE CADA ELEMENTO DA EXPRESSAO. CASO CONTRARIO         */
/*   "TAMCHAR" TERA' O VALOR 1.                                  */
/*                                                               */
/*   TIPEXP     EXPRESSAO RESULTANTE                             */
/*                                                               */
/*    -1        DE TIPO INDEFINIDO                               */
/*                                                               */
/*     0        NAO MATRICIAL INDEFINIDA                         */
/*     1        NAO MATRICIAL INTEIRA                            */
/*     2        NAO MATRICIAL REAL                               */
/*     3        NAO MATRICIAL BOOLEANA                           */
/*     4        NAO MATRICIAL CHAR                               */
/*                                                               */
/*    50        MATRICIAL INDEFINIDA                             */
/*    51        MATRICIAL INTEIRA                                */
/*    52        MATRICIAL REAL                                   */
/*    53        MATRICIAL BOOLEANA                               */
/*    54        MATRICIAL CHAR                                   */
/*                                                               */
/*   100        MATRICIAL (PARAMETRO POR ENDERECO) INDEFINIDA     */
/*   101        MATRICIAL (PARAMETRO POR ENDERECO) INTEIRA        */
/*   102        MATRICIAL (PARAMETRO POR ENDERECO) REAL           */
/*   103        MATRICIAL (PARAMETRO POR ENDERECO) BOOLEANA       */
/*   104        MATRICIAL (PARAMETRO POR ENDEREÇO) CHAR           */
/*                                                               */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS                          */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT"                  */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE"                   */
/*   PLAPP23 - RECCNHECEDOR DE COMANDO "IF"                      */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE"                    */
/*   PLAPP25 - RECCNHECEDOR DE COMANDO "FOR"                     */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                  */
/*   PLAPP28 - RECCNHECEDOR DE PARAMETROS REAIS                  */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS                 */
/*   PLAPP30 - RECCNHECEDOR DE EXPRESSOES                        */
/*****************************************************************/

DCL TIPEXP          BIN FIXED(15,0) EXTERNAL ;
DCL PTMAT           BIN FIXED(15,0) EXTERNAL ;
DCL TAMCHAR         BIN FIXED(15,0) EXTERNAL ;
```

```
/*****************************************************************/
/* MACRO PLAMV11                                                 */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "GERA", "AUXGERA"         */
/*   A VARIAVEL "GERA" E' USADA PARA INIBIR (= '0'B) OU NAO      */
/*   (= '1'B) A GERACAO DE CODIGO DEVIDO A ERRO GRAVE NO PROGRA- */
/*   MA SENDO COMPILADO.                                         */
/*   A ESTRUTURA "AUXGERA" CONTEM VARIAVEIS AUXILIARES PARA GE-  */
/*   RACAO DE CODIGO.                                            */
/*   A SUBESTRUTURA "GERAEND" CONTEM AS VARIAVEIS USADAS PARA    */
/*   O MAPEAMENTO DAS AREAS DE MEMORIA DO PROGRAMA COMPILADO     */
/*   (EM FASE DE EXECUCAO DESTE PROGRAMA). ESTAS VARIAVEIS CON-  */
/*   TROLAM A ALOCACAO DE ENDERECOS PARA AS VARIAVEIS DO PROGRA- */
/*   MA ("CNTINT", "CNTREAL", "CNTBOOL" E "CNTCHAR"), VARIAVEIS  */
/*   TEMPORARIAS GERADAS PELO COMPILADOR ("TEMINT", "TEMREAL",   */
/*   "TEMBOOL" E "TEMCHAR") E INSTRUCOES ("CINST").              */
/*   AS VARIAVEIS PREFIXADAS POR "LIV" INDICAM O LIMITE INFERIOR */
/*   PARA OS ENDERECOS ATRIBUIDOS DURANTE A COMPILACAO AS VA-    */
/*   RIAVEIS DO PROGRAMA SENDO COMPILADO.                        */
/*   OS SUFIXOS INT, REAL, BOOL E CHAR INDICAM TRATAR-SE DE VA-  */
/*   RIAVEIS OU TEMPORARIOS INTEIROS, REAIS, BOOLEANOS OU CARA-  */
/*   TER, RESPECTIVAMENTE.                                       */
/*   'EUOPND' CONTEM O ENDERECO DO ULTIMO OPERANDO PARA O QUAL   */
/*   FOI GERADO CODIGO, 'TUOPND' CONTEM O TIPO DO ENDERECO       */
/*   DESTE OPERANDO (CONFORME OS VALORES DE 'TOPND' EXISTENTES   */
/*   NA MACRO PLAMV15) E 'XUOPND' CONTEM O ENDERECO DA CONSTANTE */
/*   ZERO SE O ENDERECO DO ULTIMO OPERANDO FOR DIRETO, OU O      */
/*   ENDERECO DO TEMPORARIO QUE CONTEM O DESLOCAMENTO CALCULADO  */
/*   PARA ESTE OPERANDO, EM CASO CONTRARIO.                      */
/*   "BUOPND" E "BXUOPND" CONTEM OS NUMEROS DOS REGISTROS BASE   */
/*   ASSOCIADOS A "EUOPND" E "XUOPND", RESPECTIVAMENTE.          */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                   */
/*   PLAPP01 - IMPRIME MENSAGENS DE ERRO.                        */
/*   PLAPP08 - RECONHECEDOR DE BLOCOS.                           */
/*   PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.          */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.           */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.              */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                         */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                 */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                  */
/*   PLAPP23 - RECONHECEDOR DE COMANDO "IF".                     */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                   */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                    */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                   */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                  */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.        */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                 */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                        */
/*   PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                    */
/*   PLAPP32 - EXECUTA PROGRAMA COMPILADO.                       */
/*   PLAPP34 - GRAVA REGISTRO DE INSTRUCOES EM 'OBJ1'.           */
/*****************************************************************/

DCL GERA            BIT(1) EXTERNAL INIT('1'B) ;
DCL 1 AUXGERA EXTERNAL ,
      2 GERAEND ,
        3 CNTINT          BIN FIXED (15,0) ,
        3 LIVINT          BIN FIXED (15,0)  INIT(1000) ,
```

```
        3  CNTREAL          BIN FIXED (15,0) ,
        3  LIVREAL          BIN FIXED (15,0)  INIT(1000) ,
        3  CNTBOOL          BIN FIXED (15,0) ,
        3  LIVBOOL          BIN FIXED (15,0)  INIT(1000) ,
        3  CNTCHAR          BIN FIXED (15,0) ,
        3  LIVCHAR          BIN FIXED (15,0)  INIT(1000) ,
        3  TEMINT           BIN FIXED (15,0)  ,
        3  TEMREAL          BIN FIXED (15,0)  ,
        3  TEMBOOL          BIN FIXED (15,0)  ,
        3  TEMCHAR          BIN FIXED (15,0)  ,
        3  CINST            BIN FIXED (15,0)  INIT(0) ,
      2 ULTOPND ,
        3  EUOPND           BIN FIXED (15,0) ,
        3  TUOPND           BIN FIXED (15,0) ,
        3  XUOPND           BIN FIXED (15,0) ,
        3  BUOPND           BIN FIXED(15,0) ,
        3  BXUOPND          BIN FIXED(15,0) ;


/*****************************************************************/
/* MACRO PLAMV12                                                 */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "EXEC" E "LIST".          */
/*   A VARIAVEL "EXEC" E' USADA EM CONJUNTO COM A OPCAO          */
/*   "TRACE(4)", PARA INIBIR A EXECUCAO DO PROGRAMA COMPILADO.   */
/*   SE TRACE(4) = '1'B, EXEC = '0'B.                            */
/*   A VARIAVEL "LIST" E' USADA EM CONJUNTO COM A OPCAO          */
/*   "TRACE(5)", PARA IMPRIMIR O PROGRAMA APOS A CARGA.          */
/*   SE TRACE(5) = '1'B, LIST = '1'B.                            */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                   */
/*   PLAPP32 - EXECUTA PROGRAMA COMPILADO.                       */
/*****************************************************************/

DCL EXEC            BIT(1)  EXTERNAL INIT('1'B) ;
DCL LIST            BIT(1)  EXTERNAL INIT('0'B) ;


/*****************************************************************/
/* MACRO PLAMV13                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA 'OBJ1'.                      */
/*   O ARQUIVO 'OBJ1' E' USADO PARA GUARDAR O PROGRAMA OBJETO    */
/* GERADO PELO COMPILADOR. A PARTIR DELE SERA' GERADO O PROGRA-  */
/* MA EXECUTAVEL EM 'OBJ2'.                                      */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*   PLAPP00 - PROGRAMA PRINCIPAL.                               */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                            */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.           */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.              */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS.                         */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.      */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                 */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                  */
/*   PLAPP23 - RECONHECEDOR DE COMANDO "IF".                     */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                   */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                    */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                   */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE" .                 */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.        */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                 */
```

```
/*    PLAPP30 - RECONHECEDOR DE EXPRESSAO.                          */
/*    PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                      */
/*    PLAPP34 - GRAVA REGISTRO DE INSTRUCOES AM 'OBJ1'.             */
/*****************************************************************/


DCL OBJ1            FILE RECORD ENV (F(4000,80)) ;
DCL REG1            CHAR(80) EXTERNAL ;
DCL QFIX1(20)       UNAL BIN FIXED(31,0) BASED (PTOBJ1) ;
DCL 1 QUADRA1(5)    UNAL BASED (PTOBJ1) ,
      2 COP1        BIT(8) ,
      2 OPN1(3) ,
        3 BASE1       BIT(8) ,
        3 OPND1       BIN FIXED(31,0) ;
DCL PTOBJ1          POINTER EXTERNAL ;
DCL LC1             BIN FIXED(15,0) EXTERNAL INIT(1) ;


/*****************************************************************/
/* MACRO PLAMV14                                                 */
/* DECLARACAO DA REFERENCIA EXTERNA 'OBJ2'.                      */
/*   O ARQUIVO 'OBJ2' E' USADO PARA GUARDAR O PROGRAMA OBJETO    */
/* EXECUTAVEL, GERADO A PARTIR DE 'OBJ1', QUE CONTEM O OBJETO    */
/* COMPILADO.                                                    */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                      */
/*    PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                   */
/*    PLAPP32 - EXECUTA PROGRAMA COMPILADO.                      */
/*****************************************************************/


DCL OBJ2     FILE RECORD ENV (F(4000,80)) ;


/*****************************************************************/
/* MACRO PLAMV15                                                 */
/* DECLARACAO DA ESTRUTURA "CODOPER".                            */
/* ESTA ESTRUTURA E' USADA PARA GUARDAR OS CODIGOS DOS TIPOS DE   */
/* AREAS, DE INSTRUCOES E DE OPERANDOS.                          */
/* "AREAS" CONTEM:                                               */
/*    "COAREA" - CODIGO DE REGISTRO DE DEFINICAO DE AREAS.       */
/*    "COCINT" - CODIGO DE AREA INTEIRA.                         */
/*    "COCREAL" - CODIGO DE AREA REAL.                           */
/*    "COCBOOL" - CODIGO DE AREA BOOLEANA.                       */
/*    "COCCHAR" - CODIGO DE AREA CHAR.                           */
/* "INSTRU" CONTEM OS CODIGOS DAS INSTRUCOES E OS LIMITES DES-   */
/* TES CODIGOS, PARA TESTE DE VALIDADE.                          */
/* "CODCON" CONTEM OS CODIGOS DE CONDICAO PARA TESTE E DESVIO:   */
/*    "CCGT"  - MAIOR                                            */
/*    "CCLT"  - MENOR                                            */
/*    "CCNEQ" - DIFERENTE                                        */
/*    "CCEQ"  - IGUAL                                            */
/*    "CCGE"  - MAIOR OU IGUAL                                   */
/*    "CCLE"  - MENOR OU IGUAL                                   */
/*    "CCOOR" - DESVIO INCONDICIONAL                             */
/* "TOPND" CONTEM OS CODIGOS DOS TIPOS DE OPERANDOS:             */
/*    "ITE" - TEMPORARIO INTEIRO                                 */
/*    "ICT" - CONSTANTE INTEIRA                                  */
/*    "IVD" - VARIAVEL INTEIRA, ENDERECO DIRETO                  */
/*    "IVI" - VARIAVEL INTEIRA, ENDERECO INDIRETO                */
/*    "RTE" - TEMPORARIO REAL                                    */
```

```
/*     "RCT" - CONSTANTE REAL                                          */
/*     "RVD" - VARIAVEL REAL, ENDERECO DIRETO                          */
/*     "RVI" - VARIAVEL REAL, ENDERECO INDIRETO                        */
/*     "BTE" - TEMPORARIO BOOLEANO                                     */
/*     "BCT" - CONSTANTE BOOLEANA                                      */
/*     "BVD" - VARIAVEL BOOLEANA, ENDERECO DIRETO                      */
/*     "BVI" - VARIAVEL BOOLEANA, ENDERECO INDIRETO                    */
/*     "CCT" - CONSTANTE CHAR                                          */
/*     "CVD" - VARIAVEL CHAR, ENDERECO DIRETO                          */
/*     "CVI" - VARIAVEL CHAR, ENDERECO INDIRETO                        */
/* OBS: OS ENDERECOS DE CTES E TEMPORARIOS SAO SEMPRE DIRETOS.         */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                            */
/*    PLAPP00 - PROGRAMA PRINCIPAL.                                    */
/*    PLAPP08 - RECONHECEDOR DE BLOCO.                                 */
/*    PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURES.               */
/*    PLAPP12 - RECCNHECEDOR DE CABECALHO DE FUNCOES.                  */
/*    PLAPP17 - RECONHECEDOR DE COMANDOS.                              */
/*    PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.           */
/*    PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                      */
/*    PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                       */
/*    PLAPP23 - RECONHECEDOR DE COMANDO "IF".                          */
/*    PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                        */
/*    PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                         */
/*    PLAPP26 - RECCNHECEDOR DE COMANDO "READ".                        */
/*    PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                       */
/*    PLAPP28 - RECCNHECEDOR DE SECAO DE PARAMETROS REAIS.             */
/*    PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                      */
/*    PLAPP30 - RECONHECEDOR DE EXPRESSAO.                             */
/*    PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                         */
/*    PLAPP32 - EXECUTA PROGRAMA COMPILADO.                            */
/*******************************************************************/


DCL 1 CODOPER EXTERNAL ,
      2 AREAS ,
        3 COAREA        BIT(8) INIT('00000000'B) ,
        3 COCINT        BIT(8) INIT('00000001'B) ,
        3 COCREAL       BIT(8) INIT('00000010'B) ,
        3 COCBOOL       BIT(8) INIT('00000011'B) ,
        3 COCCHAR       BIT(8) INIT('00000100'B) ,
      2 INSTRU ,
        3 COD ,
          4 COWRITE     BIN FIXED(15,0)  INIT(5) ,
          4 COWRITELN   BIN FIXED(15,0)  INIT(6) ,
          4 COWRITEPG   BIN FIXED(15,0)  INIT(7) ,
          4 COWRITED    BIN FIXED(15,0)  INIT(8) ,
          4 COWRITELND  BIN FIXED(15,0)  INIT(9) ,
          4 COWRITEPGD  BIN FIXED(15,0)  INIT(10) ,
          4 COEXIT      BIN FIXED(15,0)  INIT(11) ,
          4 COSET       BIN FIXED(15,0)  INIT(12),
          4 COATRIB     BIN FIXED(15,0)  INIT(13) ,
          4 COCVT       BIN FIXED(15,0)  INIT(14) ,
          4 COADREG     BIN FIXED(15,0)  INIT(15) ,
          4 COCREG      BIN FIXED(15,0)  INIT(16) ,
          4 CODESV      BIN FIXED(15,0)  INIT(17) ,
          4 COZEREG     BIN FIXED(15,0)  INIT(18) ,
          4 COADI       BIN FIXED(15,0)  INIT(19) ,
          4 COCDE       BIN FIXED(15,0)  INIT(20) ,
          4 COSUBI      BIN FIXED(15,0)  INIT(21) ,
          4 COMLTI      BIN FIXED(15,0)  INIT(22) ,
```

```
      4 CODIVI        BIN FIXED(15,0)  INIT(23) ,
      4 CCMOD         BIN FIXED(15,0)  INIT(24) ,
      4 COSETM        BIN FIXED(15,0)  INIT(25) ,
      4 COADR         BIN FIXED(15,0)  INIT(26) ,
      4 COSUBR        BIN FIXED(15,0)  INIT(27) ,
      4 CODIVR        BIN FIXED(15,0)  INIT(28) ,
      4 COMLTR        BIN FIXED(15,0)  INIT(29) ,
      4 CONOT         BIN FIXED(15,0)  INIT(30) ,
      4 COCOMP        BIN FIXED(15,0)  INIT(31) ,
      4 COAND         BIN FIXED(15,0)  INIT(32) ,
      4 COOR          BIN FIXED(15,0)  INIT(33) ,
      4 COREAD        BIN FIXED(15,0)  INIT(34) ,
      4 COREADP       BIN FIXED(15,0)  INIT(35) ,
      4 COREADD       BIN FIXED(15,0)  INIT(36) ,
      4 COREADPD      BIN FIXED(15,0)  INIT(37) ,
      4 COSETB        BIN FIXED(15,0)  INIT(38) ,
      4 COSTOB        BIN FIXED(15,0)  INIT(39) ,
      4 COEMPB        BIN FIXED(15,0)  INIT(40) ,
      4 CODESB        BIN FIXED(15,0)  INIT(41) ,
      4 COMSG         BIN FIXED(15,0)  INIT(42) ,
   3 LIMITES ,
      4 LINF          BIN FIXED(15,0) INIT(5) ,
      4 LSUP          BIN FIXED(15,0) INIT(42) ,
   3 VETCOD(5:42)   CHAR(4)
         INIT('WRTE' , 'WRLN' , 'WRPG' , 'WRTD' , 'WLND' ,
              'WPGD' , 'EXIT' , 'SET ' , 'ATRB' , 'CVT ' ,
              'ADRG' , 'CREG' , 'DESV' , 'ZREG' , 'ADI ' ,
              'CDE ' , 'SUBI' , 'MLTI' , 'DIVI' , 'MOD ' ,
              'SETM' , 'ADR ' , 'SUBR' , 'DIVR' , 'MLTR' ,
              'NOT ' , 'COMP' , 'AND ' , 'OR  ' , 'READ' ,
              'REDP' , 'REDD' , 'RDPD' , 'SETB' , 'STOB' ,
              'EMPB' , 'DESB' , 'MSG '                   ) ,
2 CODCON ,
   3 CCGT             BIN FIXED(15,0) INIT(2) ,
   3 CCLT             BIN FIXED(15,0) INIT(4) ,
   3 CCNEQ            BIN FIXED(15,0) INIT(6) ,
   3 CCEQ             BIN FIXED(15,0) INIT(8) ,
   3 CCGE             BIN FIXED(15,0) INIT(10) ,
   3 CCLE             BIN FIXED(15,0) INIT(12) ,
   3 CCQQR            BIN FIXED(15,0) INIT(15) ,
2 TOPND ,
   3 ITE              BIN FIXED(15,0) INIT(1) ,
   3 ICT              BIN FIXED(15,0) INIT(2) ,
   3 IVD              BIN FIXED(15,0) INIT(3) ,
   3 IVI              BIN FIXED(15,0) INIT(4) ,
   3 RTE              BIN FIXED(15,0) INIT(5) ,
   3 RCT              BIN FIXED(15,0) INIT(6) ,
   3 RVD              BIN FIXED(15,0) INIT(7) ,
   3 RVI              BIN FIXED(15,0) INIT(8) ,
   3 BTE              BIN FIXED(15,0) INIT(9) ,
   3 BCT              BIN FIXED(15,0) INIT(10) ,
   3 BVD              BIN FIXED(15,0) INIT(11) ,
   3 BVI              BIN FIXED(15,0) INIT(12) ,
   3 CCT              BIN FIXED(15,0) INIT(13) ,
   3 CVD              BIN FIXED(15,0) INIT(14) ,
   3 CVI              BIN FIXED(15,0) INIT(15) ;

/*********************************************************************/
/* MACRO PLAMV16                                                     */
```

```
/* DECLARACAO DA ESTRUTURA "REGAREA".                                  */
/*   ESTA ESTRUTURA E' USADA P/ GRAVACAO E LEITURA DO PRIMEIRO         */
/*   REGISTRO DO ARQUIVO "OBJ2", QUE CONTEM OS TAMANHOS DAS            */
/*   AREAS NECESSARIAS P/ EXECUCAO DO PROGRAMA COMPILADO.              */
/*    "CO" - CODIGO P/INDICAR REG. C/CONTADORES DE AREAS               */
/*    "CINT" - TAM. AREA P/CONSTANTES E VARIAVEIS INTEIRAS             */
/*    "CREAL" - TAM. AREA P/CONSTANTES E VARIAVEIS REAIS               */
/*    "CBOOL" - TAM. AREA P/CONSTANTES E VARIAVEIS BOOLEANAS           */
/*    "CCHAR" - TAM. AREA P/CONSTANTES E VARIAVEIS CHAR (EM            */
/*              CARACTERS)                                             */
/*    "TINT" - TAM. AREA P/ TEMPORARIOS INTEIROS                       */
/*    "TREAL" - TAM. AREA P/ TEMPORARIOS REAIS                         */
/*    "TBOOL" - TAM. AREA P/ TEMPORARIOS BOOLEANOS                     */
/*    "TCHAR" - TAM. AREA P/ TEMPORARIOS CHAR (EM CARACTERES)          */
/*    "NCINT" - NUMERO DE CTES INTEIRAS                                */
/*    "NCREAL" - NUMERO DE CTES REAIS                                  */
/*    "NCBOOL" - NUMERO DE CTES BOOLEANAS                              */
/*    "NCCHAR" - NUMERO DE CTES CHAR (EM CARACTERES)                   */
/*    "NQUAD" - TAM. AREA P/ INSTRUCOES (NUMERO DE QUADRUPLAS)         */
/*    "NBMAX" - TAM. DA MATRIZ DE REGISTRADORES BASE                   */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                            */
/*   PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                          */
/*   PLAPP32 - EXECUTA PROGRAMA COMPILADO.                             */
/*********************************************************************/


DCL 1 REGAREA BASED (P) ,
      2  CO              BIT (8) ,
      2  CINT            BIN FIXED (31,0) ,
      2  CREAL           BIN FIXED (31,0) ,
      2  CBOOL           BIN FIXED (31,0) ,
      2  CCHAR           BIN FIXED (31,0) ,
      2  TINT            BIN FIXED (31,0) ,
      2  TREAL           BIN FIXED (31,0) ,
      2  TBOOL           BIN FIXED (31,0) ,
      2  TCHAR           BIN FIXED (31,0) ,
      2  NCINT           BIN FIXED (31,0) ,
      2  NCREAL          BIN FIXED (31,0) ,
      2  NCBOOL          BIN FIXED (31,0) ,
      2  NCCHAR          BIN FIXED (31,0) ,
      2  NQUAD           BIN FIXED (31,0) ,
      2  NBMAX           BIN FIXED (31,0) ;


/*********************************************************************/
/* MACRO PLAMV18                                                     */
/* DECLARACAO DAS REFERENCIAS EXTERNAS "SCARDS" E "CARD" :           */
/*   "SCARDS" E' O ARQUIVO DE ENTRADA PARA O COMPILADOR E PARA       */
/*   O PROGRAMA DO USUARIO. DEVERA' CONTER O PROGRAMA FONTE E        */
/*   OS DADOS PARA A EXECUCAO DESTE PROGRAMA.                        */
/*   "CARD" E' A AREA USADA PARA LEITURA DE "SCARDS".                */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP04 - RECONHECEDOR LEXICO.                                  */
/*   PLAPP32 - EXECUTA PROGRAMA COMPILADO.                           */
/*********************************************************************/


DCL SCARDS FILE INPUT RECORD ENV(F(80)) EXTERNAL ;
DCL CARD(80)              CHAR(1) EXTERNAL ;
```

```
/*********************************************************************/
/* MACRO PLAMV19                                                     */
/* DECLARACAO DA REFERENCIA EXTERNA "SPRINT" :                       */
/*   "SPRINT" E' O ARQUIVO DE IMPRESSAO USADO PELO COMPILADOR E      */
/*   PELO PROGRAMA DO USUARIO. E' USADO EM FASE DE COMPILACAO        */
/*   PARA IMPRESSOES NORMAIS DE COMPILACAO (PROGRAMA FONTE, MEN-     */
/*   SAGENS DE ERRO, ETC.) E TAMBEM PARA INFORMACOES GERADAS PE-     */
/*   LA OPCAO "TRACE".                                               */
/* ESTA MACRO E' UTILIZADA NAS PROCEDURES :                          */
/*   PLAPP00 - MODULO PRINCIPAL DO COMPILADOR.                       */
/*   PLAPP01 - IMPRIME MENSAGENS DE ERRO.                            */
/*   PLAPP02 - IMPRIME TABELA DE SIMBOLOS.                           */
/*   PLAPP03 - RECONHECEDOR DE OPCOES DE COMPILACAO.                 */
/*   PLAPP04 - RECONHECEDOR LEXICO.                                  */
/*   PLAPP05 - PROCURA UM IDENTIFICADOR NA TABELA DE SIMBOLOS.       */
/*   PLAPP06 - INSERE UM NOVO IDENTIFICADOR NA TAB. DE SIMBOLOS      */
/*             ENTRY-POINT DA PLAPP05.                               */
/*   PLAPP07 - ELIMINA DA TABELA DE SIMBOLOS OS IDENTIFICADORES      */
/*             DEFINIDOS NO BLOCO CUJA ANALISE ESTA' SENDO           */
/*             ENCERRADA - ENTRY-POINT DA PLAPP05.                   */
/*   PLAPP08 - RECONHECEDOR DE BLOCO.                                */
/*   PLAPP09 - RECONHECEDOR DE DECLARACAO DE CONSTANTES.             */
/*   PLAPP10 - RECONHECEDOR DE DECLARACAO DE VARIAVEIS.              */
/*   PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.               */
/*   PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.                  */
/*   PLAPP13 - INSERE CONSTANTE NA TABELA DE CONSTANTES INTEIRAS     */
/*   PLAPP14 - INSERE CONSTANTE NA TABELA DE CONSTANTES REAIS -      */
/*             ENTRY-POINT DA PLAPP13.                               */
/*   PLAPP15 - INSERE CONSTANTE NA TABELA DE CONSTANTES ALFA-        */
/*             NUMERICAS - ENTRY-POINT DA PLAPP13.                   */
/*   PLAPP17 - RECONHECEDOR DE COMANDOS                              */
/*   PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS                     */
/*   PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1                      */
/*   PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS.          */
/*   PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT".                     */
/*   PLAPP22 - RECONHECEDOR DE COMANDO "WHILE".                      */
/*   PLAPP23 - RECONHECEDOR DE COMANDO "IF".                         */
/*   PLAPP24 - RECONHECEDOR DE COMANDO "CASE".                       */
/*   PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                        */
/*   PLAPP26 - RECONHECEDOR DE COMANDO "READ".                       */
/*   PLAPP27 - RECONHECEDOR DE COMANDO "WRITE".                      */
/*   PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.            */
/*   PLAPP29 - RECONHECEDOR DE INDICES DE ARRAY.                     */
/*   PLAPP30 - RECONHECEDOR DE EXPRESSAO.                            */
/*   PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                        */
/*   PLAPP32 - EXECUTA PROGRAMA COMPILADO.                           */
/*   PLAPP33 - IMPRIME MENSAGENS DURANTE EXECUCAO                    */
/*********************************************************************/

DCL SPRINT FILE OUTPUT STREAM EXTERNAL ;
```

G - Listagens dos Programas Fonte

```
 /* PLAPP00 - MODULO PRINCIPAL DO COMPILADOR PASCAL                    */
  PLAPP00 : PROC(PARM1) OPTIONS(MAIN) ;
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECE OPCOES DE COMPILACAO */
$CONTINUE WITH PLAMP03 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* RECONHECEDOR DE BLOCO */
$CONTINUE WITH PLAMP08 RETURN
1 /* GERADOR DE PROGRAMA OBJETO EM DISCO */
$CONTINUE WITH PLAMP31 RETURN
1 /* EXECUTA PROGRAMA COMPILADO */
$CONTINUE WITH PLAMP32 RETURN
1 /* IMPRIME MENSAGENS DE EXECUCAO */
$CONTINUE WITH PLAMP33 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO EOF */
$CONTINUE WITH PLAMV03 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL, TABALFA, ENDZERO, ENDUM, EFALSE,      */
  /* ETRUE, ENDCTE E POSCTE                                       */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO EXEC E LIST */
$CONTINUE WITH PLAMV12 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL PARM1                CHAR(100) VAR ;
 DCL PARM                 CHAR(100) ;
 DCL PARMCAR(100)         CHAR(1) DEF PARM ;
 DCL CARTAO               CHAR(80) ;
 DCL RBASE                BIN FIXED(15,0) STATIC ;
 DCL I                    BIN FIXED(15,0) STATIC ;
 DCL GUARDPT              BIN FIXED(15,0) STATIC ;
 DCL ADDR                 BUILTIN ;
 OPEN FILE(SPRINT) LINESIZE(132) ;
 OPEN FILE (OBJ1) OUTPUT TITLE ('1') ;
 PTOBJ1 = ADDR(REG1) ;
 PARM = PARM1 ;
 IF PARMCAR(1) ¬= ' '
 THEN IF PLAPP03(PARM) /* RECONHECE OPCOES DE COMPILACAO */
      THEN DO ; /* ERRO GRAVE NAS OPCOES DE COMPILACAO */
```

```
            ERRO = 10 ; /* COMPILACAO ENCERRADA DEVIDO ERRO GRAVE*/
            COLUNA = 1 ;
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
            DO WHILE(¬EOF) ; /* SO' EFETUA ANALISE LEXICA */
              CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
            END ;
            STOP ;
          END ;
EXEC = ¬TRACE(4) ;
LIST = TRACE(5) ;
GERA = EXEC | LIST ;
CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
CALL PLAPP01 ; /* INICIALIZAR TABELA DE ERROS */
CALL PLAPP33(0) ; /* INICIALIZAR TABELA DE MENSAGENS DE EXECUCAO */
/* INICIALIZAR A TABELA DE SIMBOLOS COM "TRUE" E "FALSE" */
IDNOME = 'FALSE' ;
CALL PLAPP06 ; /* INSERE NA TABELA */
IDTIPO(IND) = 3 ; /* CONSTANTE BOOLEANA */
PTCTE(IND) = 0 ; /* VALOR DA CONSTANTE BOOLEANA (FALSE) */
/* GERACAO DE CODIGO */
ENDER(IND) = 1 ;
/* FIM DA GERACAO DE CODIGO */
IDNOME = 'TRUE' ;
CALL PLAPP06 ; /* INSERE NA TABELA */
IDTIPO(IND) = 3 ; /* CONSTANTE BOOLEANA */
PTCTE(IND) = 1 ; /* VALOR DA CONSTANTE BOOLEANA (TRUE) */
/* GERACAO DE CODIGO */
ENDER(IND) = 2 ;
/* FIM DA GERACAO DE CODIGO */
IF TIPO ¬= 50 /* "PROGRAM" ? */
THEN DO ;
       ERRO = 9 ; /* FALTA "PROGRAM" */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TIPO ¬= 1 /* IDENTIFICADOR ? */
THEN DO ;
       ERRO = 11 ; /* FALTA NOME DO PROGRAMA. ASSUMIDO "PASCAL" */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TIPO ¬= 10 /* ";" ? */
THEN DO ;
       ERRO = 12 ; /* FALTA ";". ASSUMIDO */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF GERA
THEN DO ; /* GERA PROLOGO DO PROGRAMA PRINCIPAL */
       IF PROXINT + 8 > LSKINT
       THEN DO ; /* PLAPP00 - RESTRICAO DE IMPLEMENTACAO - NUME-*/
                 /* RO MAXIMO PERMITIDO DE CONSTANTES INTEIRAS  */
                 /* NO PROGRAMA FOI ULTRAPASSADO                */
              ERRO = 147 ;
              COLUNA = PTR - 1 ;
              CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
              ERRO = 41 ; /* COMPILACAO ABORTADA */
```

```
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
            STOP ;
          END ;
        RBASE = BLOCO + 1 ; /* BASE DO PROGRAMA PRINCIPAL */
        CALL PLAPP34(COZEREG,0,1,0,1,0,1) ; /* GRAVA REG.INST.*/
        /* GUARDA ENDERECO DA AREA QUE CONTERA O TAMANHO DAS      */
        /* AREAS DO PROGRAMA PRINCIPAL (VARIAVEIS E TEMPORARIOS) */
        /* PARA COMPLETAR AO FINAL DA COMPILACAO                 */
        GUARDPT = PROXINT ;
        /* GERA INSTRUCOES PARA CARREGAR OS VALORES DOS REGS. BA-*/
        /* SE PARA O PROGRAMA PRINCIPAL E ATUALIZAR OS VALORES   */
        /* PARA PROXIMA PROCEDURE OU FUNCAO.                     */
        DO I = 1 TO 8 ;
          CALL PLAPP34(COSETB,0,I,0,RBASE,0,I+2) ;
          CALL PLAPP34(COADI,0,I+2,0,PROXINT,0,I+2) ;
          PROXINT = PROXINT + 1 ;
        END ;
        /* INICIALIZA VARIAVEIS P/GERACAO DE ENDERECOS */
        /* DE VARIAVEIS E TEMPORARIOS.                 */
        CNTINT = LIVINT ;
        CNTREAL = LIVREAL ;
        CNTBOOL = LIVBOOL ;
        CNTCHAR = LIVCHAR ;
        TEMINT = 0 ;
        TEMREAL = 0 ;
        TEMBOOL = 0 ;
        TEMCHAR = 0 ;
      END ;
CALL PLAPP08 ; /* RECONHECE BLOCO */
IF BLOCO ¬= -1 & ¬ EOF
THEN DO ; /* NAO FECHOU TODOS OS BLOCOS E    */
          /* NAO PEGOU FIM DE ARQUIVO        */
        ERRO = 10 ; /* COMPILACAO ENCERRADA DEVIDO A ERRO GRAVE */
        COLUNA = 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        STOP ;
      END ;
IF EOF
THEN DO ; /* PEGOU FIM DE ARQUIVO */
        ERRO = 50 ; /* FIM FISICO ANTES DO FIM LOGICO */
        COLUNA = 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
      END ;
ELSE DO ;
        IF TIPO ¬= 8 /* "." ? */
        THEN DO ;
               ERRO = 15 ; /* FALTA PONTO FINAL. ASSUMIDO */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
             END ;
        ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        IF ¬EOF
        THEN DO ; /* PROGRAMA DO USUARIO AINDA NAO TERMINOU */
               ERRO = 16 ; /* FIM LOGICO ANTES DO FIM FISICO */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               ERRO = 10 ; /* COMPILACAO ENCERRADA DEVIDO */
                           /* ERRO GRAVE                  */
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               DO WHILE(¬EOF) ; /* SO' EFETUA ANALISE LEXICA */
```

```
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          END ;
          STOP ;
        END ;
    IF GERA
    THEN DO ;
          /* GERA CTES C/TAMANHO DAS AREAS (VARIAVEIS E TEM-*/
          /* PORARIOS) DO PROGRAMA PRINCIPAL E CTES C/OS VA-*/
          /* LORES A SEREM CARREGADOS NOS REGS. BASE DO PRO-*/
          /* GRAMA PRINCIPAL                                */
          VALINT(GUARDPT) = CNTINT - LIVINT ;
          VALINT(3) = PROXINT - LIVINT ;
          VALINT(GUARDPT+1) = CNTREAL - LIVREAL ;
          VALINT(4) = PROXRE - LIVREAL ;
          VALINT(GUARDPT+2) = CNTBOOL - LIVBOOL ;
          VALINT(5) = 3 - LIVBOOL ;
          VALINT(GUARDPT+3) = CNTCHAR - LIVCHAR ;
          VALINT(6) = PTALFA(ULTALFA) + TAMALFA(ULTALFA) -
                      LIVCHAR ;
          VALINT(GUARDPT+4) = TEMINT ;
          VALINT(7) = 0 ;
          VALINT(GUARDPT+5) = TEMREAL ;
          VALINT(8) = 0 ;
          VALINT(GUARDPT+6) = TEMBOOL ;
          VALINT(9) = 0 ;
          VALINT(GUARDPT+7) = TEMCHAR ;
          VALINT(10) = 0 ;
          CALL PLAPP34(COEXIT,0,0,0,0,0,0) ;
          IF LC1 ¬= 1
          THEN DO ; /* GRAVA ULTIMO REGISTRO DE 'OBJ1' */
                CINST = CINST + LC1 -1 ;
                WRITE FILE(OBJ1) FROM(REG1) ;
               END ;
          CLOSE FILE (OBJ1) ;
          CALL PLAPP31 ; /* GRAVA AS CTES EM OBJ1 */
          CALL PLAPP32 ; /* EXECUTA PROGRAMA COMPILADO */
        END ;
  END ;
END PLAPP00 ;
```

```
/* PLAPPO1 - IMPRESSAO DE MENSAGENS DE ERRO                          */
PLAPPO1 : PROC ;
/*********************************************************************/
/* ESTA PROCEDURE IMPRIME A MENSAGEM DE ERRO CUJO CODIGO ESTA'       */
/* NA VARIAVEL "ERRO". ANTES DA MENSAGEM, E' IMPRESSA UMA LINHA      */
/* DE ASTERISCOS QUE TERMINA COM UM INDICADOR PARA A POSICAO         */
/* APROXIMADA DO ERRO NO CARTAO. ESTA POSICAO E' INDICADA            */
/* ATRAVES DA VARIAVEL "COLUNA".                                     */
/* SAO UTILIZADAS 2 ESTRUTURAS, "MENS" E "COMP", PARA ARMAZENAR      */
/* AS MENSAGENS DE ERRO.                                             */
/* SE A MENSAGEM TIVER APENAS 53 CARACTERES, O CAMPO "SEG" DA        */
/* ESTRUTURA "MENS" CONTERA' 0 E SERA' IMPRESSO APENAS O CAMPO       */
/* "INICIO" DA MESMA ESTRUTURA.                                      */
/* CASO CONTRARIO, O FINAL DA MENSAGEM ESTARA' NO CAMPO              */
/* "COMPLEM" DA ESTRUTURA "COMP"; O CAMPO "SEG" CORRESPONDENTE       */
/* A PRIMEIRA PARTE DA MENSAGEM APONTARA' PARA A SEGUNDA PARTE       */
/* DA MENSAGEM; O CAMPO "TER" DA SEGUNDA PARTE APONTARA' PARA        */
/* A TERCEIRA PARTE (NA ESTRUTURA "COMP") E ASSIM POR DIANTE,        */
/* ATE' QUE NA ULTIMA PARTE O CAMPO "TER" CONTEM 0.                  */
/* SAO IMPRESSAS 2 PARTES DA MENSAGEM (SE HOUVER) EM CADA LINHA.     */
/*********************************************************************/
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL  I                    BIN FIXED (15,0) STATIC ;
 DCL 1 MENS(150)           STATIC ,
       2 INICIO            CHAR (53) ,
       2 SEG               BIN FIXED (15,0) ;
 DCL 1 COMP(67)            STATIC ,
       2 COMPLEM           CHAR(53) ,
       2 TER               BIN FIXED (15,0) ;
 DCL N                     BIN FIXED(15,0) STATIC ;
 DCL FLIPFLOP              BIT(1) STATIC ;
 DCL PRIMEIRA              BIT(1) STATIC INIT ('1'B) ;
 IF PRIMEIRA
 THEN DO ; /* INICIALIZA TABELA DE ERROS */
    PRIMEIRA = '0'B ; /* INDICA TABELA JA' INICIALIZADA */
    INICIO (1) =
    'OPCAO DE COMPILACAO INVALIDA.                          ' ; /*  */
    SEG (1) = 0 ;
    INICIO (2) =
    'DUPLICIDADE DE OPCAO DE COMPILACAO. A ANTERIOR SERA I' ; /*01*/
    SEG (2) = 1 ;
    INICIO (3) =
    'FALTA SINAL DE IGUAL NA OPCAO DE COMPILACAO TRACE.    ' ; /*  */
    SEG (3) = 0 ;
    INICIO (4) =
    'FALTA LISTA DE TRACES NA OPCAO DE COMPILACAO TRACE.   ' ; /*  */
    SEG (4) = 0 ;
    INICIO (5) =
    'FOI ESPECIFICADO TRACE < 1 OU > 16 NA OPCAO DE COMPIL' ; /*02*/
    SEG (5) = 2 ;
    INICIO (6) =
    'LIMITE SUPERIOR NAO E MAIOR QUE LIMITE INFERIOR NA OP' ; /*03*/
    SEG (6) = 3 ;
    INICIO (7) =
    'FALTA ")" NA OPCAO DE COMPILACAO TRACE.                ' ; /*  */
```

```
SEG (7) = 0 ;
INICIO (8) =
'ERRO DE SINTAXE NAS OPCOES DE COMPILACAO.                ' ; /*  */
SEG (8) = 0 ;
INICIO (9) =
'"PROGRAM" NAO ENCONTRADO. FOI ASSUMIDO.                  ' ; /*  */
SEG (9) = 0 ;
INICIO (10) =
'COMPILACAO ENCERRADA DEVIDO A ERRO GRAVE. PROSSEGUE A'  ; /*04*/
SEG (10) = 4 ;
INICIO (11) =
'FALTA NOME DO PROGRAMA. ASSUMIDO "PASCAL".               ' ; /*  */
SEG (11) = 0 ;
INICIO (12) =
'FALTA ";". FOI ASSUMIDO.                                 ' ; /*  */
SEG (12) = 0 ;
INICIO (13) =
'SIMBOLO INVALIDO PARA INICIO DE BLOCO.                   ' ; /*  */
SEG (13) = 0 ;
INICIO (14) =
'SIMBOLOS ANTERIORES, A PARTIR DA ULTIMA MENSAGEM DE E'  ; /*05*/
SEG (14) = 5 ;
INICIO (15) =
'FALTA PONTO FINAL. FOI ASSUMIDO.                         ' ; /*  */
SEG (15) = 0 ;
INICIO (16) =
'FIM LOGICO ANTES DO FIM FISICO.                          ' ; /*  */
SEG (16) = 0 ;
INICIO (17) =
'IDENTIFICADOR COM MAIS DE 8 CARACTERES. CONSIDERADOS '  ; /*06*/
SEG (17) = 6 ;
INICIO (18) =
'LITERAL COM MAIS DE 256 CARACTERES. CONSIDERADOS APEN'  ; /*07*/
SEG (18) = 7 ;
INICIO (19) =
'ERRO DE SINTAXE EM CONSTANTE INTEIRA. CONSIDERADA 1. '  ; /*  */
SEG (19) = 0 ;
INICIO (20) =
'CONSTANTE INTEIRA MAIOR QUE LIMITE PERMITIDO NA IMPLE'  ; /*08*/
SEG (20) = 8 ;
INICIO (21) =
'FALTA PARTE FRACIONARIA EM CONSTANTE REAL. CONSIDERAD'  ; /*09*/
SEG (21) = 9 ;
INICIO (22) =
'ERRO DE SINTAXE EM EXPOENTE DE CONSTANTE REAL. ATRIBU'  ; /*10*/
SEG (22) = 10 ;
INICIO (23) =
'ERRO DE SINTAXE NA PARTE FRACIONARIA DE CONSTANTE REA'  ; /*11*/
SEG (23) = 11 ;
INICIO (24) =
'UNDERFLOW EM CONSTANTE REAL (VALOR FINAL DO EXPOENTE '  ; /*12*/
SEG (24) = 12 ;
INICIO (25) =
'OVERFLOW EM CONSTANTE REAL (VALOR FINAL DO EXPOENTE M'  ; /*13*/
SEG (25) = 13 ;
INICIO (26) =
'CARACTER NAO DEFINIDO NA LINGUAGEM. IGNORADO.            ' ; /*  */
SEG (26) = 0 ;
INICIO (27) =
'LABEL INVALIDO.                                          ' ; /*  */
```

```
SEG (27) = 0 ;
INICIO (28) =
'SIMBOLO INVALIDO. "," OU ";" ESPERADOS.                    ' ; /*  */
SEG (28) = 0 ;
INICIO (29) =
'DECLARACAO DE LABEL EM DUPLICIDADE OU FORA DE SEQUENC' ; /*14*/
SEG (29) = 14 ;
INICIO (30) =
'DECLARACAO DE CONSTANTE EM DUPLICIDADE OU FORA DE SEQ' ; /*15*/
SEG (30) = 15 ;
INICIO (31) =
'DECLARACAO DE VARIAVEL EM DUPLICIDADE OU FORA DE SEQU' ; /*16*/
SEG (31) = 16 ;
INICIO (32) =
'FALTA NOME DE LABEL.                                       ' ; /*  */
SEG (32) = 0 ;
INICIO (33) =
'FALTA NOME DE CONSTANTE.                                   ' ; /*  */
SEG (33) = 0 ;
INICIO (34) =
'NOME DE CONSTANTE INVALIDO.                                ' ; /*  */
SEG (34) = 0 ;
INICIO (35) =
'SIMBOLO INVALIDO. "," OU "=" ESPERADOS.                    ' ; /*  */
SEG (35) = 0 ;
INICIO (36) =
'FALTA VALOR DA CONSTANTE NA DECLARACAO DE CONSTANTE. ' ; /*  */
SEG (36) = 0 ;
INICIO (37) =
'CONSTANTE NUMERICA INVALIDA EM DECLARACAO DE CONSTANT' ; /*17*/
SEG (37) = 17 ;
INICIO (38) =
'TIPO DE CONSTANTE INVALIDO EM DECLARACAO DE CONSTANTE' ; /*18*/
SEG (38) = 18 ;
INICIO (39) =
'PLAPP09 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*19*/
SEG (39) = 19 ;
INICIO (40) =
'SIMBOLO INVALIDO. ESPERADO "LABEL", "CONST", "VAR", "' ; /*21*/
SEG (40) = 21 ;
INICIO (41) =
'A COMPILACAO SERA'' ABORTADA DEVIDO A ERRO GRAVE.      ' ; /*  */
SEG (41) = 0 ;
INICIO (42) =
'PLAPP13 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG (42) = 22 ;
INICIO (43) =
'PLAPP14 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*24*/
SEG (43) = 24 ;
INICIO (44) =
'PLAPP15 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*26*/
SEG (44) = 26 ;
INICIO (45) =
'PLAPP15 - RESTRICAO DE IMPLEMENTACAO - AREA RESERVADA' ; /*28*/
SEG (45) = 28 ;
INICIO (46) =
'VALOR DECLARADO PARA CONSTANTE NAO E'' CONSTANTE. CON' ; /*30*/
SEG (46) = 30 ;
INICIO (47) =
'IDENTIFICADOR DECLARADO MAIS DE UMA VEZ.               ' ; /*  */
```

```
SEG (47) = 0 ;
INICIO (48) =
'IDENTIFICADOR NAO DECLARADO.                                    ' ; /*  */
SEG (48) = 0 ;
INICIO (49) =
'PLAPP06 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO            ' ; /*31*/
SEG (49) = 31 ;
INICIO (50) =
'ENCONTRADO O FIM FISICO DO PROGRAMA ANTES DO FIM LOGI           ' ; /*32*/
SEG (50) = 32 ;
INICIO (51) =
'PLAPP10 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO            ' ; /*33*/
SEG (51) = 33 ;
INICIO (52) =
'FALTA NOME DA VARIAVEL NA DECLARACAO DE VARIAVEL.               ' ; /*  */
SEG (52) = 0 ;
INICIO (53) =
'NOME DE VARIAVEL INVALIDO EM DECLARACAO DE VARIAVEL.            ' ; /*  */
SEG (53) = 0 ;
INICIO (54) =
'SIMBOLO INVALIDO. "," OU ":" ESPERADOS.                         ' ; /*  */
SEG (54) = 0 ;
INICIO (55) =
'FALTA TIPO DA VARIAVEL EM DECLARACAO DE VARIAVEL. CON           ' ; /*35*/
SEG (55) = 35 ;
INICIO (56) =
'FALTA "ABRE COLCHETE" EM DECLARACAO OU REFERENCIA A A           ' ; /*36*/
SEG (56) = 36 ;
INICIO (57) =
'FALTA INDICE DO ARRAY.                                          ' ; /*  */
SEG (57) = 0 ;
INICIO (58) =
'FALTA "FECHA COLCHETE" EM DECLARACAO OU REFERENCIA A            ' ; /*37*/
SEG (58) = 37 ;
INICIO (59) =
'SIMBOLO INVALIDO. ESPERADO "OF". FOI ASSUMIDO.                  ' ; /*  */
SEG (59) = 0 ;
INICIO (60) =
'SIMBOLO INVALIDO. ESPERADO "INTEGER", "REAL", "BOOLEA           ' ; /*38*/
SEG (60) = 38 ;
INICIO (61) =
'SIMBOLO INVALIDO. ESPERADO "," , ":" OU FECHA COLCHET           ' ; /*41*/
SEG (61) = 41 ;
INICIO (62) =
'SIMBOLO INVALIDO. ESPERADO "," OU FECHA COLCHETES.              ' ; /*  */
SEG (62) = 0 ;
INICIO (63) =
'FALTA LISTA DE IDENTIFICADORES.                                 ' ; /*  */
SEG (63) = 0 ;
INICIO (64) =
'DECLARACAO DE VARIAVEIS VAZIA.                                  ' ; /*  */
SEG (64) = 0 ;
INICIO (65) =
'LIMITE SUPERIOR DAS DIMENSOES DO ARRAY < LIMITE INFER           ' ; /*40*/
SEG (65) = 40 ;
INICIO (66) =
'PLAPP18 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO            ' ; /*39*/
SEG (66) = 39 ;
INICIO (67) =
'EXPRESSAO TIPO 1 INVALIDA.                                      ' ; /*  */
```

```
SEG (67) = 0 ;
INICIO (68) =
'SOBRA ")" OU FALTA "(" NA EXPRESSAO.                          ' ; /*  */
SEG (68) = 0 ;
INICIO (69) =
'IDENTIFICADOR NAO NUMERICO.                                   ' ; /*  */
SEG (69) = 0 ;
INICIO (70) =
'PLAPP19 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO          ' ; /*44*/
SEG (70) = 44 ;
INICIO (71) =
'DIVISAO POR ZERO NO CALCULO DA EXPRESSAO.                     ' ; /*  */
SEG (71) = 0 ;
INICIO (72) =
'SOBRA "(" OU FALTA ")" NA EXPRESSAO.                          ' ; /*  */
SEG (72) = 0 ;
INICIO (73) =
'FALTA NOME DA PROCEDURE.                                      ' ; /*  */
SEG (73) = 0 ;
INICIO (74) =
'SIMBOLO INVALIDO. ESPERADO ";".                               ' ; /*  */
SEG (74) = 0 ;
INICIO (75) =
'FALTA NOME DA FUNCAO.                                         ' ; /*  */
SEG (75) = 0 ;
INICIO (76) =
'FALTA ":". FOI ASSUMIDO.                                      ' ; /*  */
SEG (76) = 0 ;
INICIO (77) =
'FALTA TIPO DA FUNCAO. CONSIDERADO INDEFINIDO.                 ' ; /*  */
SEG (77) = 0 ;
INICIO (78) =
'PLAPP20 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO          ' ; /*45*/
SEG (78) = 45 ;
INICIO (79) =
'FALTA LISTA DE PARAMETROS.                                    ' ; /*  */
SEG (79) = 0 ;
INICIO (80) =
'SIMBOLO INVALIDO. ESPERADO "PROC", "FUNCTION", "VAR"          ' ; /*46*/
SEG (80) = 46 ;
INICIO (81) =
'PLAPP20 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO          ' ; /*47*/
SEG (81) = 47 ;
INICIO (82) =
'FALTA IDENTIFICADOR.                                          ' ; /*  */
SEG (82) = 0 ;
INICIO (83) =
'SIMBOLO INVALIDO. ESPERADO IDENTIFICADOR.                     ' ; /*  */
SEG (83) = 0 ;
INICIO (84) =
'FALTA ",". FOI ASSUMIDA.                                      ' ; /*  */
SEG (84) = 0 ;
INICIO (85) =
'SIMBOLO INVALIDO. ESPERADO ",".                               ' ; /*  */
SEG (85) = 0 ;
INICIO (86) =
'SIMBOLO INVALIDO. ESPERADO ";" OU ")".                        ' ; /*  */
SEG (86) = 0 ;
INICIO (87) =
'SIMBOLO INVALIDO. LABEL ESPERADO.                             ' ; /*  */
```

```
SEG (87) = 0 ;
INICIO (88) =
'SIMBOLO INVALIDO. "THEN" ESPERADO                              ' ; /*  */
SEG (88) = 0 ;
INICIO (89) =
'IDENTIFICADOR NAO E'' LABEL.                                   ' ; /*  */
SEG (89) = 0 ;
INICIO (90) =
'ESPERADO ":=". INSERIDO.                                       ' ; /*  */
SEG (90) = 0 ;
INICIO (91) =
'SIMBOLO INVALIDO.                                              ' ; /*  */
SEG (91) = 0 ;
INICIO (92) =
'ESPERADO "(". INSERIDO.                                        ' ; /*  */
SEG (92) = 0 ;
INICIO (93) =
'NUMERO DE PARAMETROS NAO CONFERE COM O DA DECLARACAO           ' ; /*48*/
SEG (93) = 48 ;
INICIO (94) =
'IDENTIFICADOR INVALIDO PARA INICIO DE COMANDO.                 ' ; /*  */
SEG (94) = 0 ;
INICIO (95) =
'"DO" ESPERADO. INSERIDO.                                       ' ; /*  */
SEG (95) = 0 ;
INICIO (96) =
'VARIAVEL DE CONTROLE ESPERADA.                                 ' ; /*  */
SEG (96) = 0 ;
INICIO (97) =
'VARIAVEL NAO E'' SIMPLES INTEIRA.                              ' ; /*  */
SEG (97) = 0 ;
INICIO (98) =
'"TO" OU "DOWNTO" ESPERADO. INSERIDO "TO".                      ' ; /*  */
SEG (98) = 0 ;
INICIO (99) =
'FALTA ROTULO DO COMANDO.                                       ' ; /*  */
SEG (99) = 0 ;
INICIO (100) =
'ROTULO INVALIDO.                                               ' ; /*  */
SEG (100) = 0 ;
INICIO (101) =
'IDENTIFICADOR INVALIDO PARA LISTA DE COMANDO "READ".           ' ; /*  */
SEG (101) = 0 ;
INICIO (102) =
'SIMBOLO INVALIDO. "," OU ")" ESPERADO.                         ' ; /*  */
SEG (102) = 0 ;
INICIO (103) =
'FALTA ")". INSERIDO.                                           ' ; /*  */
SEG (103) = 0 ;
INICIO (104) =
'IDENTIFICADOR INVALIDO PARA LISTA DE COMANDO "WRITE".          ' ; /*  */
SEG (104) = 0 ;
INICIO (105) =
'TIPO DA EXPRESSAO INCOMPATIVEL COM O DA VARIAVEL.              ' ; /*  */
SEG (105) = 0 ;
INICIO (106) =
'EXPRESSAO NAO E'' BOOLEANA OU E'' MATRICIAL.                   ' ; /*  */
SEG (106) = 0 ;
INICIO (107) =
'EXPRESSAO NAO E'' INTEIRA NEM CHAR OU E'' MATRICIAL.           ' ; /*  */
```

```
SEG (107) = 0 ;
INICIO (108) =
'CONSTANTE NAO E'' INTEIRA NEM CHAR.                          ' ; /*  */
SEG (108) = 0 ;
INICIO (109) =
'CONSTANTE NAO E'' DO MESMO TIPO DA EXPRESSAO.                ' ; /*  */
SEG (109) = 0 ;
INICIO (110) =
'EXPRESSAO NAO E'' INTEIRA OU E'' MATRICIAL.                  ' ; /*  */
SEG (110) = 0 ;
INICIO (111) =
'TIPO DE PARAMETRO DIFERENTE DO DEFINIDO NA DECLARACAO'       ; /*49*/
SEG (111) = 49 ;
INICIO (112) =
'FALTA PARAMETRO EM CHAMADA DE PROCEDURE OU FUNCAO.           ' ; /*  */
SEG (112) = 0 ;
INICIO (113) =
'NUMERO DE DIMENSOES NA REFERENCIA DO ARRAY DIFERENTE         ' ; /*50*/
SEG (113) = 50 ;
INICIO (114) =
'OPERANDO INVALIDO.                                           ' ; /*  */
SEG (114) = 0 ;
INICIO (115) =
'SIMBOLO INVALIDO. IGNORADO.                                  ' ; /*  */
SEG (115) = 0 ;
INICIO (116) =
'FALTA OPERANDO.                                              ' ; /*  */
SEG (116) = 0 ;
INICIO (117) =
'","  INSERIDA.                                               ' ; /*  */
SEG (117) = 0 ;
INICIO (118) =
'MISTURA DE MODO INVALIDA OU OPERACAO MATRICIAL ENTRE         ' ; /*51*/
SEG (118) = 51 ;
INICIO (119) =
'ATRIBUICAO ENTRE ARRAYS DE DIMENSOES DIFERENTES.             ' ; /*  */
SEG (119) = 0 ;
INICIO (120) =
'SINAL ANTES DE ROTULO EM COMANDO "CASE". SINAL IGNORA'       ; /*52*/
SEG (120) = 52 ;
INICIO (121) =
'DADO ALFANUMERICO COM MAIS DE 1 CARACTER E'' PERMITID'       ; /*53*/
SEG (121) = 53 ;
INICIO (122) =
'DUPLICIDADE DE LABEL. IGNORADO.                              ' ; /*  */
SEG (122) = 0 ;
INICIO (123) =
'PLAPP27 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO         ' ; /*54*/
SEG (123) = 54 ;
INICIO (124) =
'PLAPP18 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO         ' ; /*56*/
SEG (124) = 56 ;
INICIO (125) =
'PLAPP27 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO         ' ; /*57*/
SEG (125) = 57 ;
INICIO (126) =
'PROCEDURE OU FUNCAO PASSADA COMO PARAMETRO SO'' PODE         ' ; /*64*/
SEG (126) = 64 ;
INICIO (127) =
'PLAPP29 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO         ' ; /*59*/
```

```
SEG(127) = 59 ;
INICIO(128) =
'PLAPP29 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG(128) = 22 ;
INICIO (129) =
'DIVISAO POR ZERO.                                     ' ; /*  */
SEG (129) = 0 ;
INICIO (130) =
'PLAPP28 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*58*/
SEG (130) = 58 ;
INICIO (131) =
'PLAPP17 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG (131) = 22 ;
INICIO (132) =
'PLAPP22 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG (132) = 22 ;
INICIO (133) =
'LABEL REFERENCIADO MAS NAO DEFINIDO NO BLOCO.         ' ; /*  */
SEG (133) = 0 ;
INICIO (134) =
'PLAPP25 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG (134) = 22 ;
INICIO (135) =
'PLAPP23 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG (135) = 22 ;
INICIO (136) =
'PLAPP26 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG (136) = 22 ;
INICIO (137) =
'PLAPP08 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*57*/
SEG (137) = 57 ;
INICIO (138) =
'PLAPP26 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*60*/
SEG (138) = 60 ;
INICIO (139) =
'PLAPP24 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*22*/
SEG (139) = 22 ;
INICIO (140) =
'PLAPP24 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*61*/
SEG (140) = 61 ;
INICIO (141) =
'PLAPP24 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*26*/
SEG (141) = 26 ;
INICIO (142) =
'DUPLICIDADE DE ROTULO EM COMANDO "CASE".              ' ; /*  */
SEG (142) = 0 ;
INICIO (143) =
'PLAPP11 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*62*/
SEG(143) = 62 ;
INICIO (144) =
'PLAPP11 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*57*/
SEG(144) = 57 ;
INICIO (145) =
'PLAPP12 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*62*/
SEG(145) = 62 ;
INICIO (146) =
'PLAPP12 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*57*/
SEG(146) = 57 ;
INICIO (147) =
'PLAPP00 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO ' ; /*57*/
```

```
SEG(147) = 57 ;
INICIO (148) =
'PARAMETRO PROCEDURE OU FUNCAO NAO PODE TER NIVEL MAIO' ; /*65*/
SEG(148) = 65 ;
INICIO (149) =
'PARAMETRO REAL NAO PODE SER PARÂMETRO PROCEDURE NEM P' ; /*66*/
SEG (149) = 66 ;
INICIO (150) =
'***ERRO***ERRO***ERRO***ERRO***ERRO***ERRO***ERRO*** ' ; /*  */
SEG (150) = 0 ;
COMPLEM (1) =
'GNORADA.                                              ' ; /*  */
TER (1) = 0 ;
COMPLEM (2) =
'ACAO TRACE.                                           ' ; /*  */
TER (2) = 0 ;
COMPLEM (3) =
'CAO DE COMPILACAO TRACE.                              ' ; /*  */
TER (4) = 0 ;
COMPLEM (4) =
'PENAS A ANALISE LEXICA.                               ' ; /*  */
TER (4) = 0 ;
COMPLEM (5) =
'RRO, FORAM IGNORADOS.                                 ' ; /*  */
TER (5) = 0 ;
COMPLEM (6) =
'APENAS OS 8 PRIMEIROS.                                ' ; /*  */
TER (6) = 0 ;
COMPLEM (7) =
'NAS OS 256 PRIMEIROS.                                 ' ; /*  */
TER (7) = 0 ;
COMPLEM (8) =
'MENTACAO (2147483647). VALOR MAXIMO ASSUMIDO.         ' ; /*  */
TER (8) = 0 ;
COMPLEM (9) =
'A APENAS A PARTE INTEIRA.                             ' ; /*  */
TER (9) = 0 ;
COMPLEM (10) =
'IDO VALOR 0 AO EXPOENTE.                              ' ; /*  */
TER (10) = 0 ;
COMPLEM (11) =
'L. O VALOR 1 A CONSTANTE.                             ' ; /*  */
TER (11) = 0 ;
COMPLEM (12) =
'MENOR QUE -75). ATRIBUIDO O VALOR 0 A CONSTANTE.      ' ; /*  */
TER (12) = 0 ;
COMPLEM (13) =
'AIOR QUE 69). ATRIBUIDO O VALOR MAXIMO A CONSTANTE.   ' ; /*  */
TER (13) = 0 ;
COMPLEM (14) =
'IA.                                                   ' ; /*  */
TER (14) = 0 ;
COMPLEM (15) =
'UENCIA.                                               ' ; /*  */
TER (15) = 0 ;
COMPLEM (16) =
'ENCIA.                                                ' ; /*  */
TER (16) = 0 ;
COMPLEM (17) =
'E.                                                    ' ; /*  */
```

```
TER (17) = 0 ;
COMPLEM (18) =
'.                                                          ' ; /*  */
TER (18) = 0 ;
COMPLEM (19) =
'DE IDENTIFICADORES NUMA LISTA DE DECLARACAO DE CONS-       ' ; /*20*/
TER (19) = 20 ;
COMPLEM (20) =
'TANTE FOI ULTRAPASSADO.                                    ' ; /*  */
TER (20) = 0 ;
COMPLEM (21) =
'PROC", "FUNCTION" OU BEGIN".                               ' ; /*  */
TER (21) = 0 ;
COMPLEM (22) =
'DE CONSTANTES INTEIRAS NOS BLOCOS ATIVOS FOI ULTRA-        ' ; /*23*/
TER (22) = 23 ;
COMPLEM (23) =
'PASSADO.                                                   ' ; /*  */
TER (23) = 0 ;
COMPLEM (24) =
'DE CONSTANTES REAIS NOS BLOCOS ATIVOS FOI ULTRAPASSA-      ' ; /*25*/
TER (24) = 25 ;
COMPLEM (25) =
'DO.                                                        ' ; /*  */
TER (25) = 0 ;
COMPLEM (26) =
'DE CONSTANTES ALFANUMERICAS NOS BLOCOS ATIVOS FOI UL-      ' ; /*27*/
TER (26) = 27 ;
COMPLEM (27) =
'TRAPASSADO.                                                ' ; /*  */
TER (27) = 0 ;
COMPLEM (28) =
' PARA ARMAZENAR AS CONSTANTES ALFANUMERICAS DOS BLO-       ' ; /*29*/
TER (28) = 29 ;
COMPLEM (29) =
'COS ATIVOS FOI INSUFICIENTE.                               ' ; /*  */
TER (29) = 0 ;
COMPLEM (30) =
'SIDERADO INDEFINIDO.                                       ' ; /*  */
TER (30) = 0 ;
COMPLEM (31) =
'DE IDENTIFICADORES NOS BLOCOS ATIVOS FOI ULTRAPASSA-       ' ; /*43*/
TER (31) = 43 ;
COMPLEM (32) =
'CO.                                                        ' ; /*  */
TER (32) = 0 ;
COMPLEM (33) =
'DE IDENTIFICADORES NUMA LISTA DE DECLARACAO DE VARIA-      ' ; /*34*/
TER (33) = 34 ;
COMPLEM (34) =
'VEL (16) FOI ULTRAPASSADO.                                 ' ; /*  */
TER (34) = 0 ;
COMPLEM (35) =
'SIDERADO INDEFINIDO.                                       ' ; /*  */
TER (35) = 0 ;
COMPLEM (36) =
'RRAY. FOI ASSUMIDO.                                        ' ; /*  */
TER (36) = 0 ;
COMPLEM (37) =
'ARRAY. FOI ASSUMIDO.                                       ' ; /*  */
```

```
TER (37) = 0 ;
COMPLEM (38) =
'N" OU "CHAR".                                                    ' ; /*  */
TER (38) = 0 ;
COMPLEM (39) =
'DE DIMENSOES DE ARRAY NOS BLOCOS ATIVOS FOI ULTRAPAS-            ' ; /*42*/
TER (39) = 42 ;
COMPLEM (40) =
'IOR.                                                             ' ; /*  */
TER (40) = 0 ;
COMPLEM (41) =
'ES.                                                              ' ; /*  */
TER (41) = 0 ;
COMPLEM (42) =
'SADO.                                                            ' ; /*  */
TER (42) = 0 ;
COMPLEM (43) =
'DO.                                                              ' ; /*  */
TER (43) = 0 ;
COMPLEM (44) =
'DE OPERADORES NA EXPRESSAO FOI ULTRAPASSADO.                     ' ; /*  */
TER (44) = 0 ;
COMPLEM (45) =
'DE PARAMETROS NA LISTA FOI ULTRAPASSADO.                         ' ; /*  */
TER (45) = 0 ;
COMPLEM (46) =
'OU IDENTIFICADOR.                                                ' ; /*  */
TER (46) = 0 ;
COMPLEM (47) =
'DE PARAMETROS NOS BLOCOS ATIVOS FOI ULTRAPASSADO.                ' ; /*  */
TER (47) = 0 ;
COMPLEM (48) =
'DA PROCEDURE.                                                    ' ; /*  */
TER (48) = 0 ;
COMPLEM (49) =
' DA PROCEDURE OU FUNCAO.                                         ' ; /*  */
TER (49) = 0 ;
COMPLEM (50) =
'DO DEFINIDO NA DECLARACAO DO ARRAY.                              ' ; /*  */
TER (50) = 0 ;
COMPLEM (51) =
'ARRAYS DE DIMENSOES DIFERENTES.                                  ' ; /*  */
TER (51) = 0 ;
COMPLEM (52) =
'DO.                                                              ' ; /*  */
TER (52) = 0 ;
COMPLEM (53) =
'O APENAS EM COMANDO "WRITE".                                     ' ; /*  */
TER (53) = 0 ;
COMPLEM (54) =
'PERMITIDO DE ELEMENTOS EM LISTA DE COMANDO "WRITE"               ' ; /*55*/
TER (54) = 55 ;
COMPLEM (55) =
'FOI ULTRAPASSADO.                                                ' ; /*  */
TER (55) = 0 ;
COMPLEM (56) =
'PERMITIDO DE ELEMENTOS EM ARRAY FOI ULTRAPASSADO.                ' ; /*  */
TER (56) = 0 ;
COMPLEM (57) =
'PERMITIDO DE CONSTANTES INTEIRAS NO PROGRAMA                     ' ; /*55*/
```

```
    TER (57) = 55 ;
    COMPLEM (58) =
    'PERMITIDO DE ELEMENTOS NA SECAO DE PARAMETROS REAIS  ' ; /*55*/
    TER (58) = 55 ;
    COMPLEM (59) =
    'PERMITIDO DE INDICES DE ARRAY FOI ULTRAPASSADO.      ' ; /*  */
    TER (59) = 0 ;
    COMPLEM (60) =
    'PERMITIDO DE ELEMENTOS EM LISTA DE COMANDO "READ"    ' ; /*55*/
    TER (60) = 55 ;
    COMPLEM (61) =
    'PERMITIDO DE ROTULOS EM COMANDO "CASE" FOI ULTRAPAS- ' ; /*42*/
    TER (61) = 42 ;
    COMPLEM (62) =
    'PERMITIDO DE ELEMENTOS NA TABELA DE PARAMETROS       ' ; /*63*/
    TER(62) = 63 ;
    COMPLEM (63) =
    'FORMAIS FOI ULTRAPASSADO.                            ' ; /*  */
    TER(63) = 0 ;
    COMPLEM (64) =
    'TER PARAMETROS POR VALOR.                            ' ; /*  */
    TER (64) = 0 ;
    COMPLEM (65) =
    'R QUE O DA PROCEDURE CHAMADA.                        ' ; /*  */
    TER (65) = 0 ;
    COMPLEM (66) =
    'ARAMETRO FUNCAO.                                     ' ; /*  */
    TER (66) = 0 ;
    COMPLEM (67) =
    '***ERRO***ERRO***ERRO***ERRO***ERRO***ERRO***ERRO ***' ; /*  */
    TER (67) = 0 ;
      END ;
ELSE DO ;
       GERA = '0'B ; /* INIBE GERACAO DE CODIGO */
       PUT FILE (SPRINT) SKIP EDIT
                ('***** ERRO ',ERRO,('*' DO I = 1 TO COLUNA+2),'|')
                (A , F(3) , COL(16) ,(COLUNA+3)A , A) ;
       PUT FILE (SPRINT) EDIT(INICIO(ERRO))(COL(7) , A) ;
       N = SEG(ERRO) ;
       FLIPFLOP = '1'B ;
       DO WHILE(N ¬= 0) ;
         IF FLIPFLOP
         THEN PUT FILE (SPRINT) EDIT(COMPLEM(N))(A) ;
         ELSE PUT FILE (SPRINT) EDIT(COMPLEM(N))(COL(7) , A) ;
         N = TER(N) ;
         FLIPFLOP = ¬FLIPFLOP ;
       END ;
     END ;
END PLAPP01 ;
```

```
/* PLAPP02 - IMPRIME TABELA DE SIMBOLOS                              */
PLAPP02 : PROC(AVAIL) ;
/*********************************************************************/
/* ESTA PROCEDURE TEM COMO FUNCAO IMPRIMIR A TABELA DE SIMBOLOS      */
/* PARA EFEITO DE DEPURACAO. RECEBE EM 'AVAIL' UM APONTADOR P/       */
/* O INICIO DA PARTE LIVRE DA TABELA, E IMPRIME APENAS A PARTE       */
/* OCUPADA DA TABELA E OS ELEMENTOS DO VETOR 'CABEC' CUJOS           */
/* VALORES SEJAM DIFERENTES DE VAZIO (APONTADOR NULO OU DE FIM       */
/* DE LISTA).                                                        */
/*********************************************************************/
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL AVAIL    BIN FIXED(15,0) ;
 DCL I        BIN FIXED(15,0) ;
 PUT FILE(SPRINT) EDIT('IMPRESSAO DA TABELA DE SIMBOLOS') (COL(1),A);
 DO I = 0 TO 63 ;
   IF CABEC(I) ¬= 0
   THEN PUT SKIP DATA(CABEC(I)) ;
 END ;
 DO I = 1 TO AVAIL - 1 ;
   PUT FILE (SPRINT) SKIP DATA(NOME(I), NIVEL(I), LINK(I),
                               IDTIPO(I), PTCTE(I), ENDER(I)) ;
 END ;
 END PLAPP02 ;
```

```
/* PLAPP03 - RECONHECEDOR DAS OPCOES DE COMPILACAO                    */
PLAPP03 : PROC(PARM1) RETURNS(BIT(1)) ;
/*********************************************************************/
/* ESTA FUNCAO DECODIFICA O PARM, RECEBIDO PELO MODULO PRINCI-       */
/* PAL DO COMPILADOR (PLAPP00), QUE CONTEM AS OPCOES DE COMPI-       */
/* LACAO DESEJADAS PELO PROGRAMADOR (SEPARADAS POR VIRGULAS E A      */
/* ULTIMA SEGUIDA DE PELO MENOS UM BRANCO, CASO O PARM TENHA         */
/* MENOS DE 100 CARACTERES. ISTO E', UM BRANCO TAMBEM DETERMINA      */
/* O FIM DA ESPECIFICACAO DAS OPCOES), E EFETUA AS INICIALIZA-       */
/* COES NECESSARIAS. DEVOLVE '1'B EM CASO DE ERRO GRAVE E '0'B       */
/* CASO CONTRARIO. SAO CONSIDERADOS ERROS NAO GRAVES :               */
/*   - OPCAO EM DUPLICATA                                            */
/* AS OPCOES DE COMPILACAO ESTAO DESCRITAS A SEGUIR :                */
/* TRACE - ATIVA OS DIVERSOS TRACES CONTIDOS NO COMPILADOR PARA      */
/*         AUXILIAR NA DEPURACAO. SAO DISPONIVEIS 16 TRACES DI-      */
/*         FERENTES. PARA CONHECER SUAS FUNCOES, VEJA LISTAGEM       */
/*         DA MACRO PLAMV05 (MACRO TRACE).                           */
/*   FORMATOS :                                                      */
/*   1) TRACE=N - ATIVA TRACE N                                      */
/*   2) TRACE=(N) - ATIVA TRACE N                                    */
/*   3) TRACE=(N1,N2,...,N) - ATIVA OS TRACES N1,N2,...,N            */
/*   4) TRACE=(N1-N2) - ATIVA OS TRACES N1 ATE N2                    */
/*   OBS.- AS DUAS ULTIMAS FORMAS PODEM SER COMBINADAS. POR EX:      */
/*         TRACE=(N1,N2-N3)                                          */
/*       - A UNICA RESTRICAO DE ORDENACAO E' COM REFERENCIA A        */
/*         FORMA 4 (OU 4 COMBINADA COM 3), EM QUE N2 TEM QUE         */
/*         SER MAIOR QUE N1.                                         */
/*********************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL PARM1          CHAR(100) ;
 DCL PARM(100)      CHAR(1) DEF PARM1 ;
 DCL VETOR(100)     CHAR(1) ;
 DCL VET            CHAR(8) DEF VETOR ;
 DCL I              BIN FIXED(15,0) INIT(1) ;
 DCL J              BIN FIXED(15,0) ;
 DCL K              BIN FIXED(15,0) ;
 DCL N              BIN FIXED(15,0) ;
 DCL FIM            BIT(1) INIT('0'B) ;
 DCL TEMMAIS        BIT(1) INIT('1'B) ;
 DCL IMPRIME        BIT(1) INIT('1'B) ;
 DCL TABOP(2)       CHAR(8) INIT('TRACE   ','        ') ;
 DCL LABOP(2)       AUTOMATIC LABEL INIT(LTRACE , LERRO ) ;
 DCL NUMOP          BIN FIXED(15,0) INIT(1) ;
 DCL LIMINF         BIN FIXED(15,0) ;
 DCL LIMSUP         BIN FIXED(15,0) ;
 DCL INDEX          BUILTIN ;
 DO WHILE(¬FIM) ;
   J = 1 ;
   DO I = I TO 100 WHILE(PARM(I) ¬= ' ' & PARM(I) ¬= ',' &
                         PARM(I) ¬= '=') ;
     VETOR(J) = PARM(I) ;
     J = J + 1 ;
   END ;
```

```
   IF J > 9
   THEN DO ; /* ERRO */
          IF IMPRIME
          THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
          ERRO = 1 ; /* OPCAO DE COMPILACAO INVALIDA */
          COLUNA = I - 1 ;
          CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
          RETURN('1'B) ;
        END ;
   DO N = J TO 8 ;
     VETOR(N) = ' ' ;
   END ;
   TABOP(NUMOP+1) = VET ;
   DO N = 1 BY 1 WHILE(VET ¬= TABOP(N)) ;
   END ;
   GOTO LABOP(N) ;
 LERRO :
   IF IMPRIME
   THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
   ERRO = 1 ; /* OPCAO DE COMPILACAO INVALIDA */
   COLUNA = I - 1 ;
   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
   RETURN('1'B) ;
 DUPTRACE :
   IF IMPRIME
   THEN DO ;
          IMPRIME = '0'B ;
          PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
        END ;
   ERRO = 2 ; /* DUPLICIDADE DE OPCAO DE COMPILACAO */
   COLUNA = I - 1 ;
   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
 LTRACE :
   IF I > 100 | PARM(I) ¬= '='
   THEN DO ;
          IF IMPRIME
          THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
          ERRO = 3 ; /* FALTA SINAL DE IGUAL NA OPCAO TRACE */
          COLUNA = I ;
          CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
          RETURN('1'B) ;
        END ;
   IF I > 99
   THEN DO ;
          IF IMPRIME
          THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
          ERRO = 4 ; /* FALTA LISTA DE TRACES */
          COLUNA = I + 1 ;
          CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
          RETURN('1'B) ;
        END ;
   I = I + 1 ;
   IF PARM(I) ¬= '('
   THEN DO ;
          LIMINF = NUMERO ; /* RETORNA NUMERO = 0 EM CASO DE ERRO */
          IF LIMINF = 0
          THEN DO ; /* ERRO 5 - NUMERO < 1 OU > 16 */
                 IF IMPRIME
                 THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
```

```
                RETURN('1'B) ;
              END ;
            TRACE(LIMINF) = '1'B ;
          END ;
        ELSE DO ;
          I = I + 1 ;
          IF I > 99 | PARM(I) = ')'
          THEN DO ;
                 IF IMPRIME
                 THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
                 ERRO = 4 ; /* FALTA LISTA DE TRACES */
                 COLUNA = I ;
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                 RETURN('1'B) ;
               END ;
          DO WHILE(TEMMAIS) ;
            LIMINF = NUMERO ; /* RETORNA NUM.= 0 EM CASO DE ERRO */
            IF LIMINF = 0
            THEN DO ; /* ERRO 5 - NUMERO < 1 OU > 16 */
                   IF IMPRIME
                   THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   RETURN('1'B) ;
                 END ;
            IF I < 100 & PARM(I) = '-'
            THEN DO ;
                   I = I + 1 ;
                   LIMSUP = NUMERO ; /* RETORNA NUM.= 0 CASO DE ERRO */
                   IF LIMSUP = 0
                   THEN DO ; /* ERRO 5 - NUMERO < 1 OU > 16 */
                          IF IMPRIME
                          THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20),A);
                          CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                          RETURN('1'B) ;
                        END ;
                   IF LIMSUP ¬> LIMINF
                   THEN DO ;
                          IF IMPRIME
                          THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20),A);
                          ERRO = 6 ; /* LIMITE SUP. ¬> LIMITE INF.*/
                          COLUNA = I - 1 ;
                          CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                          RETURN('1'B) ;
                        END ;
                   DO K = LIMINF TO LIMSUP ;
                     TRACE(K) = '1'B ;
                   END ;
                 END ;
            ELSE TRACE(LIMINF) = '1'B ;
            IF I > 100 | PARM(I) ¬= ','
            THEN TEMMAIS = '0'B ;
            ELSE I = I + 1 ;
          END ;
          IF I > 100 | PARM(I) ¬= ')'
          THEN DO ;
                 IF IMPRIME
                 THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
                 ERRO = 7 ; /* FALTA ) */
                 COLUNA = I ;
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
```

```
            RETURN('1'B) ;
          END ;
        I = I + 1 ;
      END ;
  LABOP(1) = DUPTRACE ;
  GOTO ENDCASE ;
  NUMERO : PROC RETURNS(BIN FIXED) ;
    DCL NUM          BIN FIXED(15,0) ;
    DCL AUX          BIN FIXED(15,0) ;
    NUM = 0 ;
    IF I > 100
    THEN DO ;
           ERRO = 4 ; /* FALTA LISTA DE TRACES */
           COLUNA = I ;
           RETURN(0) ;
         END ;
    AUX = INDEX('0123456789' , PARM(I)) ;
    DO WHILE(AUX ¬= 0 & I < 101) ;
      NUM = NUM * 10  + AUX - 1 ;
      IF NUM > 16
      THEN DO ;
             ERRO = 5 ; /* ESPECIFICADO TRACE < 1 OU > 16 */
             COLUNA = I ;
             RETURN(0) ;
           END ;
      I = I + 1 ;
      IF I < 101
      THEN AUX = INDEX('0123456789' , PARM(I)) ;
    END ;
    IF NUM = 0
    THEN DO ;
           ERRO = 5 ; /* ESPECIFICADO TRACE < 1 OU > 16 */
           COLUNA = I - 1 ;
         END ;
    RETURN(NUM) ;
  END NUMERO ;
ENDCASE :
  IF I > 100 | PARM(I) = ' '
  THEN FIM = '1'B ;
  ELSE IF PARM(I) ¬= ','
       THEN DO ;
              IF IMPRIME
              THEN PUT FILE(SPRINT) EDIT(PARM1)(COL(20) , A) ;
              ERRO = 8 ; /* ERRO DE SINTAXE NAS OPCOES */
              COLUNA = I ;
              CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
              RETURN('1'B) ;
            END ;
  I = I + 1 ;
END ;
RETURN('0'B) ;
END PLAPP03 ;
```

```
/* PLAPP04 - RECONHECEDOR LEXICO DO COMPILADOR                    */
PLAPP04 : PROC ;
/*****************************************************************/
/* ESTA PROCEDURE TEM COMO FUNCOES :                             */
/*  - PULAR BRANCOS SUPERFLUOS E COMENTARIOS.                    */
/*  - LER E IMPRIMIR OS CARTOES.                                 */
/*  - DEVOLVER A CADA CHAMADA O PROXIMO ELEMENTO LEXICO EXIS-    */
/*    TENTE NA SEQUENCIA DE ENTRADA, BEM COMO A CATEGORIA A QUE  */
/*    PERTENCE O ELEMENTO LEXICO DEVOLVIDO.                      */
/*    AO SER DETETADO O FIM DO ARQUIVO QUE CONTEM O PROGRAMA DO  */
/*    DO USUARIO, SE JA' TIVER SIDO INICIADO O RECONHECIMENTO    */
/*    DE ALGUM ELEMENTO LEXICO, A ANALISE E' ENCERRADA NORMAL-   */
/*    MENTE; CASO CONTRARIO, A ANALISE E' ABORTADA, RETORNANDO   */
/*    "TIPO" COM VALOR 0. EM AMBOS OS CASOS E' ATRIBUIDO O VA-   */
/*    LOR '1'B A VARIAVEL "EOF".                                 */
/* UM IDENTIFICADOR E' UMA SEQUENCIA DE LETRAS E DIGITOS, COME-  */
/* CANDO POR UMA LETRA, COM NO MAXIMO 8 CARACTERES, E QUE NAO    */
/* SEJA UMA PALAVRA RESERVADA. CASO UM IDENTIFICADOR TENHA MAIS  */
/* DE 8 CARACTERES SERAO CONSIDERADOS APENAS OS 8 PRIMEIROS E    */
/* SERA' EMITIDA UMA MENSAGEM DE ERRO.                           */
/* NUMERO INTEIRO E' UMA SEQUENCIA DE 1 OU MAIS DIGITOS, CUJO    */
/* VALOR ESTEJA ENTRE 0 E 2147483647. SE O NUMERO INTEIRO NAO    */
/* TERMINAR POR UM DELIMITADOR VALIDO PARA NUMERO, SERA' DADA    */
/* MENSAGEM DE ERRO, ATRIBUIDO AO NUMERO O VALOR 1 E PROCURADO   */
/* UM DESTES DELIMITADORES. CASO O NUMERO INTEIRO SEJA MAIOR     */
/* QUE 2147483647, SERA' DADA MENSAGEM DE ERRO E ATRIBUIDO AO    */
/* NUMERO O VALOR 2147483647. NUMERO REAL E' COMPOSTO DE UMA     */
/* PARTE INTEIRA SEGUIDA DE UMA PARTE FRACIONARIA E/OU UMA PAR-  */
/* TE EXPONENCIAL. A PARTE INTEIRA E' UMA SEQUENCIA DE 1 OU      */
/* MAIS DIGITOS. A PARTE FRACIONARIA SE COMPOE DE UM PONTO SE-   */
/* GUIDO DE 1 OU MAIS DIGITOS.A PARTE EXPONENCIAL SE COMPOE DA   */
/* LETRA "E" SEGUIDA DE UM SINAL OPCIONAL, SEGUIDO DE 1 OU MAIS  */
/* DIGITOS. EM CASO DE ERRO E' DADA MENSAGEM E :                 */
/*  - EM CASO DE UNDERFLOW E' ATRIBUIDO AO NUMERO REAL O VALOR   */
/*    0.                                                         */
/*  - EM CASO DE OVERFLOW E' ATRIBUIDO AO NUMERO REAL O VALOR    */
/*    7237008 * 10 ** 69                                         */
/*  - EM CASO DE ERRO NA PARTE FRACIONARIA OU NA PARTE EXPO-     */
/*    NENCIAL, E' ATRIBUIDO AO NUMERO REAL O VALOR 1.0           */
/* LITERAL E' UMA SEQUENCIA DE ATE' 256 CARACTERES ANTECEDIDA E  */
/* SUCEDIDA POR UM DELIMITADOR DE LITERAIS. OS DELIMITADORES DE  */
/* LITERAIS SAO OS SIMBOLOS ASPAS OU APOSTROFE. O SIMBOLO USADO  */
/* PARA INICIAR O LITERAL TEM QUE SER O MESMO USADO PARA ENCER-  */
/* RA-LO, E NAO PODE ESTAR CONTIDO NA SEQUENCIA (O OUTRO PODE).  */
/* CASO O LITERAL CONTENHA MAIS DE 256 CARACTERES, E' DADA MEN-  */
/* SAGEM DE ERRO, SAO CONSIDERADOS APENAS OS 256 PRIMEIROS CA-   */
/* RACTERES E OS CARACTERES SEGUINTES SAO IGNORADOS ATE' SER     */
/* ENCONTRADO UM ";".                                            */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO EOF */
$CONTINUE WITH PLAMV03 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO SCARDS E CARD */
```

```
$CONTINUE WITH PLAMV18 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL INDEX            BUILTIN ;
 DCL IDCHAR(8)        CHAR(1) DEF IDENT ;
 DCL CAR              CHAR(1) INIT(' ') STATIC ;
 DCL DELIM            CHAR(1) STATIC ;
 DCL I                BIN FIXED(15,0) STATIC ;
 DCL L                BIN FIXED(15,0) STATIC ;
 DCL U                BIN FIXED(15,0) STATIC ;
 DCL MAXINT           BIN FIXED(31,0) STATIC ;
 DCL MAXREAL          BIN FLOAT(21) STATIC ;
 DCL MAXMANT          BIN FIXED(31,0) STATIC ;
 DCL LIMINT           BIN FIXED(31,0) STATIC ;
 DCL LIMMANT          BIN FIXED(31,0) STATIC ;
 DCL EXPMIN           BIN FIXED(15,0) STATIC ;
 DCL EXPMAX           BIN FIXED(15,0) STATIC ;
 DCL FIMNUM           CHAR(17) INIT('+-><  =¬!),;*/&|%:') STATIC ;
 DCL OVE              BIT(1) STATIC ;
 DCL EXP              BIN FIXED(31,0) STATIC ;
 DCL EXP1             BIN FIXED(31,0) STATIC ;
 DCL EXPLIDO          BIN FIXED(31,0) STATIC ;
 DCL EXPNEG           BIT(1) STATIC ;
 DCL DIG              PIC '9' STATIC ;
 DCL CLASSE           BIN FIXED(15,0) INIT(1) STATIC ;
 DCL OK               BIT(1) INIT('0'B) STATIC ;
 DCL TABRES(59)       CHAR(8) STATIC
                          INIT('ABS     ' , 'ARCTAN  ' , 'ARRAY   ' ,
                               'BEGIN   ' , 'BOOLEAN ' , 'CASE    ' ,
                               'CHAR    ' , 'CHR     ' , 'CONST   ' ,
                               'COS     ' , 'DIV     ' , 'DO      ' ,
                               'DOWNTO  ' , 'ELSE    ' , 'END     ' ,
                               'EOF     ' , 'EOLN    ' , 'EOPG    ' ,
                               'EXP     ' , 'FOR     ' , 'FORWARD ' ,
                               'FUNCTION' , 'GOTO    ' , 'IF      ' ,
                               'IN      ' , 'INTEGER ' , 'LABEL   ' ,
                               'LN      ' , 'MOD     ' , 'NEW     ' ,
                               'ODD     ' , 'OF      ' , 'ORD     ' ,
                               'PRED    ' , 'PROC    ' , 'PROGRAM ' ,
                               'READ    ' , 'READD   ' , 'READP   ' ,
                               'READPD  ' , 'REAL    ' , 'REPEAT  ' ,
                               'ROUND   ' , 'SIN     ' , 'SQR     ' ,
                               'SQRT    ' , 'SUCC    ' , 'THEN    ' ,
                               'TO      ' , 'TRUNC   ' , 'UNTIL   ' ,
                               'VAR     ' , 'WHILE   ' , 'WRITE   ' ,
                               'WRITED  ' , 'WRITELN ' , 'WRITELND' ,
                               'WRITEPG ' , 'WRITEPGD') ;
 DCL ATIPO(59)        BIN FIXED(15,0) STATIC
                                  INIT( 99 ,  99 ,  57 ,
                                        56 ,  78 ,  64 ,
                                        79 ,  99 ,  52 ,
                                        99 ,  24 ,  72 ,
                                        73 ,  70 ,  67 ,
                                        99 ,  99 ,  99 ,
                                        99 ,  75 ,  80 ,
                                        55 ,  66 ,  68 ,
                                        41 ,  76 ,  51 ,
                                        99 ,  25 ,  65 ,
                                        99 ,  58 ,  99 ,
                                        99 ,  54 ,  50 ,
```

```
                                        91 ,  93 ,  92 ,
                                        94 ,  77 ,  60 ,
                                        99 ,  99 ,  99 ,
                                        99 ,  99 ,  69 ,
                                        74 ,  99 ,  63 ,
                                        53 ,  71 ,  81 ,
                                        84 ,  82 ,  85 ,
                                        83 ,  86) ;
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('CHAMADA PLAPP04 (SCAN) COM CAR = ',CAR,' E CLASSE = ',CLASSE)
     (A , A , A , F(2)) ;
MAXINT = 2147483647 ;
MAXMANT = 7237008 ;
EXPMIN = - 75 ;
EXPMAX = 69 ;
MAXREAL = MAXMANT * 10 ** EXPMAX ;
LIMINT = 214748365 ;
LIMMANT = 723701 ;
TIPO = 0 ;
DO WHILE(CLASSE = 1 | CLASSE = 4) ; /* BRANCO OU % */
  DO WHILE(CLASSE = 1) ; /* BRANCO */
    CALL GETCAR ;
  END ;
  IF CLASSE = 4 /* % ? */
  THEN DO ;
         CALL GETCAR ;
         DO WHILE(CLASSE ¬= 4 & CLASSE ¬= 0) ; /* NEM % NEM EOF */
           CALL GETCAR ;
         END ;
         IF CLASSE ¬= 0 /* SE NAO ACABOU O ARQUIVO */
         THEN CALL GETCAR ;
       END ;
END ;
IF CLASSE = 2    /* PEGOU LETRA */
THEN DO ;    /* PEGOU LETRA */
       DO I = 1 TO 8 WHILE(CLASSE = 2 | CLASSE = 3) ;
         IDCHAR(I) = CAR ;
         CALL GETCAR ;
       END ;
       IF CLASSE = 2 | CLASSE = 3
       THEN DO ;
              ERRO = 17 ; /* IDENTIFICADOR > 8 CARACTERES */
              COLUNA = PTR ;
              CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
              DO WHILE(CLASSE = 2 | CLASSE = 3) ;
                CALL GETCAR ;
              END ;
            END ;
       DO I = I TO 8 ;
         IDCHAR(I) = ' ' ;
       END ;
       L = 1 ;
       U = 59 ;
       CK = '0'B ;
       DO WHILE(¬OK) ;
         IF U < L
         THEN DO ; /* IDENTIFICADOR NAO E PALAVRA RESERVADA */
                TIPO = 1 ;
                OK = '1'B ;
```

```
            END ;
         ELSE DO ;
                I = ( L + U ) / 2 ;
                IF IDENT = TABRES(I)
                THEN DO ; /* IDENTIFICADOR E PALAVRA RESERVADA */
                       TIPO = ATIPO(I) ;
                       CK = '1'B ;
                     END ;
                ELSE IF IDENT < TABRES(I)
                     THEN U = I - 1 ;
                     ELSE L = I + 1 ;
              END ;
       END ;
     END ;
ELSE IF CLASSE = 45
     THEN DO ; /* PEGOU LITERAL */
            TIPO = 4 ;
            DELIM = CAR ; /* SALVA DELIMITADOR DO LITERAL */
            CALL GETCAR ;
            DO I = 1 TO 256 WHILE(CAR ¬= DELIM & ¬EOF) ;
              STRING(I) = CAR ;
              CALL GETCAR ;
            END ;
            TAM = I - 1 ;
            IF EOF | CAR ¬= DELIM
            THEN DO ; /* STRING > 256 CARACTERES */
                   ERRO = 18 ; /* LITERAL > 256 CARACTERES */
                   COLUNA = PTR ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   DO WHILE(CLASSE ¬= 10 & ¬EOF) ; /* PROCURA ; */
                     CALL GETCAR ;
                   END ;
                 END ;
            ELSE CALL GETCAR ; /* STRING CORRETO */
          END ;
     ELSE
     IF CLASSE ¬= 3 & ¬EOF
     THEN DO ; /* PEGOU SIMBOLO ESPECIAL */
            TIPO = CLASSE ;
            CALL GETCAR ;
            IF (TIPO = 13 | TIPO = 32 | TIPO = 36 | TIPO = 37)
               & CLASSE = 38
            THEN DO ; /* PEGOU OPERADOR DUPLO */
                   TIPO = TIPO + 3 ;
                   CALL GETCAR ;
                 END ;
          END ;
     ELSE
     IF CLASSE ¬= 0 /* NAO ACABOU O ARQUIVO */
     THEN DO ; /* PEGOU DIGITO */
            ON FIXEDOVERFLOW
            BEGIN ;
              OVE = '1'B ;
              NINT = MAXINT ;
            END ;
            OVE = '0'B ;
            NINT = 0 ;
            EXP = 0 ;
            EXP1 = 0 ;
            NREAL = 1 ;
```

```
DO WHILE(CLASSE = 3) ; /* ENQUANTO FOR DIGITO */
  IF NINT < LIMINT
  THEN DO ;
         IF NINT >= LIMMANT
         THEN EXP1 = EXP1 + 1 ;
         DIG = CAR ;
         NINT = NINT * 10 + DIG ;
       END ;
  ELSE EXP = EXP + 1 ;
  CALL GETCAR ;
END ;
IF CLASSE = 0 | CLASSE ¬= 8 & CAR ¬= 'E'
THEN DO ; /* NAO E' CTE REAL */
       TIPO = 2 ;
       IF CLASSE ¬= 0 & INDEX(FIMNUM , CAR) = 0
       THEN DO ; /* CTE INTEIRA COM ERRO */
              ERRO = 19 ; /* ERRO DE SINTAXE EM CTE INT.*/
              COLUNA = PTR ;
              CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
              NINT = 1 ;
              DO WHILE(CLASSE¬=0 & INDEX(FIMNUM,CAR)=0) ;
                CALL GETCAR ;
              END ;
            END ;
       ELSE IF OVE | EXP > 0
            THEN DO ; /* CTE INTEIRA > LIMITE */
                   ERRO = 20 ; /* CTE INT. > 2147483647 */
                   COLUNA = PTR ;
                   CALL PLAPP01 ; /* IMP.MENSAGEM ERRO */
                   NINT = MAXINT ;
                 END ;
     END ; /* DESTE PONTO PULA PARA O FIM */
ELSE DO ; /* PEGOU CTE REAL */
       TIPO = 3 ;
       IF EXP1 ¬= 0
       THEN DO ;
              EXP = EXP + EXP1 ;
              NINT = NINT / 10 ** EXP1 ;
            END ;
       IF CLASSE = 8 /* PONTO ? */
       THEN DO ; /* PEGA PARTE FRACIONARIA */
              CALL GETCAR ;
              IF CLASSE ¬= 3 /* SE NAO E' DIGITO */
              THEN DO ; /* FALTA PARTE FRACIONARIA */
                     ERRO = 21 ;    /*FALTA PARTE FRACIO-*/
                     COLUNA = PTR ; /*NARIA EM CTE REAL  */
                     CALL PLAPP01 ; /* IMP.MENSAGEM ERRO */
                     EXP = 0 ;
                     DO WHILE(CLASSE ¬= 0 &
                              INDEX(FIMNUM , CAR) = 0) ;
                       CALL GETCAR ;
                     END ;
                   END ; /* DAQUI PULA P/AJUSTE DE EXP.*/
              ELSE DO WHILE(CLASSE = 3) ;
                     IF NINT < LIMMANT
                     THEN DO ;
                            DIG = CAR ;
                            NINT = NINT * 10 + DIG ;
                            EXP = EXP - 1 ;
                          END ;
```

```
                CALL GETCAR ;
              END ;
        END ;
IF CLASSE ¬= 0 & CAR = 'E'
THEN DO ; /* PEGA EXPOENTE */
       EXPLIDO = 0 ;
       CALL GETCAR ;
       IF CLASSE = 20 | CLASSE = 21
       THEN DO ; /* "+" OU "-" */
              EXPNEG = CLASSE = 21 ;
              CALL GETCAR ;
            END ;
       IF CLASSE ¬= 3 /* SE NAO E' DIGITO */
       THEN DO ; /* FALTA DIGITO NO EXPOENTE */
              ERRO = 22 ;     /*ERRO DE SINTAXE NO */
              COLUNA = PTR ; /*EXPOENTE CTE REAL  */
              CALL PLAPP01 ; /* IMP.MENSAGEM ERRO */
              EXP = 0 ;
              DO WHILE(CLASSE ¬= 0 &
                       INDEX(FIMNUM , CAR) = 0) ;
                CALL GETCAR ;
              END ;
            END ; /* DAQUI PULA P/AJUSTE DE EXP. */
       ELSE  /* LE EXPOENTE */
       DO ;
         DO WHILE(CLASSE = 3) ;
           IF EXPLIDO < LIMMANT
           THEN DO ;
                  DIG = CAR ;
                  EXPLIDO = EXPLIDO * 10 + DIG ;
                END ;
           CALL GETCAR ;
         END ;
         IF CLASSE ¬= 0 & INDEX(FIMNUM , CAR) = 0
         THEN DO ; /* EXPOENTE NAO TERMINOU LEGAL */
                ERRO = 22 ;     /*ERRO SINTAXE NO  */
                COLUNA = PTR ; /*EXPOENTE CTE REAL*/
                CALL PLAPP01 ; /*IMP.MENSAGEM ERRO*/
                EXP = 0 ;
                DO WHILE(CLASSE ¬= 0 &
                         INDEX(FIMNUM,CAR) = 0) ;
                  CALL GETCAR ;
                END ;
              END ;
         ELSE IF EXPNEG
              THEN EXP = EXP - EXPLIDO ;
              ELSE EXP = EXP + EXPLIDO ;
       END ;
     END ;
IF CLASSE ¬= 0 & INDEX(FIMNUM , CAR) = 0
THEN DO ; /* CTE REAL SEM EXPOENTE NAO TERMINOU   */
          /* LEGAL                                */
       ERRO = 23 ;     /*ERRO SINTAXE PARTE FRACIO-*/
       COLUNA = PTR ; /*NARIA CTE REAL.ASSUMIDO 1.*/
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       DO WHILE(CLASSE ¬= 0 &
                INDEX(FIMNUM , CAR) = 0) ;
         CALL GETCAR ;
       END ;
     END ;
```

```
                    /* AJUSTE DE EXPOENTE */
                    ELSE IF EXP < EXPMIN
                         THEN DO ; /* UNDERFLOW CTE REAL */
                                 ERRO = 24 ; /* UNDERFLOW CTE REAL */
                                 COLUNA = PTR ;
                                 CALL PLAPP01 ; /* IMP. MENSAGEM ERRO */
                                 NREAL = 0. ;
                               END ;
                         ELSE IF EXP > EXPMAX
                              THEN DO ; /* OVERFLOW CTE REAL */
                                      ERRO = 25 ; /* OVERFLOW CTE REAL */
                                      COLUNA = PTR ;
                                      CALL PLAPP01 ; /*IMP.MENSAG.ERRO*/
                                      NREAL = MAXREAL ;
                                    END ;
                              ELSE DO ;
                                      IF NINT > MAXMANT
                                      THEN DO ;
                                              NINT = (NINT + 1) / 2 ;
                                              NREAL = NINT * 2 ;
                                           END ;
                                      ELSE NREAL = NINT ;
                                      IF EXP ¬= 0
                                      THEN NREAL = NREAL * 10 ** EXP ;
                                    END ;
                  END ;
          END ;
1GETCAR : PROC ;
   /*****************************************************************/
   /* ESTA PROCEDURE TEM COMO FUNCOES :                             */
   /* - EXTRAIR O PROXIMO CARACTER DA SEQUENCIA DE ENTRADA, COLOCANDO-*/
   /*   O NA VARIAVEL 'CAR'.                                        */
   /* - ATRIBUIR A VARIAVEL 'CLASSE' A CLASSE DO CARACTER LIDO, DE A- */
   /*   CORDO COM A TABELA ABAIXO (CL = CLASSE E CT = CARACTER) :    */
   /*   CL CT        CL CT                 CL CT         CL CT      */
   /*    1 BRANCO    10 ;                  20 +          32 ¬       */
   /*    2 LETRA     11 (                  21 -          36 >       */
   /*    3 DIGITO    12 )                  22 /          37 <       */
   /*    4 %         13 :                  23 *          38 =       */
   /*    8 .         14 ABRE COLCHETE  #   30 &          45 "       */
   /*    9 ,         15 FECHA COLCHETE !   31 |          46 '       */
   /* - POSICIONAR O PROXIMO CARACTER A SER EXTRAIDO.               */
   /* - A SEQUENCIA DE ENTRADA E CONSIDERADA COMO SENDO DA COLUNA 1 A */
   /*   COLUNA 72 (INCLUSIVES) DO CARTAO. AO SER EXTRAIDO O ULTIMO CA-*/
   /*   RACTER DE UM CARTAO, ESTA PROCEDURE SE ENCARREGA DE LER OUTRO */
   /*   CARTAO, BEM COMO IMPRIMI-LO.                                */
   /* - AO SER DETETADO O FIM DO ARQUIVO QUE CONTEM O PROGRAMA DO USU-*/
   /*   ARIO E' ATRIBUIDO A VARIAVEL 'EOF' O VALOR '1'B, A VARIAVEL  */
   /*   'CLASSE' RETORNA COM VALOR 0 E 'CAR' PERMANECE INALTERADO.   */
   /* - AO SER DETETADO UM CARACTER QUE NAO PERTENCA A NENHUMA DAS    */
   /*   CLASSES DESCRITAS ACIMA, SERA EMITIDA UMA MENSAGEM DE ERRO E  */
   /*   O CARACTER SERA IGNORADO.                                   */
   /* OBS. - NA PRIMEIRA CHAMADA DESTA PROCEDURE, PTR DEVE TER SIDO I-*/
   /*        NICIALIZADO COM 72.                                    */
   /*****************************************************************/
   DCL TAB(21)  BIN FIXED(15,0) STATIC
                              INIT( 4 ,  8 ,  9 , 10 , 11 , 12 , 13 ,
                                   14 , 15 , 20 , 21 , 22 , 23 , 30 ,
                                   31 , 32 , 36 , 37 , 38 , 45 , 45) ;
   ON ENDFILE(SCARDS) GOTO FIM ;
```

```
   CLASSE = 0 ;
   DO WHILE(CLASSE = 0) ;
     IF PTR = 72
     THEN DO ;
            READ FILE(SCARDS) INTO(CARD) ;
            IF CARD(1) ='$'
            THEN GOTO FIM ; /* FIM DE ARQUIVO */
            PUT FILE(SPRINT) EDIT(CARD)(COL(20) , (80)A(1)) ;
            PTR = 0 ;
          END ;
     PTR = PTR + 1 ;
     CAR = CARD(PTR) ;
     IF CAR = ' '
     THEN CLASSE = 1 ;
     ELSE IF (CAR >= 'A' & CAR <= 'I') | (CAR >= 'J' & CAR <= 'R') |
             (CAR >= 'S' & CAR <= 'Z')
          THEN CLASSE = 2 ;
          ELSE IF CAR >= '0' & CAR <= '9'
               THEN CLASSE = 3 ;
               ELSE DO ;
                      CLASSE = INDEX('%.,;():#!+-/*&|¬><="''',CAR) ;
                      IF CLASSE ¬= 0
                      THEN CLASSE = TAB(CLASSE) ;
                      ELSE DO ;
                             ERRO = 26 ; /* CARACTER INVALIDO.IGNORADO*/
                             COLUNA = PTR ;
                             CALL PLAPP01 ; /* IMPRIME MENSAGEM ERRO  */
                           END ;
                    END ;
   END ;
   RETURN ;
FIM :
   EOF = '1'B ;
END GETCAR ;
   IF TRACE(1)
   THEN PUT FILE(SPRINT) SKIP EDIT
        ('PLAPP04 (SCAN) RETORNOU COM TIPO = ' , TIPO) (A , F(2)) ;
END PLAPP04 ;
```

```
/* PLAPP05 - ROTINA PARA MANIPULAR COM A TABELA DE SIMBOLOS                */
PLAPP05 : PROC ;
/*************************************************************************/
/* ESTA PROCEDURE TEM 3 ENTRY-POINTS ('PLAPP05' , 'PLAPP06' E             */
/* 'PLAPP07') E E' USADA PARA MANIPULAR COM A TABELA DE SIMBOLOS DO       */
/* COMPILADOR.                                                            */
/* ENTRY-POINT PLAPP05 (BUSCA) :                                          */
/*   RECEBE EM 'IDNOME' O IDENTIFICADOR.                                  */
/*   DEVOLVE A VARIAVEL BOOLEANA 'ACHOU' IGUAL A 'TRUE' CASO O IDEN-      */
/*   TIFICADOR ESTEJA NA TABELA, E 'FALSE' EM CASO CONTRARIO.             */
/*   NO PRIMEIRO CASO, DEVOLVE TAMBEM NA VARIAVEL 'IND' A POSICAO DO      */
/*   IDENTIFICADOR NA TABELA. NO SEGUNDO CASO, DA' MENSAGEM DE ERRO,      */
/*   INSERE O IDENTIFICADOR NA TABELA E DEVOLVE NA VARIAVEL 'IND' A       */
/*   POSICAO DO IDENTIFICADOR NA TABELA.                                  */
/* ENTRY-POINT PLAPP06 (INSERE) :                                         */
/*   RECEBE EM 'IDNOME' O IDENTIFICADOR.                                  */
/*   RECEBE EM 'BLOCO' O NUMERO DO BLOCO CORRENTE.                        */
/*   DEVOLVE A VARIAVEL BOOLEANA 'ACHOU' IGUAL A 'TRUE' CASO O IDEN-      */
/*   TIFICADOR ESTEJA NA TABELA, E 'FALSE' EM CASO CONTRARIO.             */
/*   NO CASO DE INSERCAO O IDENTIFICADOR SO' E' CONSIDERADO JA' PRE-      */
/*   SENTE NA TABELA SE OCORRER TAMBEM A IGUALDADE DO CONTEUDO DO         */
/*   CAMPO 'NIVEL' COM O CONTEUDO DA VARIAVEL 'BLOCO'. E NESTE CASO       */
/*   E' DADA UMA MENSAGEM DE ERRO. CASO CONTRARIO (IDENTIFICADOR AU-      */
/*   SENTE), O IDENTIFICADOR E' INSERIDO NA TABELA E E' DEVOLVIDO EM      */
/*   'IND' SUA POSICAO NA TABELA.                                         */
/* ENTRY-POINT PLAPP07 (DELETA) :                                         */
/*   NA REALIDADE NAO PRECISAVA SER UM ENTRY-POINT DESTA PROCEDURE.       */
/*   ESTA' AQUI APENAS PARA GRUPAR AS ROTINAS DE MANIPULACAO DA TABE-     */
/*   LA DE SIMBOLOS.                                                      */
/*   RECEBE EM 'BLOCO' O NUMERO DO BLOCO QUE ACABOU DE SER ANALISADO      */
/*   E ELIMINA DA TABELA DE SIMBOLOS TODOS OS IDENTIFICADORES CUJO        */
/*   CONTEUDO DO CAMPO 'NIVEL' SEJA IGUAL AO DA VARIAVEL 'BLOCO', A-      */
/*   TUALIZANDO OS LINKS E A VARIAVEL 'AVAIL'.                            */
/* A ESTRUTURA DA TABELA DE SIMBOLOS E' DESCRITA A SEGUIR :               */
/* TODOS OS IDENTIFICADORES INSERIDOS NA TABELA DE SIMBOLOS QUE PRO-       */
/* DUZIREM O MESMO VALOR PARA A FUNCAO HASH, PERTENCERAO A UMA LISTA      */
/* ENCADEADA, QUE SERA' APONTADA PELO ELEMENTO DO VETOR 'CABEC',          */
/* CUJO INDICE E' O VALOR DA FUNCAO HASH. TAL LISTA E' ENCADEADA EM       */
/* ORDEM INVERTIDA, ISTO E', O ELEMENTO DE 'CABEC' APONTA PARA O UL-      */
/* TIMO IDENTIFICADOR INSERIDO NESTA LISTA.                               */
/* DOS CAMPOS DA TABELA DE SIMBOLOS, APENAS OUTROS 3 ALEM DE 'CABEC'      */
/* SAO NECESSARIOS PARA A MANIPULACAO :                                   */
/* 'NOME' - CONTEM O IDENTIFICADOR.                                       */
/* 'LINK' - APONTA PARA O PROXIMO ELEMENTO DA LISTA.                      */
/* 'NIVEL' - NUMERO DO BLOCO CORRENTE QUANDO DA INSERCAO DO IDENTIFI-     */
/*           CADOR NA TABELA DE SIMBOLOS.                                 */
/* A VARIAVEL 'AVAIL' APONTA PARA A PROXIMA POSICAO DISPONIVEL NA TA-     */
/* BELA DE SIMBOLOS.                                                      */
/* UM APONTADOR COM VALOR ZERO INDICA FIM DA LISTA.                       */
/* A FUNCAO HASH UTILIZADA CONSISTE NO OU-EXCLUSIVO DOS BYTES QUE         */
/* COMPOEM O IDENTIFICADOR, DESPREZANDO-SE AO FINAL OS 2 BITS MAIS A      */
/* ESQUERDA.                                                              */
/*************************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* ROTINA IMPRESSAO TAB.DE SIMBOLOS */
$CONTINUE WITH PLAMP02 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO */
```

```
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL INS          AUTOMATIC BIT(1) INIT('1'B) ;
 DCL HOME         BIN FIXED(15,0) STATIC ;
 DCL AVAIL        BIN FIXED(15,0) INIT(1) STATIC ;
 DCL ADDR         BUILTIN ;
 DCL BOOL         BUILTIN ;
 DCL PLAPA03      ENTRY(CHAR (8)) RETURNS (BIN FIXED (15,0));
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('CHAMADA PLAPP05 (BUSCA NA TABELA DE SIMBOLOS) COM IDNOME= ',
       IDNOME)(A , A) ;
 INS = '0'B ;
1PLAPP06 : ENTRY ; /* INSERE */
 IF TRACE(1) & INS
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('CHAMADA PLAPP06 (INSERCAO NA TABELA DE SIMBOLOS) COM ' ,
       'IDNOME = ' , IDNOME , ' E BLOCO = ' , BLOCO) ((4)A , F(3)) ;
 HOME = PLAPA03(IDNOME) ; /* CALCULA FUNCAO HASH */
 ACHOU = '0'B ;
 IND = CABEC(HOME) ;
 DO WHILE(IND ¬= 0 & ¬ACHOU) ;
   IF NOME(IND) = IDNOME
   THEN ACHOU = '1'B ;
   ELSE IND = LINK(IND) ;
 END ;
 IF INS & ACHOU & BLOCO = NIVEL(IND)
 THEN DO ;
        ERRO = 47 ; /* IDENTIFICADOR DECLARADO MAIS DE UMA VEZ. */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
      END ;
 ELSE IF INS | ¬ACHOU
      THEN DO ;
             IF ¬INS
             THEN DO ;
                    ERRO = 48 ; /* IDENTIFICADOR NAO DECLARADO. */
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                  END ;
             ELSE ACHOU = '0'B ;
             IF AVAIL = 300
             THEN DO ;
                    ERRO = 49 ; /* PLAPP06 - RESTRICAO DE IMPLE- */
                                /* MENTACAO - NUMERO MAXIMO DE   */
                                /* IDENTIFICADORES NOS BLOCOS    */
                                /* ATIVOS FOI PASSADO.           */
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                    ERRO = 41 ;  /* COMPILACAO ABORTADA DEVIDO */
                    COLUNA = 1 ; /* A ERRO GRAVE.              */
                    CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                    STOP ;
```

```
            END ;
         IND = AVAIL ;
         LINK(IND) = CABEC(HOME) ;
         CABEC(HOME) = IND ;
         NIVEL(IND) = BLOCO ;
         NOME(IND) = IDNOME ;
         AVAIL = AVAIL + 1 ;
       END ;
IF TRACE(1)
THEN IF INS
     THEN PUT FILE(SPRINT) SKIP EDIT
                ('PLAPP06 (INSERCAO NA TABELA DE SIMBOLOS) RE' ,
                 'TORNOU COM ACHOU = ', ACHOU , ' IND = ' , IND ,
                 ' E AVAIL = ' , AVAIL)( A , A , B(1) , A , F(3),
                  A , F(3)) ;
     ELSE PUT FILE(SPRINT) SKIP EDIT
                 ('PLAPP05 (BUSCA NA TABELA DE SIMBOLOS) RETOR',
                 'NOU COM ACHOU = ' , ACHOU , ' IND = ' , IND)
                 (A , A , B(1) , A , F(3)) ;
IF TRACE(2)
THEN CALL PLAPP02(AVAIL) ; /* IMPRIME TABELA DE SIMBOLOS */
RETURN ;
1PLAPA03 : PROC(IDENT) RETURNS(BIN FIXED(15,0)) ; /* FUNCAO HASH */
  DCL IDENT           CHAR(8) ;
  DCL NOME            CHAR(8) BASED(PTR) ;
  DCL PTR             POINTER STATIC ;
  PTR = ADDR(ELEMBOL) ;
  DCL ELEMBOL         BIT(64) UNAL STATIC ;
  DCL A32             BIT(32) UNAL DEFINED ELEMBOL ;
  DCL B32             BIT(32) UNAL DEFINED ELEMBOL POS(33) ;
  DCL A16             BIT(16) UNAL DEFINED ELEMBOL ;
  DCL B16             BIT(16) UNAL DEFINED ELEMBOL POS(17) ;
  DCL A8              BIT(8) UNAL DEFINED ELEMBOL ;
  DCL B8              BIT(8) UNAL DEFINED ELEMBOL POS(9) ;
  DCL BIT6            BIT(6) UNAL DEFINED ELEMBOL POS(3) ;
  DCL 1 VAL UNAL STATIC ,
        2 ZEROS       BIT(10) INIT('0000000000'B) ,
        2 VALBIT      BIT(6) ;
  DCL PTVAL           POINTER STATIC ;
  PTVAL = ADDR(VAL) ;
  DCL VALOR           BIN FIXED(15,0) BASED(PTVAL) ;
  IF TRACE(2)
  THEN PUT FILE(SPRINT) SKIP EDIT
       ('CHAMADA PLAPA03 (HASH) COM IDENT = ', IDENT) (A,A) ;
  NOME = IDENT ;
  A32 = BOOL(A32 , B32 , '0110'B) ;
  A16 = BOOL(A16 , B16 , '0110'B) ;
  A8 = BOOL(A8 , B8 , '0110'B) ;
  VALBIT = BIT6 ;
  IF TRACE(2)
  THEN PUT EDIT('PLAPA03 (HASH) RETORNOU COM VALOR = ' , VALOR)
               (COL(1) , A , F(3)) ;
  RETURN(VALOR) ;
END PLAPA03 ;
1PLAPP07 : ENTRY ; /* DELETA */
DCL I       BIN FIXED(15,0) STATIC ;
DCL PT      BIN FIXED(15,0) STATIC ;
DCL PTA     BIN FIXED(15,0) STATIC ;
DCL PRIM    BIT (1) STATIC ;
IF TRACE(1)
```

```
THEN PUT FILE(SPRINT) SKIP EDIT
            ('CHAMADA PLAPP07 (DELETA SIMBOLOS DA TABELA DE SIM',
             'BOLOS) COM BLOCO = ',BLOCO)(A , A , F(3));
PRIM = '1'B ;
DO I = 0 TO 63 ;
  IF CABEC(I) ¬= 0
  THEN DO ;
         PTA = CABEC(I) ;
         PT = PTA ;
         DO WHILE ( PT ¬= 0 & NIVEL(PT) = BLOCO ) ;
           IF IDTIPO(PT) = 30    /* LABEL? */
           THEN IF PTCTE(PT) = -1
                THEN DO ; /* REFERENCIADO E NAO DEFINIDO */
                       IF PRIM
                       THEN DO ;
                              PRIM = '0'B ;
                              /* LABEL REFERENCIADO MAS NAO */
                              /* DEFINIDO NO BLOCO          */
                              ERRO = 133 ;
                              COLUNA = 1 ;
                              CALL PLAPP01 ;
                              PUT SKIP ;
                            END ;
                       PUT FILE(SPRINT) EDIT(NOME(PT)) (X(2) , A) ;
                     END ;
           PTA = PT ;
           PT = LINK(PT) ;
         END ;
         IF PTA ¬= PT
         THEN DO ;
                CABEC(I) = PT ;
                IF AVAIL > PTA
                THEN AVAIL = PTA ;
              END ;
       END ;
END ;
IF TRACE(2)
THEN CALL PLAPP02(AVAIL) ; /* IMPRIME TABELA DE SIMBOLOS */
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('PLAPP07 (DELETA SIMBOLOS DA TABELA DE SIMBOLOS) RETORNOU.')
     (A) ;
END PLAPP05 ;
```

```
/* PLAPP08 - RECONHECEDOR DE BLOCO                                    */
PLAPP08 : PROC RECURSIVE ;
/*********************************************************************/
/* ESTA PROCEDURE RECONHECE BLOCO. E' CHAMADA COM O PRIMEIRO         */
/* ELEMENTO DO BLOCO E RETORNA COM TIPO = 0 (FIM DE ARQUIVO)         */
/* OU COM O TIPO CORRESPONDENTE AO SIMBOLO QUE SEGUE O "END"         */
/* QUE ENCERRA O BLOCO.                                              */
/*********************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* RECONHECE DECLARACAO DE CONSTANTE */
$CONTINUE WITH PLAMP09 RETURN
1 /* RECONHECE DECLARACAO DE VARIAVEL */
$CONTINUE WITH PLAMP10 RETURN
1 /* RECONHECE CABECALHO DE PROCEDURE */
$CONTINUE WITH PLAMP11 RETURN
1 /* RECONHECE CABECALHO DE FUNCAO */
$CONTINUE WITH PLAMP12 RETURN
1 /* MANIPULACAO DAS TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECCNHECEDOR DE COMANDOS */
$CONTINUE WITH PLAMP17 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM E PROXDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO TABPARM E PRXPARM */
$CONTINUE WITH PLAMV09 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1, REG1, QFIX1, QUADRA1, PTOBJ1 E LC1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL TIPANT          AUTOMATIC BIN FIXED(15,0) ;
 DCL FIMPROCS        AUTOMATIC BIN FIXED(15,0) ;
 DCL GUARDPT(11)     AUTOMATIC BIN FIXED(15,0) ;
 DCL TIPLIM          BIN FIXED(15,0) STATIC ;
 DCL I               BIN FIXED(15,0) STATIC ;
 DCL PILHA(16)       BIN FIXED(15,0) STATIC ;
 DCL TOPO            BIN FIXED(15,0) STATIC ;
 DCL LAB(5)          AUTOMATIC LABEL INIT(LAB1,LAB2,LAB3,LAB4,LAB5) ;
```

```
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
        ('PLAPP08 (RECONHECEDOR DE BLOCO) INICIOU COM BLOCO = ' ,
         BLOCO , ' PROXINT = ' , PROXINT , ' PROXRE = ' , PROXRE ,
         ' ULTALFA = ' , ULTALFA , ' PROXDIM = ' , PROXDIM ,
         ' E PRXPARM = ' , PRXPARM)((6)(A , F(4))) ;
/***** GUARDPT(1) = PROXINT ;     *****/
/***** GUARDPT(2) = PROXRE ;      *****/
/***** GUARDPT(3) = ULTALFA ;     *****/
GUARDPT(1) = PROXDIM ;
GUARDPT(2) = PRXPARM ;
/* ESTADO = 0 */
IF TIPO ¬= 0 & ( TIPO < 51 | TIPO > 56 )
THEN DO ; /* NAO E' FIM DE ARQUIVO NEM "LABEL", "CONST", "VAR", */
          /* "PROC", "FUNCTION" OU "BEGIN"                      */
       ERRO = 13 ; /* SIMBOLO INVALIDO PARA INICIO DE BLOCO */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       DO WHILE(TIPO ¬= 0 & (TIPO < 51 | TIPO > 56)) ;
         CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
       END ;
       ERRO = 14 ; /* SIMBOLOS ANTERIORES FORAM IGNORADOS. */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
TIPANT = 0 ;
DO WHILE(TIPO ¬= 56 & TIPO ¬= 0) ; /*TIPO ¬= BEGIN E NAO FIM ARQ.*/
  IF TIPO <= 53 & TIPANT >= TIPO
  THEN DO ;
         ERRO = TIPO - 22 ; /* DECLARACAO DE LABEL (ERRO 29),    */
                            /* CONSTANTE (ERRO 30) OU VARIAVEL   */
         COLUNA = PTR - 1 ; /* (ERRO 31) EM DUPLICIDADE OU FORA  */
                            /* DE SEQUENCIA.                     */
         CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
         TIPLIM = TIPO + 1 ;
         DO WHILE(TIPO ¬= 0 & (TIPO < TIPLIM | TIPO > 56)) ;
           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
         END ;
         ERRO = 14 ; /* SIMBOLOS ANTERIORES FORAM IGNORADOS */
         COLUNA = PTR - 1 ;
         CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       END ;
  ELSE DO ;
         GOTO LAB(TIPO - 50) ;
1LAB1 : /* PEGOU DECLARACAO DE LABEL - ESTADO = 1 */
  TIPANT = TIPO ;
  DO WHILE(TIPO ¬= 0 & TIPO ¬= 10) ;/*TIPO ¬= ; E NAO FIM DE ARQ.*/
    CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
    IF TIPO = 1
    THEN DO ; /* PEGOU IDENTIFICADOR - ESTADO = 2 */
           IDNOME = IDENT ;
           CALL PLAPP06 ; /* INSERE IDENT. NA TABELA SIMBOLOS */
           IF ¬ACHOU
           THEN DO ;
                  IDTIPO(IND) = 30 ; /* COMPLETA TAB.SIMBOLOS */
                                     /* TIPO = LABEL          */
                  PTCTE(IND) = 0 ; /* INDICA LABEL AINDA NAO  */
                                   /* DEFINIDO                */
                END ;
           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
```

```
          END ;
     ELSE DO ;
            IF TIPO = 9 | TIPO = 10 | TIPO = 0
            THEN DO ; /* , OU ; OU FIM DE ARQUIVO */
                   ERRO = 32 ; /* FALTA NOME DE LABEL */
                   COLUNA = PTR - 1 ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                 END ;
            ELSE DO ;
                   ERRO = 27 ; /* LABEL INVALIDO */
                   COLUNA = PTR - 1 ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                 END ;
          END ;
     IF TIPO ¬= 9 & TIPO ¬= 10 & TIPO ¬= 0
     THEN DO ; /* NAO E , NEM ; NEM FIM DE ARQUIVO */
            ERRO = 28 ; /* , OU ; ESPERADOS */
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPKIME MENSAGEM DE ERRO */
            DO WHILE(TIPO ¬= 9 & TIPO ¬= 10 & TIPO ¬= 0) ;
              /* PULA TUDO ATE , OU ; OU FIM DE ARQUIVO */
              CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
            END ;
            ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
          END ;
   END ;
   IF TIPO = 10 /* ; ? */
   THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
   GOTO ENDLAB ; /* FIM DA ANALISE DE DECLARACAO DE LABEL */
1LAB2 : /* PEGOU DECLARACAO DE CONSTANTE - ESTADO = 4 */
   TIPANT = TIPO ;
   CALL PLAPP09 ; /* RECONHECE DECLARACAO DE CONSTANTE */
   GOTO ENDLAB ; /* FIM DA ANALISE DE DECLARACAO DE CONSTANTE */
1LAB3 : /* PEGOU DECLARACAO DE VARIAVEL - ESTADO = 9 */
   TIPANT = TIPO ;
   CALL PLAPP10 ; /* RECONHECE DECLARACAO DE VARIAVEL */
   GOTO ENDLAB ; /* FIM DA ANALISE DE DECLARACAO DE VARIAVEL */
1LAB4 : /* PEGOU DECLARACAO DE PROCEDURE - ESTADO = XX */
   IF GERA & TIPANT ¬= 54
   THEN DO ; /* PRIMEIRA PROCEDURE DO BLOCO */
          /* GERA INSTRUCOES P/PULAR PROCEDURES E FUNCOES */
          IF PROXINT > LSKINT
          THEN DO ; /* PLAPP08 - RESTRICAO DE IMPLEMENTACAO - NU-*/
                    /* MERO MAXIMO PERMITIDO DE CTES INTEIRAS NO */
                    /* PROGRAMA FOI ULTRAPASSADO.                */
                 ERRO = 137 ;
                 COLUNA = PTR - 1 ;
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                 ERRO = 41 ; /* COMPILACAO ABORTADA */
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                 STOP ;
               END ;
          FIMPROCS = PROXINT ;
          PROXINT = PROXINT + 1 ;
          CALL PLAPP34(COSET,0,0,0,3,0,FIMPROCS) ;
          CALL PLAPP34(CODESV,0,1,0,CCQQR,0,0) ;
        END ;
```

```
TIPANT = 54 ;
GUARDPT(3) = CNTINT ;
GUARDPT(4) = CNTREAL ;
GUARDPT(5) = CNTBOOL ;
GUARDPT(6) = CNTCHAR ;
GUARDPT(7) = TEMINT ;
GUARDPT(8) = TEMREAL ;
GUARDPT(9) = TEMBOOL ;
GUARDPT(10) = TEMCHAR ;
GUARDPT(11) = PROXINT + 1 ; /* END DA AREA QUE CONTERA AS CTES*/
                            /* C/TAMANHOS DAS AREAS DA PROCED.*/
CNTINT = LIVINT ;
CNTREAL = LIVREAL ;
CNTBOOL = LIVBOOL ;
CNTCHAR = LIVCHAR ;
TEMINT = 0 ;
TEMREAL = 0 ;
TEMBOOL = 0 ;
TEMCHAR = 0 ;
CALL PLAPP11 ; /* RECONHECEDOR DE CABECALHO DE PROCEDURE */
CALL PLAPP08 ; /* RECONHECEDOR DE BLOCO */
IF TIPO ¬= 10 & TIPO ¬= 0 /* NAO E' ";" NEM EOF */
THEN DO ;
       ERRO = 74 ; /* SIMBOLO INVALIDO. ESPERADO ";" */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       DO WHILE(TIPO ¬= 10 & TIPO ¬= 0) ;
         /* ENQUANTO NAO FOR ";" NEM FIM DE ARQUIVO */
         CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
       END ;
       ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
IF GERA
THEN DO ;
       /* GERA EPILOGO DA PROCEDURE */
       I = GUARDPT(11) ;
       CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
       /* LIBERA AREAS DE VARS. E TEMPS. DA PROCEDURE */
       CALL PLAPP34(COSUBI,0,3,0,I,0,3) ;
       VALINT(I) = CNTINT - LIVINT ;
       CNTINT = GUARDPT(3) ;
       I = I + 1 ;
       CALL PLAPP34(COSUBI,0,4,0,I,0,4) ;
       VALINT(I) = CNTREAL - LIVREAL ;
       CNTREAL = GUARDPT(4) ;
       I = I + 1 ;
       CALL PLAPP34(COSUBI,0,5,0,I,0,5) ;
       VALINT(I) = CNTBOOL - LIVBOOL ;
       CNTBOOL = GUARDPT(5) ;
       I = I + 1 ;
       CALL PLAPP34(COSUBI,0,6,0,I,0,6) ;
       VALINT(I) = CNTCHAR - LIVCHAR ;
       CNTCHAR = GUARDPT(6) ;
       I = I + 1 ;
       CALL PLAPP34(COSUBI,0,7,0,I,0,7) ;
       VALINT(I) = TEMINT ;
       TEMINT = GUARDPT(7) ;
       I = I + 1 ;
```

```
            CALL PLAPP34(COSUBI,0,8,0,I,0,8) ;
            VALINT(I) = TEMREAL ;
            TEMREAL = GUARDPT(8) ;
            I = I + 1 ;
            CALL PLAPP34(COSUBI,0,9,0,I,0,9) ;
            VALINT(I) = TEMBOOL ;
            TEMBOOL = GUARDPT(9) ;
            I = I + 1 ;
            CALL PLAPP34(COSUBI,0,10,0,I,0,10) ;
            VALINT(I) = TEMCHAR ;
            TEMCHAR = GUARDPT(10) ;
            /* GERA INST.P/RETORNO DA PROCEDURE */
            CALL PLAPP34(COSET,0,0,0,3,BLOCO+2,-1) ;
            CALL PLAPP34(CODESV,0,1,0,CCQQR,0,1) ;
            /* COMPLETA INST. DE DESVIO P/PULAR PROCEDURE */
            VALINT(FIMPROCS) = CINST + LC1 ;
         END ;
   IF TIPO = 10 /* ; ? */
   THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
   GOTO ENDLAB ; /* FIM DA ANALISE DE DECLARACAO DE PROCEDURE */
LLAB5 : /* PEGOU DECLARACAO DE FUNCAO - ESTADO = XX */
   IF GERA & TIPANT ¬= 54
   THEN DO ; /* PRIMEIRA FUNCAO DO BLOCO */
            /* GERA INSTRUCOES P/PULAR PROCEDURES E FUNCOES */
            IF PROXINT > LSKINT
            THEN DO ; /* PLAPP08 - RESTRICAO DE IMPLEMENTACAO - NU-*/
                      /* MERO MAXIMO PERMITIDO DE CTES INTEIRAS NO */
                      /* PROGRAMA FOI ULTRAPASSADO.                */
                 ERRO = 137 ;
                 COLUNA = PTR - 1 ;
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                 ERRO = 41 ; /* COMPILACAO ABORTADA */
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                 STOP ;
               END ;
            FIMPROCS = PROXINT ;
            PROXINT = PROXINT + 1 ;
            CALL PLAPP34(COSET,0,0,0,3,0,FIMPROCS) ;
            CALL PLAPP34(CODESV,0,1,0,CCQQR,0,0) ;
         END ;
   TIPANT = 54 ;
   GUARDPT(3) = CNTINT ;
   GUARDPT(4) = CNTREAL ;
   GUARDPT(5) = CNTBOOL ;
   GUARDPT(6) = CNTCHAR ;
   GUARDPT(7) = TEMINT ;
   GUARDPT(8) = TEMREAL ;
   GUARDPT(9) = TEMBOOL ;
   GUARDPT(10) = TEMCHAR ;
   GUARDPT(11) = PROXINT + 1 ; /* END DA AREA QUE CONTERA AS CTES*/
                               /* C/TAMANHOS DAS AREAS DA FUNCAO */
   CNTINT = LIVINT ;
   CNTREAL = LIVREAL ;
   CNTBOOL = LIVBOOL ;
   CNTCHAR = LIVCHAR ;
   TEMINT = 0 ;
   TEMREAL = 0 ;
   TEMBOOL = 0 ;
   TEMCHAR = 0 ;
   CALL PLAPP12 ; /* RECONHECEDOR DE CABECALHO DE FUNCAO */
```

```
   CALL PLAPP08 ; /* RECONHECEDOR DE BLOCO */
   IF TIPO ¬= 10 & TIPO ¬= 0 /* NAO E' ";" NEM EOF */
   THEN DO ;
           ERRO = 74 ; /* SIMBOLO INVALIDO. ESPERADO ";" */
           COLUNA = PTR - 1 ;
           CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
           DO WHILE(TIPO ¬= 10 & TIPO ¬= 0) ;
             /* ENQUANTO NAO FOR ";" NEM FIM DE ARQUIVO */
             CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
           END ;
           ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
           COLUNA = PTR - 1 ;
           CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
         END ;
   IF GERA
   THEN DO ;
           /* GERA EPILOGO DA FUNCAO */
           I = GUARDPT(11) ;
           CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
           /* LIBERA AREAS DE VARS. E TEMPS. DA FUNCAO */
           CALL PLAPP34(COSUBI,0,3,0,I,0,3) ;
           VALINT(I) = CNTINT - LIVINT ;
           CNTINT = GUARDPT(3) ;
           I = I + 1 ;
           CALL PLAPP34(COSUBI,0,4,0,I,0,4) ;
           VALINT(I) = CNTREAL - LIVREAL ;
           CNTREAL = GUARDPT(4) ;
           I = I + 1 ;
           CALL PLAPP34(COSUBI,0,5,0,I,0,5) ;
           VALINT(I) = CNTBOOL - LIVBOOL ;
           CNTBOOL = GUARDPT(5) ;
           I = I + 1 ;
           CALL PLAPP34(COSUBI,0,6,0,I,0,6) ;
           VALINT(I) = CNTCHAR - LIVCHAR ;
           CNTCHAR = GUARDPT(6) ;
           I = I + 1 ;
           CALL PLAPP34(COSUBI,0,7,0,I,0,7) ;
           VALINT(I) = TEMINT ;
           TEMINT = GUARDPT(7) ;
           I = I + 1 ;
           CALL PLAPP34(COSUBI,0,8,0,I,0,8) ;
           VALINT(I) = TEMREAL ;
           TEMREAL = GUARDPT(8) ;
           I = I + 1 ;
           CALL PLAPP34(COSUBI,0,9,0,I,0,9) ;
           VALINT(I) = TEMBOOL ;
           TEMBOOL = GUARDPT(9) ;
           I = I + 1 ;
           CALL PLAPP34(COSUBI,0,10,0,I,0,10) ;
           VALINT(I) = TEMCHAR ;
           TEMCHAR = GUARDPT(10) ;
           /* GERA INST.P/RETORNO DA FUNCAO */
           CALL PLAPP34(COSET,0,0,0,3,BLOCO+2,-1) ;
           CALL PLAPP34(CODESV,0,1,0,CCQQR,0,1) ;
           /* COMPLETA INST. DE DESVIO P/PULAR A FUNCAO */
           VALINT(FIMPROCS) = CINST + LC1 ;
         END ;
   IF TIPO = 10 /* ; ? */
   THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
FENDLAB ; /*                                                    ****/
```

```
   IF TIPO ¬= 0 & ( TIPO < 51 | TIPO > 56 )
   THEN DO ;
          ERRO = 40 ;          /* SIMBOLO INVALIDO. ESPERADO        */
          COLUNA = PTR - 1 ; /* "LABEL", "CONST", "VAR", "PROC", */
                               /* "FUNCTION" OU "BEGIN".           */
          CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
          DO WHILE(TIPO ¬= 0 & (TIPO < 51 | TIPO > 56)) ;
            /* PULA TUDO ATE ENCONTRAR "LABEL", "CONST", "VAR", */
            /* "PROC", "FUNCTION", "BEGIN" OU FIM DE ARQUIVO.   */
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          END ;
          ERRO = 14 ; /* SIMBOLOS ANTERIORES FORAM IGNORADOS */
          COLUNA = PTR - 1 ;
          CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        END ;
        END ;
 END ;
 /* NESTE PONTO TEMOS TIPO=56 ("BEGIN") OU TIPO=0 (FIM DE ARQ.) */
 IF TIPO = 56
 THEN DO ; /* PEGOU "BEGIN" */
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        DO WHILE (TIPO ¬= 67 & TIPO ¬= 0) ;
           /* ENQUANTO NAO FOR "END" NEM EOF */
           IF TIPO = 10 /* SE FOR ";" */
           THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
           ELSE DO ;
                  CALL PLAPP17 ; /* RECONHECEDOR DE COMANDOS */
                  IF TIPO ¬= 10 & TIPO ¬= 67 /* NEM ";" NEM END */
                  THEN DO ;
                         ERRO = 12 ;  /* FALTA ";". ASSUMIDO. */
                         COLUNA = PTR - 1 ;
                         CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                       END ;
                END ;
        END ;
        IF TIPO = 67
        THEN DO ; /* PEGOU "END" */
               PROXDIM = GUARDPT(1) ;
               PRXPARM = GUARDPT(2) ;
               CALL PLAPP07 ;/* DELETA SIMB. DEFINIDOS NESTE BLOCO*/
               IF BLOCO > BLOCMAX
               THEN BLOCMAX = BLOCO ;
               BLOCO = BLOCO - 1 ;
               CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
             END ;
      END ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
          ('PLAPP08 (RECONHECEDOR DE BLOCO) TERMINOU COM BLOCO = ',
          BLOCO , ' PROXINT = ' , PROXINT , ' PROXRE = ' , PROXRE ,
          ' ULTALFA = ' , ULTALFA , ' PROXDIM = ' , PROXDIM ,
          ' E PRXPARM = ' , PRXPARM)((6) (A , F(4))) ;
 END PLAPP08 ;
```

```
/* PLAPP09 - RECONHECE DECLARACAO DE CONSTANTE                    */
PLAPP09 : PROC ;
/*****************************************************************/
/* ESTA PROCEDURE RECONHECE DECLARACAO DE CONSTANTE.              */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MANIPULACAO DAS TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL 1 LISTA STATIC ,
       2 PILHA(32)        BIN FIXED(15,0) ,
       2 TOPO             BIN FIXED(15,0) ,
       2 LIMTOPO          BIN FIXED(15,0) INIT(32) ;
 DCL SINAL            BIN FIXED(15,0) STATIC ;
 DCL TIPCTE           BIN FIXED(15,0) STATIC ;
 DCL I                BIN FIXED(15,0) STATIC ;
 DCL AUX              BIN FIXED(15,0) STATIC ;
 DCL CASE(4)          AUTOMATIC LABEL INIT(CASE1,CASE2,CASE3,CASE4) ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('PLAPP09 (REC. DCL CONSTANTES) INICIOU.')(A);
 DO WHILE(TIPO ¬= 10 & TIPO ¬= 0) ;/* NAO E' ; NEM FIM DE ARQUIVO */
   TOPO = 0 ; /* ESVAZIA A PILHA */
   DO WHILE(TIPO ¬= 10 & TIPO ¬= 38 & TIPO ¬= 0) ;
     /* ENQUANTO NAO FOR ; NEM = NEM FIM DE ARQUIVO */
     CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
     IF TIPO = 1
     THEN DO ; /* PEGOU IDENTIFICADOR */
            IDNOME = IDENT ;
            CALL PLAPP06 ; /* INSERE IDENT. NA TABELA SIMBOLOS */
            IF ¬ ACHOU
            THEN DO ;
                   IF TOPO = LIMTOPO
                   THEN DO ;
                          ERRO = 39 ; /* PLAPP09 - RESTRICAO DE   */
                                      /* IMPLEMENTACAO. NUMERO    */
                                      /* MAXIMO DE IDENTIFICADO-  */
                                      /* RES NUMA LISTA DE DECLA- */
                                      /* RACAO DE CONSTANTES      */
                                      /* FOI ULTRAPASSADO.        */
                          COLUNA = PTR - 1 ;
                          CALL PLAPP01 ; /* IMPRIME MSG DE ERRO  */
                          ERRO = 41 ;  /* COMPILACAO ABORTADA      */
                          COLUNA = 1 ; /* DEVIDO A ERRO GRAVE.     */
```

```
                        CALL PLAPP01 ; /* IMPRIME MSG DE ERRO  */
                        STOP ;
                      END ;
                TOPO = TOPO + 1 ;
                PILHA(TOPO) = IND ; /* GUARDA NA PILHA A POSICAO   */
                                    /* DA CONSTANTE NA TABELA DE   */
                                    /* SIMBOLOS P/DEPOIS COMPLETAR.*/
                IDTIPO (IND) = -1 ; /* IDENT. NAO DECLARADO */
              END ;
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          END ;
      ELSE DO ;
            IF TIPO = 9 | TIPO = 10 | TIPO = 38 | TIPO = 0
            THEN DO ; /* PEGOU , OU ; OU = OU FIM DE ARQUIVO */
                  ERRO = 33 ; /* FALTA NOME DA CONSTANTE */
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                END ;
            ELSE DO ;
                  ERRO = 34 ; /* NOME DE CONSTANTE INVALIDO */
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                  CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                END ;
          END ;
      IF TIPO ¬= 9 & TIPO ¬= 38 /* NAO E , NEM = */
      THEN DO ;
            ERRO = 35 ; /* SIMBOLO INVALIDO. "," OU "=" ESPERADOS.*/
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
            DO WHILE(TIPO ¬= 9 & TIPO ¬= 10 & TIPO ¬= 38 & TIPO ¬= 0) ;
              /* PULA TUDO ATE , OU ; OU = OU FIM DE ARQUIVO */
              CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
            END ;
            ERRO = 14 ; /* SIMBOLOS ANTERIORES FORAM IGNORADOS */
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
          END ;
    END ;
    IF TIPO = 38
    THEN DO ; /* PEGOU = */
          CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          /* RECONHECE CONSTANTE */
          SINAL = 0 ;
          IF TIPO = 20 | TIPO = 21
          THEN DO ; /* PEGOU + OU - */
                SINAL = TIPO ;
                CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
              END ;
          IF TIPO < 1 | TIPO > 4 /* NAO E IDENTIFICADOR NEM CONSTANTE */
          THEN DO ;              /* INTEIRA NEM CONSTANTE REAL E NEM  */
                                 /* LITERAL. LOGO, CONSTANTE INVALIDA */
                TIPCTE = 0 ; /* CTE INDEFINIDA */
                COLUNA = PTR - 1 ;
                IF TIPO = 9 | TIPO = 10 | TIPO = 0
                THEN DO ; /* PEGOU , OU ; OU FIM DE ARQUIVO */
                      ERRO = 36 ; /* FALTA VALOR DA CONSTANTE */
                      CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                    END ;
                ELSE DO ;
```

```
                        IF SINAL = 0 /* PEGOU SINAL ANTES? */
                        THEN ERRO = 38 ; /* TIPO CTE INVALIDO EM */
                                         /* DECLARACAO DE CTE    */
                        ELSE ERRO = 37 ; /* CTE NUMERICA INVALIDA */
                                        /* EM DECLARACAO DE CTE   */
                        CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                      END ;
                END ;
           ELSE DO ;
                  GOTO CASE(TIPO) ;
CASE1 : /* PEGOU IDENTIFICADOR */
                   IDNOME = IDENT ;
                   CALL PLAPP05 ; /* PROCURA IDENTIFICADOR NA TABELA  */
                                  /* DE SIMBOLOS                      */
                   IF ¬ACHOU
                   THEN DO ;
                          /* COLOCA TIPO DE IDENTIFICADOR (CONS- */
                          /* TANTE) E TIPO DE CONSTANTE (INDEFI- */
                          /* NIDO) NA TABELA DE SIMBOLOS.        */
                          IDTIPO(IND) = 0 ;
                          TIPCTE = -1 ; /* IDENT. NAO DECLARADO */
                          ERRO = 38 ;  /* EM DECLARACAO DE CTE.  */
                          COLUNA = PTR - 1 ;
                          CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                        END ;
                   ELSE IF IDTIPO(IND) > 4 /* TIPO DA TABELA DE SIMBO-*/
                        THEN DO ;          /* LOS NAO E CONSTANTE     */
                               TIPCTE = 0 ; /* TIPO INDEFINIDO DE CTE */
                               ERRO = 46 ; /* VALOR DECLARADO PARA    */
                                           /* CONSTANTE NAO E CONSTAN-*/
                                           /* TE. CONSIDERADO INDEFI- */
                                           /* NIDO.                   */
                               COLUNA = PTR - 1 ;
                               CALL PLAPP01 ; /* IMPRIME MENSAG.ERRO */
                             END ;
                        ELSE IF SINAL ¬= 0 &
                               (IDTIPO(IND) = 4 | IDTIPO(IND)) = 3
                             THEN DO ; /* PEGOU SINAL ANTES E      */
                                       /* TIPO DA TABELA DE SIM-   */
                                       /* LOS E' DE CONSTANTE AL- */
                                       /* FANUMERICA OU BOOLEANA. */
                                  TIPCTE = 0 ; /* CONSTANTE NU- */
                                   /* MERICA INVALIDA EM DECLA- */
                                  ERRO = 37 ; /* RACAO DE CTE   */
                                  COLUNA = PTR - 1 ;
                                  CALL PLAPP01 ;
                                END ;
                             ELSE DO ; /* CTE DEFINIDA POR MEIO */
                                       /* CONSTANTE JA DEFINIDA.*/
                                  TIPCTE = IDTIPO(IND) ;
                                  IF TIPCTE = -1
                                  THEN TIPCTE = 0 ;
                                  IF SINAL = 21 & TIPCTE > 0
                                  THEN IF TIPCTE = 1
                                       THEN DO ; /* IDENT. CTE INT.*/
                                              CTEINT =
                                                - VALINT(PTCTE(IND));
                                              /* INSERE TAB.CTE INT*/
                                              CALL PLAPP13 ;
```

```
                                        END ;
                                    ELSE DO ; /* IDENT.CTE REAL */
                                           CTEREAL =
                                             -VALREAL(PTCTE(IND));
                                           /*INSERE TAB.CTE REAL*/
                                           CALL PLAPP14 ;
                                        END ;
                                ELSE DO ;
                                       POSCTE = PTCTE(IND) ;
                                       ENDCTE = ENDER(IND) ;
                                    END ;
                            END ;
                    GOTO ENDCASE ;
CASE2 : /* PEGOU CONSTANTE INTEIRA */
                    TIPCTE = 1 ; /* CONSTANTE INTEIRA OK */
                    IF SINAL = 21 /* PEGOU SINAL MENOS ANTES ? */
                    THEN CTEINT = - NINT ;
                    ELSE CTEINT = NINT ;
                    CALL PLAPP13 ; /* INSERE NA TAB. CTES INTEIRAS */
                    GOTO ENDCASE ;
CASE3 : /* PEGOU CONSTANTE REAL */
                    TIPCTE = 2 ; /* CONSTANTE REAL OK */
                    IF SINAL = 21 /* PEGOU SINAL "-" ANTES ? */
                    THEN CTEREAL = - NREAL ;
                    ELSE CTEREAL = NREAL ;
                    CALL PLAPP14 ; /* INSERE NA TAB. DE CTES REAIS */
                    GOTO ENDCASE ;
CASE4 : /* PEGOU LITERAL */
                    IF SINAL = 0 /* PEGOU SINAL ANTES ? */
                    THEN DO ; /* CONSTANTE ALFANUMERICA OK */
                           TIPCTE = 4 ;
                           CTELIT = STRING ;
                           TAMLIT = TAM ;
                           CALL PLAPP15 ; /* INSERE TAB. CTES ALFA */
                        END ;
                    ELSE DO ; /* PEGOU SINAL */
                           TIPCTE = 0 ; /* CTE NUMERICA INVALIDA  */
                           ERRO = 37 ;  /* DECLARACAO DE CTE.     */
                           COLUNA = PTR - 1 ;
                           CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                        END ;
ENDCASE :           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                END ;
            IF TIPO ¬= 9 & TIPO ¬= 10
            THEN DO ; /* NAO E , NEM ; */
                   ERRO = 28 ; /* SIMBOLO INVALIDO. ESPERADO , OU ; */
                   COLUNA = PTR - 1 ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   DO WHILE(TIPO ¬= 9 & TIPO ¬= 10 & TIPO ¬= 0) ;
                     /* PULA TUDO ATE , OU ; OU FIM DE ARQUIVO */
                     CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                   END ;
                   ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
                   COLUNA = PTR - 1 ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                END ;
        END ;
    ELSE DO ; /* PEGOU ; OU FIM DE ARQUIVO */
            ERRO = 36 ; /* FALTA VALOR DA CONSTANTE */
            COLUNA = PTR - 1 ;
```

```
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
            TIPCTE = 0 ; /* TIPO DE CONSTANTE INDEFINIDO */
          END ;
    IF TIPCTE = 0     /* CTE INDEFINIDA ? */
    THEN DO I = 1 TO TOPO ;
            /* COMPLETA POSICAO PILHA(I) DA TABELA DE SIMBOLOS */
            IDTIPO(PILHA(I)) = TIPCTE ;
          END ;
    ELSE DO I = 1 TO TOPO ;
            /* COMPLETA POSICAO PILHA(I) DA TABELA DE SIMBOLOS */
            AUX = PILHA(I) ;
            IDTIPO(AUX) = TIPCTE ;
            PTCTE(AUX) = POSCTE ;
            ENDER(AUX) = ENDCTE ;
          END ;
END ;
IF TIPO = 10 /* PEGOU ";" ? */
THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('PLAPP09 (REC. DCL CONSTANTES) RETORNOU.')(A);
END PLAPP09 ;
```

```
/* PLAPP10 - RECONHECE DECLARACAO DE VARIAVEIS.                    */
PLAPP10 : PROC ;
/*****************************************************************/
/* ESTA PROCEDURE RECONHECE DECLARACAO DE VARIAVEL                */
/*****************************************************************/
1 /* IMPRIME MENSAGEM DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* RECONHECE TIPO DE VARIAVEL */
$CONTINUE WITH PLAMP18 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABDIM E PROXDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO GERA E GERAEND */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL TOPO           BIN FIXED (15,0) STATIC ;
 DCL PILHA (16)     BIN FIXED (15,0) STATIC ;
 DCL TIPVAR         BIN FIXED (15,0) STATIC ;
 DCL PTDIM          BIN FIXED (15,0) STATIC ;
 DCL (I,N,AUX)      BIN FIXED (15,0) STATIC ;
 DCL NELEM          BIN FIXED (31,0) STATIC ;
 DCL CASE(0:4)      LABEL INIT (FIM , INT , RE , BOOL , CHAR) ;
 DCL MOD            BUILTIN ;
 /* CHAMADO REC. DE DECLARACAO DE VARIAVEIS COM TIPO = VAR */
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('PLAPP10 (REC. DCL VARIAVEL) INICIOU.')(A) ;
 CALL PLAPP04 ;  /* RECONHECEDOR LEXICO */
 IF TIPO = 10 | TIPO = 0   /* SE PEGOU ";" OU EOF  */
 THEN DO ;
       ERRO = 64 ; /* DECLARACAO DE VARIAVEIS VAZIA */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
      END ;
 ELSE DO WHILE ( TIPO ¬= 10 & TIPO ¬= 0 ) ; /* ENQUANTO NAO PEGAR */
                                            /* ";" OU EOF         */
       TOPO = 0 ;
       IF TIPO = 13   /*  SE PEGOU ":"  */
       THEN DO ;
            ERRO = 63 ; /* FALTA LISTA DE IDENTIFICADORES */
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ;
            END ;
       DO WHILE ( TIPO ¬= 10 & TIPO ¬= 13 & TIPO ¬= 0 ) ;
          /* ENQUANTO NAO PEGAR ";" , ":" OU EOF  */
        IF TIPO = 9 /* SE PEGOU "," */
        THEN DO ;
             ERRO = 52 ; /* FALTA IDENTIFICADOR DE VARIAVEL */
```

```
          COLUNA = PTR - 1 ;
          CALL PLAPP01 ;
         END ;
    ELSE IF TIPO ¬= 1  /*  NAO PEGOU IDENTIFICADOR */
         THEN DO ;
               ERRO = 53 ;   /*  NOME DE VARIAVEL INVALIDO */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ;
               CALL PLAPP04 ;
              END ;
         ELSE DO ;
               IDNOME = IDENT ;
               CALL PLAPP06 ; /* INSERE NA TABELA DE SIMBOLOS */
               IF ¬ACHOU
               THEN DO ;
                     IF TOPO = 16
                     THEN DO ;
                           ERRO = 51 ; /* LISTA COM MAIS DE 16    */
                                       /* IDENTIFICADORES. RESTRI-*/
                                       /* CAO DE IMPLEMENTACAO.   */
                           COLUNA = PTR - 1 ;
                           CALL PLAPP01 ;
                           ERRO = 41 ;         /* COMPILACAO ABOR- */
                           COLUNA = PTR - 1 ; /* TADA DEVIDO A ER-*/
                                               /* RO GRAVE.       */
                           CALL PLAPP01 ; /* IMP. MENSAGEM ERRO */
                           STOP ;
                          END ;
                     TOPO = TOPO + 1 ;
                     PILHA ( TOPO ) = IND ;
                    END ;
               CALL PLAPP04 ;  /* SCAN */
              END ;
    IF TIPO = 9   /* SE PEGOU "," */
    THEN CALL PLAPP04 ;  /* SCAN */
    ELSE IF TIPO ¬= 13 /* NAO E' ":" */
         THEN DO ;
               ERRO = 54 ; /* SIMBOLO INVALIDO.    */
                           /* "," OU ":" ESPERADO */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ;
               DO WHILE ( TIPO ¬= 13 & TIPO ¬= 10 & TIPO ¬= 0 );
                 /* ENQUANTO NAO PEGAR ":" , ";" , EOF */
                CALL PLAPP04 ;  /* SCAN */
               END ;
               ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ;
              END ;
END ;
IF TIPO ¬= 13  /*  NAO E' ":" */
THEN DO ;
      ERRO = 55 ; /* FALTA TIPO DA VARIAVEL . */
                  /* CONSIDERADO INDEFINIDO   */
      COLUNA = PTR - 1 ;
      CALL PLAPP01 ;
      TIPVAR = 40 ; /* TIPO DA VAR. INDEFINIDO */
     END ;
ELSE DO ; /* RECONHECE TIPO DA VARIAVEL */
      CALL PLAPP18(TIPVAR , PTDIM) ; /* RECONHEC.TIPO VARIAV. */
```

```
            /* PULA TIPO SE VALIDO */
            IF TIPVAR ¬= 40 & TIPVAR ¬= 50
            THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
            IF TIPO ¬= 9 & TIPO ¬= 10 /* NAO E' "," NEM ";" */
            THEN DO ;
                   ERRO = 28 ;          /* SIMBOLO INVALIDO. ESPERADO */
                   COLUNA = PTR - 1 ; /* "," OU ";"                    */
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   IF TIPO ¬= 0
                   THEN DO ;
                          DO WHILE(TIPO ¬= 10 & TIPO ¬= 0) ;
                            /* ENQUANTO NAO FOR ";" NEM EOF */
                            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                          END ;
                          ERRO = 14 ;          /* SIMBOLOS ANTERIORES */
                          COLUNA = PTR - 1 ; /* IGNORADOS           */
                          CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO*/
                        END ;
                 END ;
            ELSE IF TIPO = 9 /* PEGOU "," ? */
                 THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          END ;
       DO I = 1 TO TOPO ; /* COMPLETA POSICAO PILHA(I) DA */
                          /* TABELA DE SIMBOLOS.          */
         AUX = PILHA(I) ;
         IDTIPO (AUX) = TIPVAR ;
         IF TIPVAR > 50
         THEN DO ; /* ARRAY */
                PTCTE(AUX) = PTDIM ;
                NELEM = TABDIM(PTDIM+1) ;
              END ;
         ELSE NELEM = 1 ; /* VAR. SIMPLES */
         GOTO CASE(MOD(TIPVAR,10)) ;
INT :    ENDER(AUX) = CNTINT ;
         CNTINT = CNTINT + NELEM ;
         GOTO FIM ;
RE:      ENDER(AUX) = CNTREAL ;
         CNTREAL = CNTREAL + NELEM ;
         GOTO FIM;
BOOL :   ENDER(AUX) = CNTBOOL ;
         CNTBOOL = CNTBOOL + NELEM ;
         GOTO FIM ;
CHAR :   ENDER(AUX) = CNTCHAR ;
         CNTCHAR = CNTCHAR + NELEM ;
FIM :
       END ;
     END ;
IF TIPO = 10 /* PEGOU ";" ? */
THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('PLAPP10 (REC. DCL VARIAVEL) RETORNOU.')(A);
END PLAPP10 ;
```

```
/* PLAPP11 - RECONHECEDOR DE CABECALHO DE PROCEDURE.                */
PLAPP11 : PROC ;
/*******************************************************************/
/* ESTA PROCEDURE RECONHECE CABECALHO DE PROCEDURE.                 */
/* E' CHAMADA PELO RECONHECEDOR DE BLOCO (PLAPP08) COM TIPO = 54    */
/* ("PROC") E RETORNA COM O TIPO CORRESPONDENTE AO SIMBOLO QUE      */
/* SEGUE O ";" OU TIPO = 0 (FIM DE ARQUIVO).                        */
/*******************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* REC. SECAO PARAMETROS FORMAIS */
$CONTINUE WITH PLAMP20 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABPARM, PRXPARM E LIMPARM */
$CONTINUE WITH PLAMV09 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL RBASE                    BIN FIXED(15,0) STATIC ;
 DCL (I,J)                    BIN FIXED(15,0) STATIC ;
 DCL GUARDAPOS                BIN FIXED(15,0) STATIC ;
 DCL GUARDAPT                 BIN FIXED(15,0) STATIC ;
 DCL NPARM                    BIN FIXED(15,0) STATIC ;
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
          ('PLAPP11 (REC.DE CABECALHO DE PROCEDURE) INICIOU COM' ,
           ' TIPO= ',TIPO) (A , A , F(3)) ;
 GUARDAPOS = -1 ;
 CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
 IF TIPO ¬= 1
 THEN DO ; /* NAO E IDENTIFICADOR */
        ERRO = 73 ; /* FALTA NOME DA PROCEDURE */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
      END ;
 ELSE DO ;
        IDNOME = IDENT ; /* INSERE NOME DA PROCEDURE NA TABELA */
        CALL PLAPP06 ;   /* DE SIMBOLOS                        */
```

```
        IF ¬ACHOU /* INSERCAO OK ? */
        THEN DO ;
                IDTIPO(IND) = 10 ; /* IDTIPO = NOME DE PROCEDURE */
                IF GERA
                THEN ENDER(IND) = CINST + LC1 ;
                GUARDAPOS = IND ; /* GUARDA POSICAO DO NOME DA     */
                                  /* PROCEDURE NA TABELA DE SIM-   */
                                  /* BOLOS P/ COMPLETAR DEPOIS     */
             END ;
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
      END ;
BLOCO = BLOCO + 1 ;
IF PRXPARM + 1 > LIMPARM
THEN DO ;
        /* PLAPP11 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO  */
        /* PERMITIDO DE ELEMENTOS NA TABELA DE PARAMETROS FORMAIS*/
        /* FOI ULTRAPASSADO                                      */
        ERRO = 143 ;
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        ERRO = 41 ; /* COMPILACAO ABORTADA */
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        STOP ;
      END ;
IF GERA
THEN DO ;
        IF PROXINT + 8 > LSKINT
        THEN DO ;
                /* PLAPP11 - RESTRICAO DE IMPLEMENTACAO - NUMERO  */
                /* MAXIMO PERMITIDO DE CTES INTEIRAS NO PROGRAMA  */
                /* FOI ULTRAPASSADO.                              */
                ERRO = 144 ;
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                ERRO = 41 ; /* COMPILACAO ABORTADA */
                CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                STOP ;
             END ;
        TABPARM(PRXPARM) = PROXINT ; /* ENDERECO DE LIGACAO */
        /* GERA PROLOGO DA PROCEDURE */
        RBASE = BLOCO + 1 ; /* BASE DA PROCEDURE */
        CALL PLAPP34(COZEREG,0,1,0,1,0,1) ; /* ZERA REGS. INDEX.*/
        J = PROXINT + 1 ;
        DO I = 0 TO 7 ;
          CALL PLAPP34(COSETB,0,I+1,0,RBASE,0,I+3) ;
          CALL PLAPP34(COADI,0,I+3,0,J+I,0,I+3) ;
        END ;
        /* GERA INSTRUCAO P/SALVAR END.DE RETORNO */
        CALL PLAPP34(COSET,0,0,0,2,0,PROXINT) ;
        CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
        TEMINT = TEMINT - 1 ;
        CALL PLAPP34(COATRIB,0,1,0,0,RBASE,TEMINT) ;
        /* GERACAO DE PROLOGO DA PROCEDURE CONTINUARA' NO REC.DE */
        /* SECAO DE PARAMETROS FORMAIS.                          */
        PROXINT = PROXINT + 9 ;
      END ;
GUARDAPT = PRXPARM ; /* SALVA PONTEIRO DA TABELA DE PARAMETROS */
PRXPARM = PRXPARM + 2 ;
IF TIPO = 11 /* PEGOU "(" ? */
THEN NPARM = PLAPP20(TABPARM(GUARDAPT),0) ; /* REC.DE SECAO DE   */
```

```
                                                    /* PARAMETROS FORMAIS*/
 ELSE NPARM = 0 ; /* PROCEDURE SEM PARAMETROS */
 IF GUARDAPOS ¬= -1 /* NOME DA PROCEDURE OK ? */
 THEN DO ;
         PTCTE(GUARDAPOS) = GUARDAPT ;
         TABPARM(GUARDAPT+1) = NPARM ;
       END ;
 IF TIPO ¬= 10 & TIPO ¬= 0
 THEN DO ; /* NAO E ";" NEM FIM DE ARQUIVO */
         ERRO = 74 ; /* SIMBOLO INVALIDO. ESPERADO ";" */
         COLUNA = PTR - 1 ;
         CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
         DO WHILE(TIPO ¬= 10 & TIPO ¬= 0) ;
           /* ENQUANTO NAO FOR ";" NEM FIM DE ARQUIVO */
           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
         END ;
         ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
         COLUNA = PTR - 1 ;
         CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       END ;
 IF TIPO = 10 /* PEGOU ";" ? */
 THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
           ('PLAPP11 (RECONHECEDOR DE CABECALHO DE PROCEDURE) RET' ,
            'ORNOU')(A , A) ;
 END PLAPP11 ;
```

```
/* PLAPP12 - RECONHECEDOR DE CABECALHO DE FUNCAO.                    */
PLAPP12 : PROC ;
/*********************************************************************/
/* ESTA PROCEDURE RECONHECE CABECALHO DE FUNCAO.                     */
/* E' CHAMADA PELO RECONHECEDOR DE BLOCO (PLAPP08) COM TIPO=55       */
/* ("FUNCTION") E RETORNA COM O SIMBOLO APOS O ";", OU COM FIM       */
/* DE ARQUIVO (TIPO= 0).                                             */
/* SE O NOME DA FUNCAO QUE ESTA' SENDO ANALISADA TIVER SIDO IN-      */
/* SERIDO NA TABELA DE SIMBOLOS, SERA' INSERIDO MAIS UMA VEZ,        */
/* COMO PARAMETRO VARIAVEL SIMPLES (DEPOIS QUE A VARIAVEL            */
/* "BLOCO" TIVER SIDO INCREMENTADA), PARA PERMITIR COMANDO DE        */
/* ATRIBUICAO PARA O NOME DA FUNCAO NO CORPO DA MESMA, E IMPOS-      */
/* SIBILITAR A EXISTENCIA DE UM PARAMETRO COM O MESMO IDENTIFI-      */
/* CADOR NA SECAO DE PARAMETROS FORMAIS DESTA FUNCAO.                */
/*********************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* REC. SECAO PARAMETROS FORMAIS */
$CONTINUE WITH PLAMP20 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABPARM E PRXPARM */
$CONTINUE WITH PLAMV09 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL RBASE                    BIN FIXED(15,0) STATIC ;
 DCL (I,J)                    BIN FIXED(15,0) STATIC ;
 DCL GUARDAPOS(2)             BIN FIXED(15,0) STATIC ;
 DCL GUARDAPT                 BIN FIXED(15,0) STATIC ;
 DCL TIPFUN                   BIN FIXED(15,0) STATIC ;
 DCL NPARM                    BIN FIXED(15,0) STATIC ;
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
          ('PLAPP12 (REC.DE CABECALHO DE FUNCAO) INICIOU COM TIP' ,
           '0= ' , TIPO)(A , A , F(3)) ;
 CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
 GUARDAPOS(1) = -1 ; /* FALTA NOME DA FUNCAO OU NOME JA EXISTE */
                     /* NA TABELA DE SIMBOLOS.                */
```

```
IF TIPO ¬= 1
THEN DO ; /* NAO E' IDENTIFICADOR */
        ERRO = 75 ; /* FALTA NOME DA FUNCAO */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
      END ;
ELSE DO ;
        IDNOME = IDENT ; /* INSERE NOME DA FUNCAO NA TABELA */
        CALL PLAPP06 ;   /* SIMBOLOS.                       */
        IF ¬ACHOU /* INSERCAO OK ? */
        THEN DO ;
               /* GUARDA POSICAO EM QUE O NOME DA FUNCAO FOI IN- */
               /* SERIDO NA TABELA DE SIMBOLOS P/COMPLETAR DEPOIS*/
               GUARDAPOS(1) = IND ;
               IF GERA
               THEN ENDER(IND) = CINST + LC1 ;
             END ;
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
      END ;
BLOCO = BLOCO + 1 ;
IF PRXPARM + 1 > LIMPARM
THEN DO ;
        /* PLAPP12 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO  */
        /* PERMITIDO DE ELEMENTOS NA TABELA DE PARAMETROS FORMAIS*/
        /* FOI ULTRAPASSADO                                      */
        ERRO = 145 ;
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        ERRO = 41 ; /* COMPILACAO ABORTADA */
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        STOP ;
      END ;
IF GERA
THEN DO ;
        IF PROXINT + 8 > LSKINT
        THEN DO ;
               /* PLAPP12 - RESTRICAO DE IMPLEMENTACAO - NUMERO  */
               /* MAXIMO PERMITIDO DE CTES INTEIRAS NO PROGRAMA  */
               /* FOI ULTRAPASSADO.                              */
               ERRO = 146 ;
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               ERRO = 41 ; /* COMPILACAO ABORTADA */
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               STOP ;
             END ;
        TABPARM(PRXPARM) = PROXINT ; /* ENDERECO DE LIGACAO */
        /* GERA PROLOGO DA FUNCAO */
        RBASE = BLOCO + 1 ; /* BASE DA FUNCAO */
        CALL PLAPP34(COZEREG,0,1,0,1,0,1) ; /* ZERA REGS. INDEX.*/
        J = PROXINT + 1 ;
        DO I = 0 TO 7 ;
          CALL PLAPP34(COSETB,0,I+1,0,RBASE,0,I+3) ;
          CALL PLAPP34(COADI,0,I+3,0,J+I,0,I+3) ;
        END ;
        /* GERA INSTRUCAO P/SALVAR END.DE RETORNO */
        CALL PLAPP34(COSET,0,0,0,2,0,PROXINT) ;
        CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
        TEMINT = TEMINT - 1 ;
        CALL PLAPP34(COATRIB,0,1,0,0,RBASE,TEMINT) ;
```

```
        PROXINT = PROXINT + 9 ;
      END ;
IF GUARDAPOS(1) ¬= -1
THEN DO ;
        /* NOME DA FUNCAO FOI INSERIDO OK NA TABELA DE SIMBOLOS. */
        /* INSERE DE NOVO (BLOCO JA FOI INCREMENTADO) COMO PARA- */
        /* METRO VARIAVEL SIMPLES, P/PERMITIR COMANDO DE ATRIBUI-*/
        /* CAO P/O NOME DA FUNCAO E NAO PERMITIR PARAMETRO COM   */
        /* O MESMO IDENTIFICADOR                                 */
        CALL PLAPP06 ; /* INSERE NA TABELA DE SIMBOLOS */
        GUARDAPOS(2) = IND ;
        IF GERA
        THEN DO ;
               TEMINT = TEMINT - 1 ;
               ENDER(IND) = TEMINT ;
               /* GERA INSTRUCAO P/SALVAR END.DO TEMPORARIO QUE */
               /* RECEBERA O VALOR DA FUNCAO                    */
               CALL PLAPP34(COATRIB,0,1,0,1,RBASE,TEMINT) ;
               /* GERACAO DO PROLOGO DA FUNCAO CONTINUARA' NO    */
               /* RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS    */
             END ;
      END ;
GUARDAPT = PRXPARM ; /* SALVA PONTEIRO DA TABELA DE PARAMETROS */
PRXPARM = PRXPARM + 2 ;
IF TIPO = 11 /* PEGOU "(" ? */
THEN NPARM = PLAPP20(TABPARM(GUARDAPT),1) ; /* REC.DE SECAO DE   */
                                            /* PARAMETROS FORMAIS*/
ELSE NPARM = 0 ; /* FUNCAO SEM PARAMETROS */
IF TIPO = 13 /* PEGOU ":" ? */
THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
ELSE DO ;
        ERRO = 76 ; /* FALTA ":". FOI ASSUMIDO. */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
      END ;
IF TIPO < 76 | TIPO > 79
THEN DO ; /* NAO E INTEGER, REAL, BOOLEAN NEM CHAR */
        ERRO = 77 ; /* FALTA TIPO DA FUNCAO */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        TIPFUN = 20 ; /* FUNCAO INDEFINIDA */
      END ;
ELSE DO ;
        TIPFUN = TIPO - 55 ; /* TIPO DA FUNCAO */
                             /* 21 = INTEIRA   */
                             /* 22 = REAL      */
                             /* 23 = BOOLEAN   */
                             /* 24 = CHAR      */
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
      END ;
IF GUARDAPOS(1) ¬= -1
THEN DO ; /* NOME DA FUNCAO OK */
        PTCTE(GUARDAPOS(1)) = GUARDAPT ;
        IDTIPO(GUARDAPOS(1)) = TIPFUN ; /* TIPO DA FUNCAO */
        IDTIPO(GUARDAPOS(2)) = TIPFUN + 60 ; /* PARAM VAR SIMPLES */
        TABPARM(GUARDAPT+1) = NPARM ;
      END ;
IF TIPO ¬= 10 & TIPO ¬= 0
THEN DO ; /* NAO E ";" NEM FIM DE ARQUIVO */
        ERRO = 74 ; /* SIMBOLO INVALIDO. ESPERADO ":" */
```

```
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        DO WHILE(TIPO ¬= 10 & TIPO ¬= 0) ;
          /* ENQUANTO NAO FOR ";" NEM FIM DE ARQUIVO */
          CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        END ;
        ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
      END ;
IF TIPO = 10 /* PEGOU ";" ? */
THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
          ('PLAPP12 (RECONHECEDOR DE CABECALHO DE FUNCAO) RETORN' ,
           'OU')(A , A) ;
END PLAPP12 ;
```

```
 /* PLAPP13 - MANIPULACAO TABELAS DE CONSTANTES.                    */
 PLAPP13 : PROC ;
 /*****************************************************************/
 /* ESTA PROCEDURE MANIPULA COM AS TABELAS DE CONSTANTES.          */
 /*****************************************************************/
1 /* IMPRIME MENSAGEM DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL I              BIN FIXED(15,0) STATIC ;
 DCL Q              BIN FIXED(15,0) STATIC ;
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP13 (INSERE CTE INTEIRA) INICIOU.')( A );
 Q = 0 ;
 POSCTE = PTINT(Q) ;
 DO WHILE(POSCTE ¬= 0 & VALINT(POSCTE) < CTEINT) ;
   Q = POSCTE ;
   POSCTE = PTINT(Q) ;
 END ;
 IF POSCTE = 0 | VALINT(POSCTE) > CTEINT
 THEN IF PROXINT > LSKINT
      THEN DO ;
             ERRO = 42 ; /* PLAPP13 - RESTRICAO DE IMPLEMENTACAO.*/
                         /* NUMERO MAXIMO DE CONSTANTES INTEIRAS */
                         /* NOS BLOCOS ATIVOS FOI ULTRAPASSADO.  */
             COLUNA = PTR - 1 ;
             CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
             ERRO = 41 ; /* COMPILACAO ABORTADA DEVIDO */
                         /* A ERRO GRAVE.              */
             COLUNA = 1 ;
             CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
             STOP ;
           END ;
      ELSE DO ;
             VALINT(PROXINT) = CTEINT ;
             PTINT(PROXINT) = POSCTE ;
             PTINT(Q) = PROXINT ;
             POSCTE = PROXINT ;
             PROXINT = PROXINT + 1 ;
           END ;
 /* FUNCAO DA GERACAO DE CODIGO */
 ENDCTE = POSCTE ;
 /* FIM DA ROTINA DE GERACAO DE CODIGO */
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP13 (INSERE CTE INTEIRA) TERMINOU.')( A ) ;
 RETURN ;
1PLAPP14 : ENTRY ;
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP14 (INSERE CTE REAL) INICIOU.')( A ) ;
```

```
Q = 0 ;
POSCTE = PTREAL(Q) ;
DO WHILE(POSCTE ¬= 0 & VALREAL(POSCTE) < CTEREAL) ;
  Q = POSCTE ;
  POSCTE = PTREAL(Q) ;
END ;
IF POSCTE = 0 | VALREAL(POSCTE) > CTEREAL
THEN IF PROXRE > LSKRE
     THEN DO ;
            ERRO = 43 ; /* PLAPP14 - RESTRICAO DE IMPLEMENTACAO  */
                        /* NUMERO MAXIMO DE CONSTANTES REAIS NOS */
                        /* BLOCOS ATIVOS FOI ULTRAPASSADO.       */
            CCLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
            ERRO = 41 ; /* COMPILACAO ABORTADA DEVIDO */
                        /* A ERRO GRAVE.              */
            COLUNA = 1 ;
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
            STOP ;
          END ;
     ELSE DO ;
            VALREAL(PROXRE) = CTEREAL ;
            PTREAL(PROXRE) = POSCTE ;
            PTREAL(Q) = PROXRE ;
            POSCTE = PROXRE ;
            PROXRE = PROXRE + 1 ;
          END ;
/* FUNCAO DA GERACAO DE CODIGO */
ENDCTE = POSCTE ;
/* FIM DA ROTINA DE GERACAO DE CODIGO */
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
     ('PLAPP14 (INSERE CTE REAL) TERMINOU.')( A ) ;
RETURN ;
1PLAPP15 : ENTRY ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
     ('PLAPP15 (INSERE CTE ALFA) INICIOU.')( A ) ;
IF ULTALFA = LSKCH
THEN DO ;
       ERRO = 44 ; /* PLAPP15 - RESTRICAO DE IMPLEMENTACAO.      */
                   /* NUMERO MAXIMO DE CONSTANTES ALFANUMERICAS */
                   /* NOS BLOCOS ATIVOS FOI ULTRAPASSADO.       */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       ERRO = 41 ; /* COMPILACAO ABORTADA DEVIDO A ERRO GRAVE */
       COLUNA = 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       STOP ;
     END ;
PTALFA(ULTALFA + 1) = PTALFA(ULTALFA) + TAMALFA(ULTALFA) ;
ULTALFA = ULTALFA + 1 ;
POSCTE = ULTALFA ;
TAMALFA(ULTALFA) = TAMLIT ;
/* FUNCAO DA GERACAO DE CODIGO */
ENDCTE = PTALFA(POSCTE) ;
/* FIM DA ROTINA DE GERACAO DE CODIGO */
Q = PTALFA(ULTAIFA) - 1 ;
IF Q + TAMLIT > LSKCHT
THEN DO ;
```

```
        ERRO = 45 ; /* PLAPP15 - RESTRICAO DE IMPLEMENTACAO.      */
                    /* AREA RESERVADA PARA ARMAZENAR AS CONSTAN- */
                    /* TES ALFANUMERICAS DOS BLOCOS ATIVOS FOI   */
                    /* INSUFICIENTE.                             */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        ERRO = 41 ; /* COMPILACAO ABORTADA DEVIDO A ERRO GRAVE */
        COLUNA = 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        STOP ;
      END ;
ELSE DO I = 1 TO TAMLIT ;
        TEXTO(Q + I) = CTELIT(I) ;
      END ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
     ('PLAPP15 (INSERE CTE ALFA) TERMINOU.')( A ) ;
RETURN ;
1
/* PLAPP16 : ENTRY ;                                              */
/*   IF TRACE(1)                                                  */
/*   THEN PUT FILE (SPRINT) SKIP EDIT                             */
/*                ('PLAPP16 (DELETA CTES) INICIOU.')( A ) ;       */
/*   Q = 0 ;                                                      */
/*   DO WHILE(PTINT(Q) ¬= 0) ;                                    */
/*     IF PTINT(Q) >= PROXINT                                     */
/*     THEN PTINT(Q) = PTINT(PTINT(Q)) ;                          */
/*     ELSE Q = PTINT(Q) ;                                        */
/*   END ;                                                        */
/*   Q = 0 ;                                                      */
/*   DO WHILE(PTREAL(Q) ¬= 0) ;                                   */
/*     IF PTREAL(Q) >= PROXRE                                     */
/*     THEN PTREAL(Q) = PTREAL(PTREAL(Q)) ;                       */
/*     ELSE Q = PTREAL(Q) ;                                       */
/*   END ;                                                        */
/* IF TRACE(1)                                                    */
/* THEN PUT FILE (SPRINT) SKIP EDIT                               */
/*              ('PLAPP16 (DELETA CTES) TERMINOU.')( A ) ;        */
/* RETURN ;                                                       */
END PLAPP13 ;
```

```
/* PLAPP17 - RECONHECEDOR DE COMANDOS                              */
PLAPP17: PROC RECURSIVE ;
/*****************************************************************/
/*                                                               */
/*  ESTA PROCEDURE RECONHECE OS COMANDOS EXECUTAVEIS DO          */
/*  PROGRAMA FONTE.                                              */
/*  E' CHAMADA COM O PRIMEIRO SIMBOLO DO COMANDO LIDO E          */
/*  RETORNA COM O PRIMEIRO SIMBOLO APOS O COMANDO RECONHECIDO.   */
/*  E' CHAMADA PELO RECONHECEDOR DE BLOCO (PLAPP08), POR SI      */
/*  MESMA (PLAPP17) E PELOS RECONHECEDORES ESPECIFICOS DOS       */
/*  COMANDOS REPEAT (PLAPP21), WHILE (PLAPP22), IF (PLAPP23),    */
/*  CASE (PLAPP24) E FOR (PLAPP25).                              */
/*                                                               */
/*****************************************************************/
1 /* MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MAN. TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MAN. TABELAS DE CONSTANTE  */
$CONTINUE WITH PLAMP13 RETURN
1 /* REC. 'REPEAT' */
$CONTINUE WITH PLAMP21 RETURN
1 /* REC. 'WHILE' */
$CONTINUE WITH PLAMP22 RETURN
1 /* REC. 'IF' */
$CONTINUE WITH PLAMP23 RETURN
1 /* REC. 'CASE' */
$CONTINUE WITH PLAMP24 RETURN
1 /* REC. 'FOR' */
$CONTINUE WITH PLAMP25 RETURN
1 /* REC. 'READ' */
$CONTINUE WITH PLAMP26 RETURN
1 /* REC. 'WRITE' */
$CONTINUE WITH PLAMP27 RETURN
1 /* REC. PARAM. REAIS */
$CONTINUE WITH PLAMP28 RETURN
1 /* REC. INDICES ARRAYS */
$CONTINUE WITH PLAMP29 RETURN
1 /* REC. EXPRESSAO */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO EM 'OBJ1' */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO TABPARM */
$CONTINUE WITH PLAMV09 RETURN
```

```
1 /* MACRO TIPEXP */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL     AUX            BIN FIXED (15,0) STATIC ; /* * * * * * * * * */
 DCL     ENDDESV        BIN FIXED (15,0) STATIC ; /*                  */
 DCL     TIPOVAR        BIN FIXED (15,0) STATIC ; /*                  */
 DCL     TIPDESV        BIN FIXED (15,0) STATIC ; /*                  */
 DCL     SALVA          BIN FIXED (15,0) STATIC ; /*                  */
 DCL     MAT            BIN FIXED (15,0) STATIC ; /*  VARIAVEIS       */
 DCL     PONT           BIN FIXED (15,0) STATIC ; /*                  */
 DCL     TEXP           BIN FIXED (15,0) STATIC ; /*                  */
 DCL     BVAR           BIN FIXED(15,0) STATIC ;  /*                  */
 DCL     BINDEX         BIN FIXED(15,0) STATIC ;  /*  AUXILIARES      */
 DCL     RBASE          BIN FIXED(15,0) STATIC ;  /*                  */
 DCL     NELEM          BIN FIXED (15,0) STATIC ; /*                  */
 DCL     BNELEM         BIN FIXED(15,0) STATIC ;  /*                  */
 DCL     COP            BIN FIXED(15,0) STATIC ;  /* * * * * * * * * */
 DCL     ENDIND         BIN FIXED (15,0) ;
 DCL     TEND           BIN FIXED (15,0) ;
 DCL     ENDVAR         BIN FIXED (15,0) ;
 DCL     ENDINDEX3      BIN FIXED (15,0) ;
 DCL     MOD            BUILTIN ;
 DCL     VERDIM         ENTRY RETURNS(BIT(1)) ;
 IF TRACE (1)
 THEN PUT FILE (SPRINT) SKIP EDIT
          ('PLAPP17 (RECONHECEDOR DE COMANDOS) INICIOU COM ',
           'TIPO= ', TIPO) (A , A , F(3)) ;
 DO WHILE (TIPO = 10) ; /* ENQUANTO FOR ";" */
   CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
 END ;
 IF TIPO = 1 /* IDENTIFICADOR */
 THEN DO ;
        IDNOME = IDENT ;
        CALL PLAPP05 ; /* BUSCA NA TABELA DE SIMBOLOS */
        IF ACHOU
        THEN IF IDTIPO(IND) = 30   /* LABEL */
             THEN DO ;
                    IF PTCTE(IND) = 1 /* JA DEFINIDO ? */
                    THEN DO ;
                           ERRO = 122 ; /* DUPLICIDADE DE LABEL */
                           COLUNA = PTR - 1 ;
                           CALL PLAPP01 ; /* IMP. MSG ERRO */
                         END ;
                    ELSE DO ;
                           IF PTCTE(IND) = -1 /* JA REFERENCIADO ?*/
                           THEN VALINT(ENDER(IND)) = CINST + LC1 ;
                           PTCTE(IND) = 1 ; /* INDICA LABEL JA' */
                                            /* DEFINIDO         */
                           ENDER(IND) = CINST + LC1 ; /* END. DO */
                                                      /* LABEL   */
                         END ;
                  CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                  IF TIPO ¬= 13  /*  NAO E' ':' */
```

```
                    THEN DO ;
                          COLUNA = PTR - 1 ;
                          ERRO = 76 ;/* ESPERADO ':'. INSERIDO */
                          CALL PLAPP01 ; /* IMP. MSG DE ERRO */
                        END ;
                    ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                    IF TIPO = 1 /* SE E' IDENTIFICADOR */
                    THEN DO ;
                          IDNOME = IDENT ;
                          CALL PLAPP05 ; /* BUSCA NA TAB.DE SIMB.*/
                        END ;
                    ELSE IF TIPO = 10 | TIPO = 63 | TIPO = 67 |
                            TIPO = 70
                         THEN DO ; /*';','UNTIL','END' OU 'ELSE'*/
                                IF TRACE(1)
                                THEN PUT FILE(SPRINT) SKIP EDIT
                                       ('PLAPP17 (RECONHECEDOR ' ,
                                        'DE COMANDOS) RETORNOU ' ,
                                        'COM TIPO =' , TIPO)
                                       (A , A , A , F(3)) ;
                                RETURN ; /* COM.VAZIO APOS LABEL */
                              END ;
                  END ;
      END ;
IF TIPO = 1 /* SE E' IDENTIFICADOR */
THEN DO ;
       IF ¬ACHOU  /* NAO SE ENCONTRA NA TABELA DE SIMBOLOS */
       THEN IDTIPO (IND) = -1 ; /* TIPO INDEFINIDO */
       SALVA = IND ; /*GUARDA POSICAO DO IDENT. NA TABELA DE SIMB*/
       AUX = IDTIPO (IND) ; /* GUARDA TIPO DO IDENT. */
       IF AUX >= 40 & AUX <= 44 | /* VAR SIMPLES */
          AUX >= 50 & AUX <= 54 | /* VAR INDEXADA */
          AUX >= 80 & AUX <= 84 | /* PAR. VAR SIMPLES */
          AUX >= 90 & AUX <= 94 | /* PARAM. VAR. SIMPLES P/VALOR */
          AUX >= 100 & AUX <= 104 | /* PARAM. VAR. INDEXADA */
          AUX >= 110 & AUX <= 114 /* PAR. VAR. INDEXADA P/VALOR */
       THEN DO ;
              /* OBTEM  CODIGO DO TIPO DA VARIAVEL :  */
              /* 0 = INDEFINIDO , 1 = INTEIRO ,       */
              /* 2 = REAL , 3 = BOOLEANO , 4 = CHAR   */
              TIPOVAR = MOD (AUX,10) ;
              MAT = '0'B ; /* P/INDICAR SE VAR. SIMPLES (='0'B)*/
                           /* OU MATRICIAL. (='1'B)            */
              CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
              IF GERA
              THEN DO ;
                     IF AUX < 80 | AUX > 84 & AUX < 100 | AUX > 104
                     THEN DO ; /* NAO E' PARAMETRO POR ENDERECO */
                            /* GUARDA END.BASE DA VAR.DE DESTINO */
                            ENDVAR = ENDER(SALVA) ;
                            /* INDICA ENDERECO DIRETO */
                            ENDINDEX3 = ENDZERO ;
                            /* GUARDA BASE DA VAR.DE DESTINO */
                            BVAR = NIVEL(SALVA) + 1 ;
                            BINDEX = 0 ;
                          END ;
                     ELSE DO ;
                            ENDVAR = 0 ;
                            BVAR = 0 ;
                            ENDINDEX3 = ENDER(SALVA) ;
```

```
                    BINDEX = NIVEL(SALVA) + 1 ;
                  END ;
          END ;
    IF (AUX >= 50 & AUX <= 54 | AUX >= 100)
    THEN IF TIPO = 14 /* VAR. INDEXADA OU PAR.VAR.INDE-*/
                    /* XADA, SEGUIDA DE ABRE COLCHETES  */
          THEN DO ;
                CALL PLAPP29(SALVA,ENDIND,TEND,RBASE) ;
                                /* REC. INDICES DE ARRAY */
                IF GERA
                THEN IF TEND = 2 /* END. DIRETO? */
                     THEN ENDVAR = ENDIND ;   /* END. */
                                /* BASE + DESLOCAMENTO CTE */
                     ELSE DO ;
                            ENDINDEX3 = ENDIND ;
                            BINDEX = RBASE ;
                          END ;
                END ;
          ELSE DO ;
                 MAT = '1'B ; /* VAR.MATRICIAL */
                 PONT = PTCTE(SALVA) ;/* OBTEM PONTEIRO */
                                /* P/ DIMENSOES DO ARRAY */
                END ;
    IF TIPO ¬= 16 /* NAO E' ':=' */
    THEN DO ;
           COLUNA = PTR - 1 ;
           ERRO = 90 ; /* ESPERADO ':='. INSERIDO */
           CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
          END ;
    ELSE CALL PLAPP04 ; /* REC. LEXICO */
    CALL PLAPP30 ; /* REC. EXPRESSAO */
    IF TIPEXP >= 0
    THEN DO ;
           IF MAT & TIPEXP >= 50 /* SE VARIAVEL E */
                           /* EXPRESSAO SAO MATRICIAIS */
           THEN IF VERDIM /* DIMENSOES DA VARIAVEL ¬= */
                THEN DO ; /* DIMENSOES DA EXPRESSAO   */
                       ERRO = 119 ;
                       COLUNA = PTR - 1 ;
                       CALL PLAPP01 ; /* IMP. MSG ERRO*/
                     END ;
           /* OBTEM CODIGO DO TIPO DA EXPRESSAO : */
           /* 0 = INDEFINIDO , 1 = INTEIRO ,      */
           /* 2 = REAL , 3 = BOOLEANO , 4 = CHAR  */
           TEXP = MOD (TIPEXP,10) ;
           IF ¬MAT & TIPEXP >= 50 |
              TEXP > 0 & TEXP <= 2 & TIPOVAR > 2 |
              TIPOVAR > 0 & TEXP > 2 & TEXP¬=TIPOVAR
                /* SE VAR.SIMPLES E EXP.MATRICIAL OU  */
                /* SE EXP. NUMERICA, VAR. TEM QUE SER */
                /* NUM. (REAL OU INTEIRA). CASO CON-  */
                /* TRARIO, TEM QUE SER DO MESMO TIPO. */
           THEN DO ; /* TIPO DA EXPRESSAO NAO E'     */
                     /* COMPATIVEL COM O DA VARIAVEL */
                  ERRO = 105 ;
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ;
                END ;
           ELSE IF  TIPOVAR = 4 & TAMCHAR ¬= 1
                THEN DO ; /* DADO ALFANUMERICO COM     */
```

```
                    /* MAIS DE 1 CARACTER E' PER- */
                    /* MITIDO APENAS EM COMANDO   */
                    /* "WRITE".                   */
                  ERRO = 121 ;
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMP MSG ERRO */
                END ;
          IF GERA
          THEN DO ;
                 IF MAT
                 THEN IF AUX > 100 & AUX <= 104
                      THEN DO ;
                             TEMINT = TEMINT - 1 ;
                             CALL PLAPP34(COZEREG,0,1,
                                          0,1,0,1) ;
                             CALL PLAPP34(COADI,BINDEX,
                                          ENDINDEX3,0,
                                          TABDIM(PONT+2),
                                          BLOCO+1,TEMINT);
                             NELEM = TEMINT ;
                             BNELEM = BLOCO + 1 ;
                           END ;
                      ELSE DO ;
                             NELEM = TABDIM(PONT+2) ;
                             BNELEM = 0 ;
                           END ;
                 /* GERA INST. SET REG. 2 */
                 CALL PLAPP34(COSET,0,0,0,2,BXUOPND,
                              XUOPND) ;
                 /* GERA INST. SET REG. 3 */
                 CALL PLAPP34(COSET,0,0,0,3,BINDEX,
                              ENDINDEX3) ;
                 IF TEXP = TIPOVAR /*TIPO EXP=TIPO VAR?*/
                 THEN COP = COATRIB ; /* ATRIBUICAO */
                 ELSE COP = COCVT ; /* CONVERSAO */
                 /* OPERANDO 1 = TIPO DA ATRIBUICAO OU */
                 /* CONVERSAO                          */
                 /* OPERANDO 2 = END.DO RESULTADO DA   */
                 /* EXPRESSAO                          */
                 /* OPERANDO 3 = END.BASE DO ARRAY     */
                 /* SE VAR. MATRICIAL, GUARDA END. DA  */
                 /* INSTRUCAO 'ATRIB/CVT' P/ LOOP DE   */
                 /* ATRIBUICAO MATRICIAL               */
                 IF MAT
                 THEN ENDDESV = CINST + LC1 ;
                 CALL PLAPP34(COP,0,TIPOVAR,BUOPND,
                              EUOPND,BVAR,ENDVAR) ;
                 IF TIPEXP > 50 /* EXP. MATRICIAL ? */
                 THEN /* GERA INST.ADREG REG 2,CTE 1 */
                      CALL PLAPP34(COADREG,0,0,0,2,0,
                                   ENDUM) ;
                 IF MAT
                 THEN DO ; /* VARIAVEL MATRICIAL */
                        /* GERA INST. ADREG REG 3,CTE 1 */
                        CALL PLAPP34(COADREG,0,0,0,3,0,
                                     ENDUM) ;
                        /* GERA INST.CREG REG.3,CTE NUM.*/
                        /* ELEMENTOS DO ARRAY (OU CTE   */
                        /* ULTIMO ELEMENTO DO ARRAY SE  */
                        /* FOR PARAMETRO)               */
```

```
                                    CALL PLAPP34(COCREG,0,0,0,3,
                                                 BNELEM,NELEM) ;
                                    /* GERA INST.DESV.CONDICAO "¬=",*/
                                    /* END.DESVIO = END.INST.ATRI-  */
                                    /* BUICAO OU CONVERSAO          */
                                    CALL PLAPP34(CODESV,0,0,0,CCNEQ,
                                                 0,ENDDESV) ;
                                  END ;
                              END ;
                        END ;
                  END ;
            ELSE IF AUX = 10 /* IDENTIFICADOR DE PROCEDURE */
                  THEN DO ;
                        CALL PLAPP04 ; /* REC. LEXICO */
                        CALL PLAPP28 (SALVA,ENDIND) ; /*REC.PAR.REAIS*/
                       END ;
                  ELSE DO ;
                        COLUNA = PTR - 1 ;
                        ERRO = 94 ; /*IDENT.INVALIDO P/INICIO COMANDO*/
                        CALL PLAPP01 ;
                        DO WHILE (TIPO ¬= 0 & TIPO ¬= 10 & TIPO ¬= 67);
                          CALL PLAPP04 ; /* PROCURA ";","END" OU EOF */
                        END ;
                        COLUNA = PTR - 1 ;
                        ERRO = 14 ; /* SIMB.ANTERIORES IGNORADOS */
                        CALL PLAPP01 ;
                       END ;
      END ;
ELSE IF TIPO = 56        /* 'BEGIN' */
      THEN DO ;
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
            DO WHILE (TIPO ¬= 67 & TIPO ¬= 0) ;
               /* ENQUANTO NAO FOR 'END' NEM EOF */
               IF TIPO = 10
               THEN CALL PLAPP04 ; /* PULA ';' */
               ELSE DO ;
                      CALL PLAPP17 ; /* REC.DE COMANDOS */
                      IF TIPO ¬= 10 & TIPO ¬= 67
                      THEN DO ; /* NEM ";" NEM "END" */
                             ERRO = 12 ; /* FALTA ";". INSERIDO */
                             COLUNA = PTR - 1 ;
                             CALL PLAPP01 ;
                           END ;
                    END ;
            END ;
            IF TIPO = 67  /* 'END' */
            THEN CALL PLAPP04 ;
           END ;
ELSE IF TIPO = 66      /* 'GOTO' */
      THEN DO ;
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
            IF TIPO ¬= 1
            THEN DO ; /* NAO E' IDENTIFICADOR */
                   COLUNA = PTR - 1 ;
                   ERRO = 87 ; /* LABEL ESPERADO */
                   CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                   DO WHILE (TIPO ¬= 0 & TIPO ¬= 10 & TIPO ¬= 67);
                     /* ENQUANTO NAO FOR ";" NEM "END" NEM EOF */
                     CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                   END ;
```

```
                    COLUNA = PTR - 1 ;
                    ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
                    CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                  END ;
/* NESTE PONTO FORAM RETIRADOS BRANCOS DO INICIO DA LINHA */
ELSE DO ;
       IDNOME = IDENT ;
       CALL PLAPP05 ; /* BUSCA NA TAB. SIMBOLOS */
       IF ¬ACHOU
       THEN IDTIPO(IND) = -1 ; /* IDENTIFICADOR NAO DECLARADO */
       ELSE IF IDTIPO (IND) ¬= 30  /* NAO E' LABEL*/
            THEN DO ;
                    COLUNA = PTR - 1 ;
                    ERRO = 89 ; /* IDENT. NAO E' LABEL */
                    CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                  END ;
            ELSE IF GERA
                 THEN DO ;
                        IF PTCTE(IND) <= 0
                        THEN DO ; /*LABEL NAO DEFINIDO*/
                               IF PTCTE(IND) = 0
                               THEN DO ; /* AINDA NAO REFERENC. */
                                      PTCTE(IND) = -1 ;
                                      IF PROXINT > LSKINT
                                      THEN DO ;
                                             /* PLAPP17 - RESTRI-*/
                                             /* CAO DE IMPLEMEN- */
                                             /* TACAO - NUM.MAX. */
                                             /* CTES INTEIRAS NOS*/
                                             /* BLOCOS ATIVOS FOI*/
                                             /* ULTRAPASSADO.    */
                                             ERRO = 131 ;
                                             COLUNA = PTR - 1 ;
                                             /* IMP. MSG. ERRO */
                                             CALL PLAPP01 ;
                                             /* COMPILACAO ABORT.*/
                                             ERRO = 41 ;
                                             CALL PLAPP01 ;
                                             STOP ;
                                           END ;
                                      ENDER(IND) = PROXINT ;
                                      PROXINT = PROXINT + 1 ;
                                    END ;
                               /* CARREGA NO REG. 3 O END. DESVIO*/
                               CALL PLAPP34(COSET,0,0,0,3,0,
                                            ENDER(IND)) ;
                               TIPDESV = 1 ; /* DESVIO INDEXADO */
                               ENDDESV = 0 ;
                             END ;
                        ELSE DO ; /* LABEL JA DEFINIDO */
                               TIPDESV = 0 ; /* DESVIO DIRETO */
                               ENDDESV = ENDER(IND) ;
                             END ;
                        /* GERA INST. DESV INCONDICIONAL */
                        CALL PLAPP34(CODESV,0,TIPDESV,0,CCQQR,0,
                                     ENDDESV) ;
                      END ;
                 ELSE IF PTCTE(IND) ¬= 1
                      THEN PTCTE(IND) = -1 ;
       CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
```

```
      END ;
           END ;
/* NESTE PONTO FORAM RETIRADOS BRANCOS DO INICIO DA LINHA */
 ELSE IF TIPO = 60
      THEN CALL PLAPP21 ; /* 'REPEAT' */
      ELSE IF TIPO = 71
           THEN CALL PLAPP22 ; /* 'WHILE' */
           ELSE IF TIPO = 68
                THEN CALL PLAPP23 ; /* 'IF' */
                ELSE IF TIPO = 64
                     THEN CALL PLAPP24 ; /* 'CASE' */
                     ELSE IF TIPO = 75
                          THEN CALL PLAPP25 ; /* 'FOR' */
                          ELSE IF TIPO >= 91 & TIPO <= 94
                               THEN CALL PLAPP26 ; /* 'READ' */
                               ELSE IF TIPO >= 81 & TIPO <= 86
                                    THEN CALL PLAPP27 ;/* 'WRITE' */
ELSE DO ; /* SIMBOLO INVALIDO P/ INICIO DE COMANDO */
        COLUNA = PTR - 1 ;
        ERRO = 91 ; /* SIMBOLO INVALIDO */
        CALL PLAPP01 ;
        IF TIPO ¬= 0 /* SE NAO E' EOF */
        THEN CALL PLAPP04 ; /* PULAR O SIMBOLO INVALIDO */
      END ;
 IF TRACE (1)
 THEN PUT FILE (SPRINT) SKIP EDIT
          ('PLAPP17 (RECONHECEDOR DE COMANDOS) RETORNOU' ,
           ' COM TIPO =' , TIPO) (A , A , F(3)) ;
1VERDIM : PROC RETURNS(BIT(1)) ;
 DCL RETCODE              BIT(1) STATIC ;
 DCL (I , J , PT1 , PT2) BIN FIXED(15,0) STATIC ;
 PT1 = PONT ; /* PONT.P/ DIMENSOES DA VARIAVEL EM 'TABDIM' */
 PT2 = PTMAT ; /* PONT.P/ DIMENSOES DA EXPRESSAO EM 'TABDIM' */
 RETCODE = TABDIM(PT1)¬=TABDIM(PT2) | TABDIM(PT1+1)¬=TABDIM(PT2+1);
 J = TABDIM(PT1) ;
 PT1 = PT1 + 3 ;
 PT2 = PT2 + 3 ;
 DO I = 1 TO J WHILE(¬RETCODE) ;
   RETCODE = TABDIM(PT1+1)-TABDIM(PT1) ¬= TABDIM(PT2+1)-TABDIM(PT2);
   PT1 = PT1 + 2 ;
   PT2 = PT2 + 2 ;
 END ;
 RETURN(RETCODE) ;
 END VERDIM ;
 END PLAPP17 ;
```

```
/* PLAPP18 - RECONHECEDOR DE TIPO DE VARIAVEIS.                    */
PLAPP18 : PROC(TIPVAR , PTDIM) ;
/*****************************************************************/
/* ESTA PROCEDURE RECONHECE TIPO DE VARIAVEIS NA DECLARACAO DE   */
/* VARIAVEIS E NA DECLARACAO DE PARAMETROS FORMAIS. E' CHAMADA   */
/* PELO RECONHECEDOR DE DECLARACAO DE VARIAVEIS (PLAPP10) E      */
/* PELO RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS (PLAPP20)    */
/* COM TIPO = 13 (":") E RETORNA EM "TIPVAR" O TIPO DA VARIAVEL  */
/* (CONFORME TABELA ABAIXO) E NO CASO DE ARRAY RETORNARA' EM     */
/* "PTDIM" O INDICE DA TABELA DE DIMENSOES DE ARRAY ("TABDIM")   */
/* ONDE FORAM ARMAZENADAS AS DIMENSOES DO ARRAY. ESTE RECONHE-   */
/* CEDOR EFETUA A ANALISE ATE' O TIPO DA VARIAVEL. SE O TIPO     */
/* FOR VALIDO, ENTAO RETORNA POSICIONADA NO PROPRIO SIMBOLO      */
/* (QUE DEVERA' SER PULADO PELA PROCEDURE QUE CHAMOU). CASO      */
/* CONTRARIO, RETORNA POSICIONADA NO SIMBOLO QUE OCUPA O LUGAR   */
/* DO TIPO (SIMBOLO ESTE QUE DEVE SER ANALISADO PELA PROCEDURE   */
/* QUE CHAMOU).                                                  */
/* TIPVAR TIPO DA VARIAVEL                                       */
/*    40    VARIAVEL SIMPLES INDEFINIDA                          */
/*    41    VARIAVEL SIMPLES INTEIRA                             */
/*    42    VARIAVEL SIMPLES REAL                                */
/*    43    VARIAVEL SIMPLES BOOLEANA                            */
/*    44    VARIAVEL SIMPLES CHAR                                */
/*    50    VARIAVEL INDEXADA INDEFINIDA                         */
/*    51    VARIAVEL INDEXADA INTEIRA                            */
/*    52    VARIAVEL INDEXADA REAL                               */
/*    53    VARIAVEL INDEXADA BOOLEANA                           */
/*    54    VARIAVEL INDEXADA CHAR                               */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* RECONHECE DIMENSOES DE ARRAYS */
$CONTINUE WITH PLAMP19 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM E PROXDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL ESTADO          BIN FIXED (15,0) STATIC ;
 DCL TIPVAR          BIN FIXED (15,0) ;
 DCL PTDIM           BIN FIXED (15,0) ;
 DCL LIMSUP          BIN FIXED (15,0) STATIC ;
 DCL LIMINF          BIN FIXED (15,0) STATIC ;
 DCL VALOR           BIN FIXED (15,0) STATIC ;
 DCL MOD             BUILTIN ;
 DCL (AUX,NDIM)      BIN FIXED (31,0) STATIC ;
 DCL (X,I)           BIN FIXED (15,0) STATIC ;
 ON FIXEDOVERFLOW
 BEGIN ; /* NUMERO MAXIMO PERMITIDO DE ELEMENTOS EM ARRAY */
         /* FOI ULTRAPASSADO.                             */
   ERRO = 124 ;
```

```
   COLUNA = PTR - 1 ;
   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
   AUX = 0 ;
 END ;
 IF TRACE (1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP18 (REC.DE TIPO DE VARIAVEIS) INICIOU COM TIPO = ',
        TIPO) ( A , F(3) ) ;
 CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
 LIMINF = 1 ;
 TIPVAR = 0 ;
 IF TIPO = 57  /* ARRAY */
 THEN DO ;
       TIPVAR = 10 ;
       /* OVERFLOW EM "TABDIM" E' DETETADO ADIANTE */
       PTDIM = PROXDIM ;
       PROXDIM = PROXDIM + 3 ;
       CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
       IF TIPO ¬= 14 /* ABRE COLCHETES ? */
       THEN DO ;
             ERRO = 56 ; /* FALTA ABRE COLCHETES. INSERIDO */
             COLUNA = PTR - 1 ;
             CALL PLAPP01 ;
            END ;
       ELSE CALL PLAPP04 ;  /* RECONHECEDOR LEXICO */
       ESTADO = 0 ;    /* ESTADO INICIAL */
       DO WHILE ( ESTADO < 4 ) ; /* ENQUANTO NAO PEGAR FECHA */
                                 /* COLCHETES OU ERRO        */
        IF ESTADO < 2
        THEN DO ;
              IF TIPO=9 | TIPO=13 | TIPO=10 | TIPO=15 | TIPO=0
                   /* SE E' ",", ":", ";", FECHA COLCHETES OU EOF */
              THEN DO ;
                    ERRO = 57 ; /* FALTA DIM. ASSUMIDO INDEFINIDO*/
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ;
                    DO WHILE(TIPO¬=10 & TIPO¬=0 & TIPO¬=58 & TIPO¬=15);
                     /*  NAO E' ";", EOF, "OF" OU FECHA COLCHETE */
                     CALL PLAPP04 ;
                    END ;
                    ERRO = 14 ;
                    COLUNA = PTR - 1 ; /* SIMB. ANTERIORES IGNOR. */
                    CALL PLAPP01 ;
                    ESTADO = 4 ;
                   END ;
              ELSE DO ;
                   /* RECONHECE EXP TIPO 1 */
                    IF PLAPP19 ( VALOR)  /* SE EXP INVALIDA */
                    THEN DO ;
                          ESTADO = 4 ;
                          /* ASSUME DIM */
                          LIMSUP = 1;
                         END ;
                    ELSE DO ;
                          ESTADO = ESTADO + 2 ;
                          LIMSUP = VALOR ;
                         END ;
                   END ;
             END ;
        ELSE DO ; /* ESTADO = 2 OU ESTADO = 3 */
```

```
          IF ESTADO = 2 & TIPO = 13  /* DOIS PONTOS */
          THEN DO ;
                ESTADO = 1 ;
                LIMINF = LIMSUP ;
                CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
               END ;
          ELSE DO ;
                /* TESTAR LIMITES */
                IF LIMSUP < LIMINF
                THEN DO ;
                      ERRO = 65 ; /* LIMITE SUPERIOR < LIMITE */
                          /* INFERIOR NA DECLARACAO DO ARRAY */
                      COLUNA = PTR - 1 ;
                      CALL PLAPP01 ;
                      LIMSUP = LIMINF ;
                     END ;
                /* GUARDAR DIMENSOES  */
                IF PROXDIM + 1 > MAXDIM
                THEN DO ;
                      /* RESTRICAO DE IMPLEMENTACAO ARRAY */
                      /* DEMAIS OU DIMENSOES DEMAIS       */
                      ERRO = 66 ;
                      COLUNA = PTR - 1 ;
                      CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                      STOP ;
                     END ;
                TABDIM (PROXDIM) = LIMINF ;
                TABDIM (PROXDIM + 1 ) = LIMSUP ;
                PROXDIM = PROXDIM + 2 ;
                LIMINF = 1 ;
                IF TIPO = 9   /* VIRGULA */
                THEN DO ;
                       ESTADO = 0 ;
                       CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                     END ;
                ELSE IF TIPO = 15   /* FECHA COLCHETES */
                     THEN ESTADO = 4 ;
                     ELSE DO ;
                           IF ESTADO = 2
                           THEN ERRO = 61 ; /* "," , ":" OU */
                                    /* FECHA COLCHETES ESPERADO */
                           ELSE ERRO = 62 ; /* "," OU FECHA */
                                    /* COLCHETES ESPERADO.    */
                           COLUNA = PTR - 1 ;
                           CALL PLAPP01 ; /* IMP.MSG ERRO */
                           DO WHILE (TIPO ¬= 10 & TIPO ¬= 0 &
                                     TIPO ¬= 58 ) ;
                            /* ENQUANTO NAO PEGAR ";", EOF OU "OF" */
                             CALL PLAPP04 ;  /* SCAN */
                           END ;
                           ERRO = 14 ; /* SIMB. ANT. IGNOR. */
                           CALL PLAPP01 ;
                           ESTADO = 4 ;
                          END ;
               END ;
      END ;
END ;
IF TIPO ¬= 10 & TIPO ¬= 0  /* NEM ";" NEM EOF */
THEN DO ;
     IF TIPO = 58 | TIPO ¬= 15
```

```
          THEN DO ;
                ERRO = 58 ; /* FALTA FECHA COLCHETES. INSERIDO */
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ;
               END ;
          ELSE CALL PLAPP04 ; /* SCAN */
          IF TIPO ¬= 58   /*  NAO E' "OF" */
          THEN DO ;
                ERRO = 59 ; /* FALTA "OF" . INSERIDO */
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ;
               END ;
          ELSE CALL PLAPP04 ; /* SCAN */
         END ;
      /* COMPLETA TABDIM (NUM.DE DIMENSOES, NUM.DE ELEMENTOS, */
      /* E PONTEIRO P/AREA DE CTES QUE CONTERA' NUMERO DE ELE-*/
      /* MENTOS, NUMERO DE DIMENSOES E CONSTANTES PARA CALCU- */
      /* LO DO ENDERECO DO ELEMENTO EM FASE DE EXECUCAO) E    */
      /* COLOCA NA AREA DE CONSTANTES INTEIRAS O NUMERO DE    */
      /* ELEMENTOS, O NUMERO DE DIMENSOES E AS CTES PARA CAL- */
      /* CULO DO ENDERECO QUANDO FOR REFERENCIA A ELEMENTO DO */
      /* ARRAY.                                               */
      NDIM = (PROXDIM - PTDIM - 3) / 2 ;
      TABDIM(PTDIM) = NDIM ;
      TABDIM(PTDIM+2) = PROXINT ;
      PROXINT = PROXINT + 2 * NDIM + 2 ;
      IF PROXINT - 1 > LSKINT
      THEN DO ;
             ERRO = 42 ; /* PLAPP18 - RESTRICAO DE IMPLEMENTACAO.*/
                         /* NUMERO MAXIMO DE CONSTANTES INTEIRAS   */
                         /* NOS BLOCOS ATIVOS FOI ULTRAPASSADO.    */
             COLUNA = PTR - 1 ;
             CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
             ERRO = 41 ; /* COMPILACAO ABORTADA DEVIDO ERRO GRAVE*/
             COLUNA = 1 ;
             CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
             STOP ;
            END ;
      VALINT(PROXINT-1) = 0 ;
      X = PROXDIM - 1 ;
      AUX = 1 ;
      DO I = PROXINT-2 BY -2 TO TABDIM(PTDIM+2) + 3 ;
        AUX = AUX * (TABDIM(X)- TABDIM(X-1) + 1) ;
        VALINT(I - 1) = AUX ;
        VALINT(PROXINT-1) = VALINT(PROXINT-1) - AUX ;
        VALINT(I) = TABDIM(X-1) - 1 ;
        X = X - 2 ;
      END ;
      AUX = AUX * (TABDIM(X) - TABDIM(X-1) + 1) ;
      TABDIM(PTDIM+1) = AUX ;
      VALINT(TABDIM(PTDIM+2)) = AUX ;
      VALINT(TABDIM(PTDIM+2)+1) = TABDIM(PTDIM) ;
      VALINT(TABDIM(PTDIM+2)+2) = TABDIM(X-1) - 1 ;
     END ;  /* RECONHECEU DIMENSOES DE ARRAY */
IF TIPO < 76 | TIPO > 79  /* NAO E' TIPO DE VARIAVEL */
THEN DO ;
      ERRO = 60 ; /* FALTA TIPO DA VAR. ASSUMIDO INDEFINIDO */
      COLUNA = PTR - 1 ;
      CALL PLAPP01 ;
      TIPVAR = TIPVAR + 40 ; /* TIPO INDEFINIDO */
```

```
      END ;
ELSE TIPVAR = TIPVAR + TIPO - 35 ; /* INTEGER, REAL, BOOLEAN OU CHAR */
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP18 (REC.DE TIPO DE VARIAVEIS) RETORNOU COM TIPO = ',
        TIPO)( A , F(3) ) ;
END PLAPP18 ;
```

```
/* PLAPP19 - RECONHECEDOR DE EXPRESSAO TIPO 1        */
PLAPP19: PROC ( VALOR ) RETURNS ( BIT(1) ) ;
/*****************************************************************/
/*  ESTA FUNCAO RECONHECE EXPRESSAO UTILIZADA NAS DECLARACOES DE */
/* VARIAVEIS PARA DIMENSIONAMENTO DE ARRAYS. E' CHAMADA COM O    */
/* PRIMEIRO ELEMENTO DA EXPRESSAO LIDO (MESMO QUE INVALIDO) E RE-*/
/* TORNA EM "VALOR" O VALOR CALCULADO DA EXPRESSAO E, '0'B CASO  */
/* A EXPRESSAO ESTEJA CORRETA OU '1'B EM CASO DE ERRO.           */
/* ESTA FUNCAO RETORNA SEMPRE COM "TIPO" IGUAL A 9, 10, 13, 15   */
/* OU 0 (",", ";", ":", FECHA COLCHETE OU FIM DE ARQUIVO,        */
/* RESPECTIVAMENTE).                                             */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULADOR DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1  /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1  /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL, TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL VALOR                  BIN FIXED(15,0) ;
 DCL EXPERRO                BIT(1) STATIC ;
 DCL ESTADO                 BIN FIXED(15,0) STATIC ;
 DCL BASE                   BIN FIXED(15,0) STATIC ;
 DCL PRIDIN                 BIN FIXED(15,0) STATIC ;
 DCL TIPO                   BIN FIXED(15,0) STATIC ;
 DCL OPER(64)               BIN FIXED(15,0) STATIC ;
 DCL PRIOR(64)              BIN FIXED(15,0) STATIC ;
 DCL OPAND(64)              BIN FLOAT(21) STATIC ;
 DCL FIXED                  BUILTIN ;
 DCL MOD                    BUILTIN ;
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP19 (REC.DE EXPRESSAO TIPO 1) INICIOU ' ,
                  'COM TIPO = ' , TIPO)(A , A , F(3)) ;
 ESTADO = 0 ;
 BASE = 0 ;
 EXPERRO = '0'B ;
 TIPO = 1 ;
 PRIOR(1) = 0 ;
 OPER(1) = 26 ;
 DO WHILE(ESTADO < 4) ;
   /*ENQUANTO EXPRESSAO ESTIVER CORRETA */
   IF ESTADO ¬= 3 & TIPO = 11
   THEN DO ; /* "(" DEPOIS DE OPERADOR OU "(" */
           ESTADO = 1 ;
           BASE = BASE + 5 ;
           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        END ;
   ELSE IF ESTADO = 3 & TIPO = 12
```

```
THEN          /* ")" DEPOIS DE OPERANDO OU ")" */
     IF BASE = 0
     THEN DO ;
            ERRO = 68 ; /* SOBRA ")" OU FALTA "(" */
            ESTADO = 6 ; /* FIM DE EXP.C/ERRO PARENTIZACAO*/
          END ;
     ELSE DO ;
            BASE = BASE - 5 ;
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          END ;
ELSE IF ESTADO ¬= 3 & TIPO = 1
     THEN DO ; /* IDENT. DEPOIS DE OPERADOR OU "(" */
            IDNOME = IDENT ;
            CALL PLAPP05 ; /* BUSCA NA TABELA DE SIMBOLOS */
            IF ¬ACHOU
            THEN DO ;
                   ESTADO = 5 ; /* FIM EXPRESSAO COM ERRO */
                   IDTIPO (IND) = -1 ;
                   /* IDENT. NAO DECLARADO */
                 END ;
            IF IDTIPO(IND) > 2
            THEN DO ; /* NAO E' CTE NUMERICA */
                   ERRO = 69 ; /*IDENT.NAO NUMERICO */
                   ESTADO = 6 ;/* FIM EXP C/ ERRO */
                 END ;
            ELSE DO ;
                  IF IDTIPO(IND) = 1 /* CONST.INTEIRA ? */
                  THEN OPAND(TOPO) = VALINT(PTCTE(IND)) ;
                  ELSE IF IDTIPO(IND) = 2 /* CTE REAL ? */
                       THEN OPAND(TOPO)=VALREAL(PTCTE(IND));
                       ELSE OPAND(TOPO) = 1 ;
                  CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                  ESTADO = 3 ;
                 END ;
          END ;
     ELSE IF ESTADO ¬= 3 & TIPO > 1 & TIPO < 4
          THEN DO ; /* NUMERO DEPOIS DE OPERADOR OU "(" */
                 IF TIPO = 2 /* NUMERO INTEIRO */
                 THEN OPAND(TOPO) = NINT ;
                 ELSE OPAND(TOPO) = NREAL ;
                 CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                 ESTADO = 3 ;
               END ;
          ELSE IF ( TIPO = 20 | TIPO = 21 ) & ESTADO < 2
               THEN DO ; /* "+" OU "-" UNARIO */
                      IF TIPO = 21
                      THEN DO ; /* "-" UNARIO */
                             IF TOPO = 64
                             THEN DO ;
                                    /* PLAPP19 - RESTRICAO DE */
                                    /* IMPLEMENTACAO - NUMERO */
                                    /* MAXIMO DE OPERADORES   */
                                    /* NA EXPRESSAO (64) FOI  */
                                    /* ULTRAPASSADO.          */
                                    ERRO = 70 ;
                                    COLUNA = PTR - 1 ;
                                    CALL PLAPP01 ; /* IMPRIME */
                                                   /* MENSAGEM*/
                                                   /* DE ERRO */
                                    ERRO = 41 ; /* COMPILACAO */
```

```
                                        /* ABORTADA    */
                                  CALL PLAPP01 ; /* IMPRIME */
                                                 /* MENSAGEM*/
                                                 /* DE ERRO */
                                    STOP ;
                                  END ;
                              OPAND(TOPO) = 0 ;
                              TOPO = TOPO + 1 ;
                              OPER(TOPO) = TIPO ;
                              PRIOR(TOPO) = BASE + 1 ;
                            END ;
                        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                        ESTADO = 2 ;
                      END ;
                ELSE IF ESTADO = 3
                     THEN DO ; /* SIMBOLO ¬= ")"        */
                               /* APOS OPERANDO OU ")" */
                          IF TIPO >= 20 & TIPO <= 25 |
                             TIPO = 9 | TIPO = 13 |
                             TIPO = 15
                          THEN DO ; /* OPERADOR OU ",", */
                                    /* ":" , OU         */
                                    /* FECHA COLCHETES  */
                                IF TIPO < 20
                                THEN PRIDIN = 0 ;
                                ELSE IF TIPO < 22
                                     THEN PRIDIN=BASE + 1;
                                     ELSE PRIDIN=BASE + 2;
                                CALL REDUZ ;
                                IF TOPO ¬= 0
                                THEN DO ;
                                       IF TOPO = 64
                                       THEN DO ;
                                         /*RESTRICAO DE */
                                         /*IMPLEMENTACAO*/
                                              ERRO = 70 ;
                                              /* NUM DE */
                                              /*OPER.>64*/
                                              COLUNA=PTR-1;
                                              CALL PLAPP01 ;
                                            /*COMPILACAO*/
                                            /* ABORTADA */
                                              ERRO = 41 ;
                                              CALL PLAPP01 ;
                                              STOP ;
                                            END ;
                                       TOPO = TOPO + 1 ;
                                       OPER(TOPO) = TIPO ;
                                       PRIOR (TOPO) = PRIDIN ;
                                       ESTADO = 2 ;
                                       CALL PLAPP04 ; /*SCAN*/
                                     END ;
                                ELSE IF ESTADO ¬= 7
                                     THEN IF BASE ¬= 0
                                          THEN DO ;
                                             /*SOBRA '('*/
                                             /*  OU     */
                                             /*FALTA ')'*/
                                                ERRO = 72 ;
                                             /*FIM EXP.*/
```

```
                                                        /*C/ ERRO */
                                                          ESTADO = 6 ;
                                                        END ;
                                                    ELSE DO ;
                                                          ESTADO = 4 ;
                                                          VALOR =
                                                             OPAND(1) ;
                                                        END ;
                                            END ;
                                        ELSE DO ;
                                              ERRO = 67 ;/*EXPRES.INVALIDA*/
                                              ESTADO = 6 ;/*FIM EXP.C/ERRO*/
                                            END ;
                                    END ;
                                ELSE DO ;
                                      ERRO = 67 ; /* EXPRESSAO INVALIDA */
                                      ESTADO = 6 ;
                                    END ;
 END ;
 IF ESTADO > 4
 THEN DO ;
        IF ESTADO > 5
        THEN DO ;
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ;
             END ;
        DO WHILE(TIPO ¬= 9 & TIPO ¬= 10 & TIPO ¬= 13 & TIPO ¬= 15 &
                 TIPO ¬= 0) ;
          /* ENQUANTO NAO FOR ",",";",":",FECHA COLCHETE OU FIM ARQ */
          CALL PLAPP04 ; /* SCAN */
        END ;
        ERRO = 14 ; /* SIMBOLOS ANT. IGNORADOS */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ;
        EXPERRO = '1'B ;
        VALOR = 1 ;
      END ;
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP19 (REC.DE EXPRESSAO TIPO 1) RETORNOU ' ,
       'COM VALOR= ', VALOR , ' , EXPERRO= ' , EXPERRO,
       ' E TIPO= ' , TIPO)(A,A,F(10),A,B(1),A,F(3)) ;
 RETURN(EXPERRO) ;
REDUZ : PROC ;
   DCL OPERADOR              BIN FIXED(15,0) STATIC ;
   DO TOPO = TOPO TO 1 BY -1 WHILE(PRIOR(TOPO) >= PRIDIN) ;
     OPERADOR = OPER(TOPO) ;
     IF OPERADOR = 20  /* "+"  */
     THEN OPAND(TOPO-1) = OPAND(TOPO-1) + OPAND(TOPO);
     ELSE IF OPERADOR = 21 /* "-" */
          THEN OPAND(TOPO-1) = OPAND(TOPO-1) - OPAND(TOPO) ;
          ELSE IF OPERADOR = 23  /* "*"  */
               THEN OPAND(TOPO-1) = OPAND(TOPO-1) * OPAND(TOPO) ;
               ELSE IF OPERADOR ¬= 26
                    THEN DO ;
                           IF OPAND (TOPO) ¬= 0
                           THEN IF OPERADOR = 22 /* "/" */
                                THEN OPAND(TOPO-1) = OPAND(TOPO-1) /
                                                     OPAND(TOPO) ;
                                ELSE IF OPERADOR = 24 /* DIV */
```

```
                                THEN OPAND(TOPO-1) =
                                        FIXED(OPAND(TOPO-1) /
                                              OPAND(TOPO)) ;
                                ELSE OPAND(TOPO-1) =
                                              MOD(OPAND(TOPO-1) ,
                                              OPAND(TOPO)) ;
                        ELSE DO ;
                                ERRO = 71 ; /* DIVISAO POR 0 */
                                ESTADO = 7 ; /* FIM EXP C/ERRO */
                                TOPO = 1 ; /* P/ FORCAR FIM REDUCAO */
                              END ;
                      END ;
   END ;
END REDUZ ;
END PLAPP19 ;
```

```
/* PLAPP20 - RECONHECEDOR DE SECAO DE PARAMETROS FORMAIS          */
PLAPP20 : PROC(ENDLISTA,DESLOC) RETURNS(BIN FIXED(15,0)) ;
/*****************************************************************/
/* ESTA FUNCAO RECONHECE SECAO DE PARAMETROS FORMAIS DE PROCE-    */
/* DURES E FUNCOES. E' CHAMADA PELO RECONHECEDOR DE CABECALHO     */
/* DE PROCEDURE (PLAPP11) E PELO RECONHECEDOR DE CABECALHO DE     */
/* FUNCAO (PLAPP12), COM TIPO IGUAL A 11 ("(") E RETORNA COM      */
/* O NUMERO DE PARAMETROS RECONHECIDOS.                           */
/* RECEBE EM "ENDLISTA" O ENDERECO DA POSICAO DA AREA DE CTES     */
/* (ENDERECO DE LIGACAO) ONDE SERA' COLOCADO (EM FASE DE EXECU-   */
/* CAO) O ENDERECO ABSOLUTO DA LISTA DE PARAMETROS E "DESLOC"     */
/* INDICA QUEM CHAMOU ESTA FUNCAO (0 = RECONHECEDOR DE CABECA-    */
/* LHO DE PROCEDURE, 1 = RECONHECEDOR DE CABECALHO DE FUNCAO).    */
/* AMBOS OS PARAMETROS SAO USADOS P/GERACAO DE CODIGO.            */
/* ALEM DISSO, RETORNARA' AO SER DETETADO FIM DE ARQUIVO          */
/* (TIPO = 0) OU O SIMBOLO ")" (TIPO = 12) QUE ENCERRA A SECAO    */
/* DE PARAMETROS, E NESTE CASO A VARIAVEL TIPO CONTERA' O         */
/* VALOR CORRESPONDENTE AO PRIMEIRO SIMBOLO ENCONTRADO APOS O     */
/* ")".                                                           */
/* ESTA FUNCAO COLOCA NA TABELA DE PARAMETROS (TABPARM) AS        */
/* INFORMACOES REFERENTES AOS PARAMETROS RECONHECIDOS.            */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MANIPULACAO DAS TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECONHECE TIPO DE VARIAVEL */
$CONTINUE WITH PLAMP18 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL, TABALFA, ENDZERO, ENDUM, EFALSE, */
  /* ETRUE, ENDCTE E POSCTE                                   */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM, MAXDIM E PROXDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO TABPARM, LIMPARM E PRXPARM */
$CONTINUE WITH PLAMV09 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1 DCL (ENDLISTA,DESLOC)      BIN FIXED(15,0) ;
```

```
DCL NPARM                   BIN FIXED(15,0) STATIC ;
DCL ESTADO                  BIN FIXED(15,0) STATIC ;
DCL TOPO                    BIN FIXED(15,0) STATIC ;
DCL PILHA(16)               BIN FIXED(15,0) STATIC ;
DCL LIMTOPO                 BIN FIXED(15,0) STATIC INIT (16) ;
DCL TIPARM                  BIN FIXED(15,0) STATIC ;
DCL TIPVAR                  BIN FIXED(15,0) STATIC ;
DCL PTDIM                   BIN FIXED(15,0) STATIC ;
DCL TIPOV                   BIN FIXED(15,0) STATIC ;
DCL NELEM                   BIN FIXED(15,0) STATIC ;
DCL ENDDEST                 BIN FIXED(15,0) STATIC ;
DCL I                       BIN FIXED(15,0) STATIC ;
DCL M                       BIN FIXED(15,0) STATIC ;
DCL N                       BIN FIXED(15,0) STATIC ;
DCL K                       BIN FIXED(15,0) STATIC ;
DCL RBASE                   BIN FIXED(15,0) STATIC ;
DCL MOD                     BUILTIN ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
          ('PLAPP20 (REC.SECAO PARAMETROS FORMAIS) INICIOU COM',
           ' TIPO= ' , TIPO) (A, A, F(3)) ;
/* CHAMADA COM "(" */
/* PULA "(" */
CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
ESTADO = 0 ; /* INICIO DE SECAO DE PARAMETROS */
NPARM = 0 ;
TIPARM = 0 ;
RBASE = BLOCO + 1 ;
DO WHILE(ESTADO < 14) ;
  /* NESTE PONTO TEREMOS ESTADO 0 , 4 , 5 , 6 , 7 , 8 OU 9 */
  IF ESTADO = 0
  THEN DO ;
          TOPO = 0 ; /* ESVAZIA A PILHA */
          IF GERA
          THEN IF TIPARM > 90 & TIPARM <= 94 |
                  TIPARM > 110 & TIPARM <= 114
               THEN /* LISTA ANTERIOR ERA DE PARAM.POR VALOR. */
                    /* RESTAURA INDEXADOR 2                    */
                    CALL PLAPP34(COSET,0,0,0,2,0,ENDLISTA) ;
          IF TIPO = 54 /* "PROC" ? */
          THEN DO ;
                 TIPARM = 60 ; /* PARAMETRO PROCEDURE */
                 ESTADO = 1 ;
               END ;
          ELSE IF TIPO = 55 /* "FUNCTION" ? */
               THEN DO ;
                      TIPARM = 70 ; /* PARAMETRO FUNCAO. FALTA */
                                    /* DETERMINAR O TIPO.      */
                      ESTADO = 2 ;
                    END ;
               ELSE IF TIPO = 53 /* "VAR" ? */
                    THEN DO ;
                           TIPARM = 80 ; /* PARAMETRO VARIAVEL    */
                                         /*  POR ENDERECO. FALTA  */
                                         /* DETERMINAR O TIPO.    */
                           ESTADO = 3 ;
                         END ;
                    ELSE IF TIPO = 1 /* IDENTIFICADOR ? */
                         THEN DO ;
                                TIPARM = 90 ; /* PARAM. VARIAVEL */
```

```
                                        /* POR VALOR.FALTA */
                                        /* DETERMINAR TIPO */
                                ESTADO = 6 ;
                                IDNOME = IDENT ;
                                CALL PLAPP06 ; /* INSERE TAB.SIMB.*/
                                IF ¬ ACHOU /* INSERCAO OK ? */
                                THEN IF TOPO = LIMTOPO
                                     THEN DO ;
                                             ERRO = 78 ;/* PLAPP20 */
/* RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO DE PARAMETROS */
/* NA LISTA (16) FOI ULTRAPASSADO.                          */
                                             COLUNA = PTR - 1 ;
                                             CALL PLAPP01 ; /* IMP.*/
                                                  /* MSG ERRO */
                                             ERRO = 41 ;/* COMPILA-*/
                                        /* CAO ABORTADA DEVIDO A ERRO GRAVE */
                                             CALL PLAPP01 ;/* IMP. */
                                                  /* MSG ERRO */
                                             STOP ;
                                        END ;
                                     ELSE DO ;
                                             TOPO = TOPO + 1 ;
                                             /* GUARDA POSICAO DO  */
                                             /* PARAM.NA TAB.SIMB. */
                                             PILHA(TOPO) = IND ;
                                             TEMINT = TEMINT - 1 ;
                                             /* END.DO TEMP.C/END */
                                             /* DO PARAMETRO      */
                                             ENDER(IND) = TEMINT ;
                                             IF GERA
                                             THEN DO ;
                                                    /* SALVA END  */
                                                    /* DO PARAM.  */
                                                    DESLOC = DESLOC
                                                             + 1 ;
                                                    CALL PLAPP34(
                                                         COATRIB,0,
                                                         1,0,DESLOC,
                                                         RBASE,
                                                         TEMINT) ;
                                                  END ;
                                           END ;
                                END ;
                           ELSE DO ;
                                  IF TIPO = 12
                                  THEN DO ; /* PEGOU ")" */
                                         ERRO = 79 ; /* FALTA LISTA */
                                                     /* DE PARAM.   */
                                         ESTADO = 14 ;
                                       END ;
                                  ELSE DO ;
                                         ERRO = 80 ; /* SIMBOLO IN- */
                                         /* VALIDO. ESPERADO "PROC",*/
                                         /* "FUNCTION", "VAR" OU    */
                                         /* IDENTIFICADOR.          */
                                         ESTADO = 13 ;
                                       END ;
                                  COLUNA = PTR - 1 ;
                                  CALL PLAPP01 ; /* IMP.MSG ERRO */
                                END ;
```

```
        IF ESTADO < 7
        THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
      END ;
/* NESTE PONTO TEREMOS ESTADO 1, 2, 3, 6, 13 OU 14 */
IF ESTADO < 4
THEN IF TIPO = 1 /* IDENTIFICADOR ? */
     THEN DO ;
            IDNOME = IDENT ;
            CALL PLAPP06 ; /* INSERE NA TABELA DE SIMBOLOS */
            IF ¬ ACHOU
            THEN DO ; /* INSERCAO OK */
                   IF TIPARM = 60 /* PARAMETRO PROCEDURE ? */
                   THEN DO ;
                          IDTIPO(IND) = 60 ;
                          TEMINT = TEMINT - 1 ;
                          ENDER(IND) = TEMINT ;
                          PTCTE(IND) = 0 ;
                          IF GERA
                          THEN DO ;
                                 /* SALVA END. DO PARAMETRO */
                                 DESLOC = DESLOC + 1 ;
                                 CALL PLAPP34(COATRIB,0,1,0,
                                        DESLOC,RBASE,TEMINT) ;
                                 /* SALVA END.DE LIGACAO */
                                 DESLOC = DESLOC + 1 ;
                                 TEMINT = TEMINT - 1 ;
                                 CALL PLAPP34(COATRIB,0,1,0,
                                        DESLOC,RBASE,TEMINT) ;
                                 /* SALVA NIVEL DA PROCEDURE */
                                 DESLOC = DESLOC + 1 ;
                                 TEMINT = TEMINT - 1 ;
                                 CALL PLAPP34(COATRIB,0,1,0,
                                        DESLOC,RBASE,TEMINT) ;
                               END ;
                          /* JOGA NA LISTA DE PARAMETROS */
                          IF PRXPARM > LIMPARM
                          THEN DO ;
                                 /* PLAPP20 - RESTRICAO DE IM-*/
                                 /* PLEMENTACAO - NUMERO MAXI-*/
                                 /* MO DE ITENS PERMITIDOS NA */
                                 /* TABELA DE PARAMETROS RELA-*/
                                 /* TIVOS AOS BLOCOS ATIVOS   */
                                 /* FOI ULTRAPASSADO.         */
                                 ERRO = 81 ;
                                 COLUNA = PTR - 1 ;
                                 CALL PLAPP01 ; /* IMP.MSG ERRO*/
                                 ERRO = 41 ; /* COMP. ABORTADA */
                                 CALL PLAPP01 ; /* IMP.MSG ERRO*/
                                 STOP ;
                               END ;
                          TABPARM(PRXPARM) = TIPARM ;
                          PRXPARM = PRXPARM + 1 ;
                          NPARM = NPARM + 1 ;
                        END ;
                   ELSE DO ; /* PARAMETRO FUNCAO OU VARIAVEL */
                          /* POR ENDERECO.                  */
                          IF TOPO = LIMTOPO
                          THEN DO ;
                                 /* PLAPP20 - RESTRICAO DE IM- */
                                 /* PLEMENTACAO - NUMERO MAXI- */
```

```
                                        /* MD DE PARAMETROS NA LISTA  */
                                        /* FOI ULTRAPASSADO.          */
                                        ERRO = 78 ;
                                        COLUNA = PTR - 1 ;
                                        CALL PLAPP01 ; /* IMP.MSG ERRO*/
                                        ERRO = 41 ; /* COMP.ABORTADA  */
                                        CALL PLAPP01 ; /* IMP.MSG ERRO*/
                                        STOP ;
                                    END ;
                                TOPO = TOPO + 1 ;
                                PILHA(TOPO) = IND ; /* GUARDA NA     */
                                            /* PILHA POSICAO DO PA- */
                                            /* RAMETRO NA TABELA DE */
                                            /* SIMBOLOS.            */
                                TEMINT = TEMINT - 1 ;
                                ENDER(IND) = TEMINT ;
                                IF GERA
                                THEN DO ;
                                        /* SALVA END. DO PARAMETRO */
                                        DESLOC = DESLOC + 1 ;
                                        CALL PLAPP34(COATRIB,0,1,0,
                                                DESLOC,RBASE,TEMINT) ;
                                        IF TIPARM = 70
                                        THEN DO ; /* PARAM.FUNCAO */
                                                /* SALVA END.DE LIGACAO*/
                                                DESLOC = DESLOC + 1 ;
                                                TEMINT = TEMINT - 1 ;
                                                CALL PLAPP34(COATRIB,0,1,
                                                        0,DESLOC,
                                                        RBASE,TEMINT) ;
                                                /*SALVA NIVEL DA FUNCAO*/
                                                DESLOC = DESLOC + 1 ;
                                                TEMINT = TEMINT - 1 ;
                                                CALL PLAPP34(COATRIB,0,1,
                                                        0,DESLOC,
                                                        RBASE,TEMINT) ;
                                        END ;
                                END ;
                        END ;
                ESTADO = ESTADO + 3 ;
                CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        END ;
    ELSE IF TIPO = 9 /* PEGOU "," ? */
        THEN DO ;
                ERRO = 82 ; /* FALTA IDENTIFICADOR */
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                ESTADO = ESTADO + 6 ;
                CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
            END ;
        ELSE DO ;
                ERRO = 83 ; /* SIMB.INVALIDO. ESPERADO IDENT.*/
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                ESTADO = 13 ;
            END ;
/* NESTE PONTO TEREMOS ESTADO 4, 5, 6, 7, 8, 9, 13 OU 14 */
IF ESTADO = 4
THEN IF TIPO = 10 /* PEGOU ";" ? */
```

```
      THEN DO ;
             ESTADO = 0 ;
             CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
           END ;
      ELSE IF TIPO = 12 /* PEGOU ")" ? */
           THEN ESTADO = 14 ;
           ELSE IF TIPO = 9
                THEN DO ; /* PEGOU "," */
                       ESTADO = 7 ;
                       CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                     END ;
                ELSE DO ;
                       IF TIPO = 1
                       THEN DO ; /* PEGOU IDENTIFICADOR */
                              ERRO = 84 ; /* FALTA ",".ASSUMIDA */
                              ESTADO = 7 ;
                            END ;
                       ELSE DO ;
                              ERRO = 85 ; /* SIMBOLO INVALIDO. */
                                          /* ESPERADO  ",".    */
                              ESTADO = 13 ;
                            END ;
                       COLUNA = PTR - 1 ;
                       CALL PLAPP01 ; /* IMP. MSG DE ERRO */
                     END ;
/* NESTE PONTO TEREMOS ESTADO 0, 5, 6, 7, 8, 9, 13 OU 14 */
IF ESTADO = 5 | ESTADO = 6
THEN IF TIPO = 9
     THEN DO ; /* PEGOU "," */
            ESTADO = ESTADO + 3 ;
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          END ;
     ELSE IF TIPO = 13 /* PEGOU ":" ? */
          THEN ESTADO = ESTADO + 5 ;
          ELSE DO ;
                 IF TIPO = 1
                 THEN DO ; /* PEGOU IDENTIFICADOR */
                        ERRO = 84 ; /* FALTA ",". FOI ASSUMIDA */
                        ESTADO = ESTADO + 3 ;
                      END ;
                 ELSE DO ;
                        ERRO = 85 ; /* SIMBOLO INVALIDO */
                        ESTADO = 13 ; /* ESPERADO ",".  */
                      END ;
                 COLUNA = PTR - 1 ;
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               END ;
/* NESTE PONTO TEREMOS ESTADO 0, 7, 8, 9, 10, 11, 13 OU 14 */
IF ESTADO > 6 & ESTADO < 10
THEN IF TIPO = 1
     THEN DO ; /* PEGOU IDENTIFICADOR */
            IDNOME = IDENT ;
            CALL PLAPP06 ; /* INSERE NA TABELA DE SIMBOLOS */
            IF ¬ ACHOU
            THEN DO ; /* INSERCAO OK */
                   IF TIPARM = 60
                   THEN DO ; /* PARAMETRO PROCEDURE */
                          IDTIPO(IND) = TIPARM ;
                          TEMINT = TEMINT - 1 ;
                          ENDER(IND) = TEMINT ;
```

```
                PTCTE(IND) = 0 ;
                IF GERA
                THEN DO ;
                        /* SALVA END. DO PARAMETRO */
                        DESLOC = DESLOC + 1 ;
                        CALL PLAPP34(COATRIB,0,1,0,
                                DESLOC,RBASE,TEMINT) ;
                        /* SALVA END.DE LIGACAO */
                        DESLOC = DESLOC + 1 ;
                        TEMINT = TEMINT - 1 ;
                        CALL PLAPP34(COATRIB,0,1,0,
                                DESLOC,RBASE,TEMINT) ;
                        /* SALVA NIVEL DA PROCEDURE */
                        DESLOC = DESLOC + 1 ;
                        TEMINT = TEMINT - 1 ;
                        CALL PLAPP34(COATRIB,0,1,0,
                                DESLOC,RBASE,TEMINT) ;
                     END ;
                /* JOGA NA LISTA DE PARAMETROS */
                IF PRXPARM = LIMPARM
                THEN DO ;
                        /* PLAPP20 - RESTRICAO DE IMP-*/
                        /* PLEMENTACAO - NUMERO MAXI- */
                        /* MO DE ITENS PERMITIDOS NA  */
                        /* TABELA DE PARAMETROS RELA- */
                        /* TIVOS AOS BLOCOS ATIVOS    */
                        /* FOI ULTRAPASSADO.          */
                        ERRO = 81 ;
                        COLUNA = PTR - 1 ;
                        CALL PLAPP01 ; /*IMP.MSG.ERRO*/
                        ERRO = 41 ; /* COMPIL.ABORTADA*/
                        CALL PLAPP01 ; /*IMP.MSG.ERRO */
                        STOP ;
                     END ;
                TABPARM(PRXPARM) = TIPARM ;
                PRXPARM = PRXPARM + 1 ;
                NPARM = NPARM + 1 ;
             END ;
        ELSE DO ; /* PARAMETRO FUNCAO OU VARIAVEL, */
                  /* POR ENDERECO OU VALOR.       */
                IF TOPO = LIMTOPO
                THEN DO ;
                        /* PLAPP20 - RESTRICAO DE IM- */
                        /* PLEMENTACAO - NUMERO MAXI- */
                        /* MO DE PARAMETROS NA LISTA  */
                        /* FOI ULTRAPASSADO.          */
                        ERRO = 78 ;
                        COLUNA = PTR - 1 ;
                        CALL PLAPP01 ; /*IMP.MSG ERRO*/
                        ERRO = 41 ;/*COMPIL. ABORTADA*/
                        CALL PLAPP01 ; /*IMP.MSG ERRO */
                        STOP ;
                     END ;
                TOPO = TOPO + 1 ;
                /* GUARDA NA PILHA POSICAO DO PARA- */
                /* METRO NA TABELA DE SIMBOLOS.     */
                PILHA(TOPO) = IND ;
                TEMINT = TEMINT - 1 ;
                ENDER(IND) = TEMINT ;
                IF GERA
```

```
                        THEN DO ;
                                /* SALVA END. DO PARAMETRO */
                                DESLOC = DESLOC + 1 ;
                                CALL PLAPP34(COATRIB,0,1,0,
                                        DESLOC,RBASE,TEMINT) ;
                                IF TIPARM = 70
                                THEN DO ; /* PARAM. FUNCAO */
                                        /* SALVA END.DE LIGACAO*/
                                        DESLOC = DESLOC + 1 ;
                                        TEMINT = TEMINT - 1 ;
                                        CALL PLAPP34(COATRIB,0,1,
                                                0,DESLOC,
                                                RBASE,TEMINT) ;
                                        /*SALVA NIVEL DA FUNCAO*/
                                        DESLOC = DESLOC + 1 ;
                                        TEMINT = TEMINT - 1 ;
                                        CALL PLAPP34(COATRIB,0,1,
                                                0,DESLOC,
                                                RBASE,TEMINT) ;
                                    END ;
                            END ;
                    END ;
                END ;
            ESTADO = ESTADO - 3 ;
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        END ;
    ELSE IF TIPO = 9
        THEN DO ; /* PEGOU "," */
                ERRO = 82 ; /* FALTA IDENTIFICADOR */
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
            END ;
        ELSE IF ESTADO > 7 & TIPO = 13
                /* ESTADO 8 OU 9 E TIPO ":" */
            THEN DO ;
                    ERRO = 82 ; /* FALTA IDENTIFICADOR */
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                    ESTADO = ESTADO + 2 ;
                END ;
            ELSE DO ;
                    /* SIMB. INVALIDO. ESPERADO IDENT. */
                    ERRO = 83 ;
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMP. MSG DE ERRO */
                    ESTADO = 13 ;
                END ;
/* NESTE PONTO TEREMOS ESTADO 0, 4, 5, 6, 7, 8, 9, 10, 11, 13 OU 14*/
IF ESTADO = 10 | ESTADO = 11
THEN DO ;
        IF ESTADO = 10
        THEN DO ;
                /* PULA ":" */
                CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                IF TIPO > 75 & TIPO < 80 /* "INTEGER", "REAL",  */
                                         /* "BOOLEAN" OU "CHAR"? */
                THEN TIPARM = TIPO - 5 ; /* PARAM.FUNCAO INTEIRA,*/
                                         /* REAL, BOOLEANA OU CARACTER */
                ELSE DO ;
                        ERRO = 77 ; /* FALTA TIPO DA FUNCAO.    */
```

```
                                    /* CONSIDERADO INDEFINIDO.  */
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMP. MSG DE ERRO */
                END ;
        END ;
ELSE DO ;
        CALL PLAPP18(TIPVAR , PTDIM) ; /* REC. DE TIPO */
                                       /* DE VARIAVEIS */
        IF TIPVAR < 50 /* VARIAVEL SIMPLES ? */
        THEN TIPVAR = TIPVAR - 40 ;
        ELSE TIPVAR = TIPVAR - 30 ;
        TIPARM = TIPARM + TIPVAR ;
      END ;
/* ESTADO 12 */
IF GERA
THEN IF TIPARM > 90 & TIPARM <= 94 |
        TIPARM > 110 & TIPARM <= 114
     THEN DO ; /* PARAMETRO POR VALOR */
             /* SALVA VALORES DOS PARAMETROS */
             TIPOV = MOD(TIPARM,10) ;
             IF TIPOV = 1
             THEN ENDDEST = CNTINT ;
             ELSE IF TIPOV = 2
                  THEN ENDDEST = CNTREAL ;
                  ELSE IF TIPOV = 3
                       THEN ENDDEST = CNTBOOL ;
                       ELSE ENDDEST = CNTCHAR ;
             IF TIPARM <= 94
             THEN DO ; /* PARAM.VAR.SIMPLES POR VALOR */
                    CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO);
                    DO I = 1 TO TOPO ;
                    N = PILHA(I) ;
                    CALL PLAPP34(COSET,0,0,0,2,RBASE,
                                 ENDER(N)) ;
                    CALL PLAPP34(COATRIB,0,TIPOV,0,0,
                                 RBASE,ENDDEST) ;
                    ENDER(N) = ENDDEST ;
                    ENDDEST = ENDDEST + 1 ;
                  END ;
                END ;
             ELSE DO ; /* PARAM.VAR.INDEXADA POR VALOR */
                    /* CRIA CTE C/NUM.DE ELEM.DE CADA ARRAY*/
                    CTEINT = TABDIM(PTDIM+1) ;
                    CALL PLAPP13 ; /*INSERE NA TAB.CTES INT*/
                    NELEM = ENDCTE ;
                    DO I = 1 TO TOPO ;
                      N = PILHA(I) ;
                      CALL PLAPP34(COSET,0,0,0,2,RBASE,
                                   ENDER(N)) ;
                      CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO);
                      CALL PLAPP34(COATRIB,0,TIPOV,0,0,
                                   RBASE,ENDDEST) ;
                      CALL PLAPP34(COADREG,0,0,0,2,0,ENDUM);
                      CALL PLAPP34(COADREG,0,0,0,3,0,ENDUM);
                      CALL PLAPP34(COCREG,0,0,0,3,0,NELEM) ;
                      CALL PLAPP34(CODESV,0,0,0,CCLT,0,
                                   CINST+LC1-4) ;
                      ENDER(N) = ENDDEST ;
                      ENDDEST = ENDDEST + CTEINT ;
                    END ;
```

```
                END ;
            IF TIPOV = 1
            THEN CNTINT = ENDDEST ;
            ELSE IF TIPOV = 2
                 THEN CNTREAL = ENDDEST ;
                 ELSE IF TIPOV = 3
                      THEN CNTBOOL = ENDDEST ;
                      ELSE CNTCHAR = ENDDEST ;
            END ;
/* COMPLETA TABELA DE SIMBOLOS */
DO I = 1 TO TOPO ;
  N = PILHA(I) ;
  IDTIPO(N) = TIPARM ; /* COLOCA TIPO NA TAB.DE SIMB. */
  IF TIPARM >= 100 /* PARAMETRO VARIAVEL INDEXADA ? */
  THEN PTCTE(N) = PTDIM ; /* COLOCA PONTEIRO P/TABELA */
                          /* DE DIMENSOES DE ARRAY NA */
                          /* TABELA DE SIMBOLOS.      */
  ELSE PTCTE(N) = 0 ; /* P/INDICAR AUSENCIA DE PARAME-*/
                      /* TROS FORMAIS SE FUNCAO       */
END ;
NPARM = NPARM + TOPO ;
IF TIPARM < 100 /* SE NAO FOR PARAM.VARIAVEL INDEXADA */
THEN DO ;
       IF PRXPARM + TOPO - 1 > LIMPARM
       THEN DO ;
              /* PLAPP20 - RESTRICAO DE IMPLEMENTACAO  */
              /* NUMERO MAXIMO DE ITENS PERMITIDOS NA  */
              /* TABELA DE PARAMETROS RELATIVOS AOS    */
              /* BLOCOS ATIVOS FOI ULTRAPASSADO.       */
              ERRO = 81 ;
              COLUNA = PTR - 1 ;
              CALL PLAPP01 ; /* IMP. MSG DE ERRO */
              ERRO = 41 ; /* COMPILACAO ABORTADA   */
                          /* DEVIDO A ERRO GRAVE.  */
              CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
              STOP ;
            END ;
       DO I = 1 TO TOPO ;
         TABPARM(PRXPARM) = TIPARM ; /* COLOCA TIPO DO */
                          /* PARAMETRO NA TABELA DE PARAMETROS */
         PRXPARM = PRXPARM + 1 ;
       END ;
     END ;
ELSE IF PRXPARM + TOPO * (TABDIM(PTDIM)*2 + 3)-1 > LIMPARM
     /* TABDIM(PTDIM) NOS DA' O NUMERO DE DIMENSOES DO  */
     /* ARRAY E TABDIM(PTDIM+1) NOS DA' O NUMERO DE     */
     /* ELEMENTOS. PARA CADA DIMENSAO TEMOS LIMITE IN-  */
     /* FERIOR E SUPERIOR. PARA CADA ARRAY TEREMOS NA   */
     /* TABELA DE PARAMETROS, O TIPO DO ARRAY, O NUMERO */
     /* DE DIMENSOES, O NUMERO DE ELEMENTOS DO ARRAY E  */
     /* OS LIMITES DE CADA DIMENSAO, NESTA ORDEM.       */
     /* FINALMENTE, TOPO REPRESENTA O NUMERO DE ARRAYS. */
     THEN DO ;
            /* PLAPP20 - RESTRICAO DE IMPLEMENTACAO - */
            /* NUMERO MAXIMO DE ITENS PERMITIDOS NA   */
            /* TABELA DE PARAMETROS RELATIVOS AOS     */
            /* BLOCOS ATIVOS FOI ULTRAPASSADO.        */
            ERRO = 81 ;
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
```

```
                    /* COMPILACAO ABORTADA DEVIDO A ERRO GRAVE */
                    ERRO = 41 ;
                    CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                    STOP ;
                  END ;
            ELSE DO ;
                    N = TABDIM(PTDIM) ; /* NUM.DIM. DO ARRAY */
                    M = TABDIM(PTDIM+1) ; /* NUM.ELEM.DO ARRAY */
                    DO I = 1 TO TOPO ;
                      /* COLOCA TIPO DO ARRAY, NUM. DE DIMEN-  */
                      /* SOES E NUMERO DE ELEMENTOS DO ARRAY   */
                      /* NA TABELA DE PARAMETROS FORMAIS.      */
                      TABPARM(PRXPARM) = TIPARM ;
                      PRXPARM = PRXPARM + 1 ;
                      TABPARM(PRXPARM) = N ;
                      PRXPARM = PRXPARM + 1 ;
                      TABPARM(PRXPARM) = M ;
                      PRXPARM = PRXPARM + 1 ;
                      /* COLOCA DIMENSOES NA TABELA DE PARAM. */
                      DO K = PTDIM + 3 TO PTDIM + 2 + N * 2 ;
                        TABPARM(PRXPARM) = TABDIM(K) ;
                        PRXPARM = PRXPARM + 1 ;
                      END ;
                    END ;
                  END ;
        IF TIPARM ¬= 70 & TIPARM ¬= 80 & TIPARM ¬= 90 &
             TIPARM ¬= 100 & TIPARM ¬= 110 /* SE TIPO VALIDO */
        THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        IF TIPO = 10
        THEN DO ; /* PEGOU ";" */
               ESTADO = 0 ;
               CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
             END ;
        ELSE IF TIPO = 12 /* PEGOU ")" ? */
             THEN ESTADO = 14 ;
             ELSE DO ;
                    ERRO = 86 ; /* SIMB. INVALIDO. ESPERADO ";"*/
                                /* OU ")".                     */
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                    ESTADO = 13 ;
                  END ;
      END ;
/* NESTE PONTO TEREMOS ESTADO 0, 4, 5, 6, 7, 8, 9, 13 OU 14 */
IF ESTADO = 13
THEN DO ;
        DO WHILE(TIPO ¬= 10 & TIPO ¬= 12 & TIPO ¬= 0) ;
          /* ENQUANTO NAO FOR ";" NEM ")" NEM FIM DE ARQUIVO */
          CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        END ;
        ERRO = 14 ; /* SIMBOLOS ANTERIORES FORAM IGNORADOS */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        IF TIPO = 10
        THEN DO ; /* PEGOU ";" */
               ESTADO = 0 ;
               CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
             END ;
        ELSE ESTADO = 14 ;
      END ;
```

```
END ;
IF TIPO = 12 /* PEGOU ")" ? */
THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
          ('PLAPP20 RETORNOU COM ' , NPARM)(A, F(3)) ;
RETURN(NPARM) ;
END PLAPP20 ;
```

```
/*  PLAPP21 - RECONHECEDOR DE COMANDO "REPEAT"                    */
PLAPP21 : PROC RECURSIVE ;
/*****************************************************************/
/*                                                               */
/*  ESTA PROCEDURE RECONHECE COMANDO "REPEAT".                   */
/*  E' CHAMADA PELO RECONHECEDOR DE COMANDOS (PLAPP17), COM A    */
/*  PALAVRA RESERVADA "REPEAT" LIDA, E RETORNA NO SIMBOLO        */
/*  APOS O COMANDO.                                              */
/*                                                               */
/*****************************************************************/
1 /* MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* RECONHECEDOR DE COMANDOS */
$CONTINUE WITH PLAMP17 RETURN
1 /* RECONHECEDOR DE EXPRESSAO */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TIPEXP, PTMAT E TAMCHAR */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL ENDTEST        BIN FIXED(15,0) ;
 IF TRACE (1)
 THEN PUT FILE(SPRINT) SKIP EDIT
          ('PLAPP21(RECONHECEDOR DE COMANDO "REPEAT") INICIOU')(A);
 CALL PLAPP04 ; /* RECONHECEDOR LEXICO P/ PULAR O "REPEAT" */
 IF GERA
 THEN ENDTEST = CINST + LC1 ;
 DO WHILE (TIPO ¬= 63 & TIPO ¬= 0) ;
    /* ENQUANTO NAO FOR "UNTIL" NEM EOF */
    IF TIPO = 10  /* SE FOR ";" */
    THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
    ELSE DO ;
           CALL PLAPP17 ; /* RECONHECEDOR DE COMANDOS */
           IF TIPO ¬= 10 & TIPO ¬= 63 /* NEM ";" NEM "UNTIL" */
           THEN DO ;
                  ERRO = 12 ; /* FALTA ';'. INSERIDO */
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                END ;
         END ;
 END ;
 IF TIPO = 63 /* "UNTIL" */
```

```
THEN DO ;
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        CALL PLAPP30 ; /* RECONHECEDOR DE EXPRESSAO */
        IF TIPEXP > 0 & TIPEXP ¬= 3
        THEN DO ;
               ERRO = 106 ; /* EXPRESSAO NAO E' BOOLEANA */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ;
             END ;
        IF GERA
        THEN DO ;
               IF TUOPND ¬= BCT
               THEN DO ; /* EXPRESSAO BOOLEANA VARIAVEL */
                      /* GERA TESTE */
                      /* GERA INST. SET REG. 2 */
                      CALL PLAPP34(COSET,0,0,0,2,BXUOPND,XUOPND) ;
                      /* GERA INST. SET REG. 3 */
                      CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
                      /* GERA INST. COMP. BOOL. */
                      CALL PLAPP34(COCOMP,0,3,BUOPND,EUOPND,0,
                                   EFALSE) ;
                      /* GERA INST. DESV SE IGUAL */
                      CALL PLAPP34(CODESV,0,0,0,CCEQ,0,ENDTEST) ;
                    END ;
               ELSE IF EUOPND = EFALSE
                    THEN /* GERA INST. DESV INCOND. P/INICIO */
                         /* DO REPEAT                        */
                         CALL PLAPP34(CODESV,0,0,0,CCQQR,0,
                                      ENDTEST) ;
             END ;
     END ;
IF TRACE (1)
THEN PUT FILE(SPRINT) SKIP EDIT
          ('PLAPP21 (RECONHECEDOR DE "REPEAT") RETORNOU COM TIPO =',
              TIPO) (A , F(3)) ;
END PLAPP21 ;
```

```
/*  PLAPP22 - RECONHECEDOR DE COMANDO "WHILE"                    */
PLAPP22 : PROC RECURSIVE ;
/*****************************************************************/
/*                                                               */
/*  ESTA PROCEDURE RECONHECE COMANDO "WHILE".                    */
/*  E' CHAMADA PELO RECONHECEDOR DE COMANDOS (PLAPP17), COM A     */
/*  PALAVRA RESERVADA "WHILE" LIDA, E RETORNA NO SIMBOLO APOS O  */
/* COMANDO.                                                      */
/*                                                               */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* RECONHECEDOR DE COMANDOS */
$CONTINUE WITH PLAMP17 RETURN
1 /* RECONHECEDOR DE EXPRESSAO */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TIPEXP */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1 DCL ENDTEST          BIN FIXED(15,0) ;
  DCL ENDDESV          BIN FIXED(15,0) ;
  DCL CCOMP            BIN FIXED(15,0) STATIC ;
  IF TRACE (1)
  THEN PUT FILE(SPRINT) SKIP EDIT
          ('PLAPP22(RECONHECEDOR DE COMANDO "WHILE") INICIOU')(A) ;
  CALL PLAPP04 ; /* RECONHECEDOR LEXICO P/ PULAR O "WHILE" */
  IF GERA
  THEN ENDTEST = CINST + LC1 ;
  CALL PLAPP30 ; /* RECONHECEDOR DE EXPRESSAO */
  IF TIPEXP > 0 & TIPEXP ¬= 3
  THEN DO ;
         ERRO = 106 ; /* EXPRESSAO NAO E' BOOLEANA */
         COLUNA = PTR - 1 ;
         CALL PLAPP01 ;
       END ;
  IF TIPO ¬= 72 /* NAO E' "DO" */
  THEN DO ;
         ERRO = 95 ; /* ESPERADO "DO". INSERIDO. */
         COLUNA = PTR - 1 ;
         CALL PLAPP01 ;
       END ;
```

```
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO P/ PULAR 'DO' */
IF TIPO ¬= 10 & TIPO ¬= 67 & TIPO ¬= 63 & TIPO ¬= 70 & TIPO ¬= 0
     /* SE NAO E' ';' NEM 'END' NEM 'UNTIL' NEM 'ELSE' NEM EOF */
THEN DO ;
       IF GERA
       THEN DO ; /* GERA TESTE DA EXPRESSAO */
              ENDDESV = 0 ;
              IF TUOPND ¬= BCT | EUOPND = EFALSE
              THEN DO ; /* EXPRESSAO BOOLEANA VARIAVEL */
                     /* GERA TESTE */
                     IF TUOPND ¬= BCT
                     THEN DO ;
                           /* GERA INST. SET REG. 2 */
                           CALL PLAPP34(COSET,0,0,0,2,BXUOPND,
                                        XUOPND) ;
                           /* GERA INST. SET REG. 3 */
                           CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
                           /* GERA INST. COMP BOOLEANA */
                           CALL PLAPP34(COCOMP,0,3,BUOPND,EUOPND,
                                        0,ETRUE) ;
                           CCOMP = CCNEQ ;
                          END ;
                     ELSE CCOMP = CCQQR ;
                     IF PROXINT > LSKINT
                     THEN DO ;
                           /* PLAPP22 - RESTRICAO DE IMPLEMEN- */
                           /* TACAO - NUM. MAX. CTES INTS. NOS */
                           /* BLOCOS ATIVOS FOI ULTRAPASSADO.  */
                           ERRO = 132 ;
                           COLUNA = PTR - 1 ;
                           CALL PLAPP01 ; /* IMP. MSG ERRO */
                           ERRO = 41 ; /* COMP. ABORTADA */
                           CALL PLAPP01 ; /* IMP. MSG ERRO */
                           STOP ;
                          END ;
                     ENDDESV = PROXINT ;
                     PROXINT = PROXINT + 1 ;
                     /* GERA INSTRUCAO DESVIO INDEXADO */
                     CALL PLAPP34(COSET,0,0,0,3,0,ENDDESV) ;
                     CALL PLAPP34(CODESV,0,1,0,CCOMP,0,0) ;
                    END ;
              END ;
       CALL PLAPP17 ; /* RECONHECEDOR DE COMANDOS */
       IF GERA
       THEN DO ;
              /* GERA INST. DESV INCONDICIONAL PARA O TESTE */
              CALL PLAPP34(CODESV,0,0,0,CCQQR,0,ENDTEST) ;
              IF ENDDESV ¬= 0
              THEN /* FOI GERADO DESVIO A FRENTE  */
                   /* RESOLVE REFERENCIA ANTERIOR */
                   VALINT(ENDDESV) = CINST + LC1 ;
              END ;
     END ;
IF TRACE (1)
THEN PUT FILE(SPRINT) SKIP EDIT
        ('PLAPP22(RECONHECEDOR DE COMANDO "WHILE") RETORNOU COM',
         ' TIPO =' , TIPO) (A , A , F(3)) ;
END PLAPP22 ;
```

```
/*  PLAPP23 - RECONHECEDOR DE COMANDO "IF"                          */
PLAPP23 : PROC RECURSIVE ;
/*****************************************************************/
/*                                                               */
/*  ESTA PROCEDURE RECONHECE COMANDO "IF".                       */
/*  E' CHAMADA PELO RECONHECEDOR DE COMANDOS (PLAPP17), COM A    */
/*  PALAVRA RESERVADA "IF" LIDA, E RETORNA NO SIMBOLO APOS O     */
/* COMANDO.                                                      */
/*                                                               */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* RECONHECEDOR DE COMANDOS */
$CONTINUE WITH PLAMP17 RETURN
1 /* RECONHECEDOR DE EXPRESSAO */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO EM OBJ1 */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TIPEXP, PTMAT E TAMCHAR */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL REF            BIN FIXED(15,0) ;
 DCL 1 ESTRUT       STATIC ,
       2 LISTA(5),
         3 C            BIN FIXED(31,0) ,
         3 OP1          BIN FIXED(31,0) ,
         3 OP2          BIN FIXED(31,0) ,
         3 OP3          BIN FIXED(31,0) ,
         3 RB2          BIN FIXED(15,0) ,
         3 RB3          BIN FIXED(15,0) ,
       2 TOPO         BIN FIXED(15,0) ;
 DCL CBOOL         BIN FIXED(31,0) STATIC ;
 DCL I             BIN FIXED(15,0) STATIC ;
 IF TRACE (1)
 THEN PUT FILE (SPRINT) SKIP
      EDIT ('PLAPP23(RECONHECEDOR DE COMANDO "IF") INICIOU') (A) ;
 CALL PLAPP04 ; /* RECONHECEDOR LEXICO P/ PULAR O "IF" */
 CALL PLAPP30 ; /* RECONHECEDOR DE EXPRESSAO */
 IF TIPEXP > 0 & TIPEXP ¬= 3
 THEN DO ;
        ERRO = 106 ; /* EXPRESSAO NAO E' BOOLEANA */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMP MSG ERRO */
```

```
      END ;
IF TIPO ¬= 69 /* NAO E' "THEN" */
THEN DO ;
        ERRO = 88 ; /* ESPERADO "THEN". */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMP MSG ERRO */
        DO WHILE (TIPO ¬= 69 & TIPO ¬= 10 & TIPO ¬= 0) ;
           /* ENQUANTO NAO FOR "THEN" NEM ";" NEM EOF */
           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        END ;
        ERRO = 14 ; /* SIMBOLOS ANTERIORES IGNORADOS */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMP MSG ERRO */
      END ;
IF TIPO = 69 /* "THEN" ? */
THEN DO ;
         CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
         IF TIPO ¬= 10 & TIPO ¬= 67 & TIPO ¬= 63 & TIPO ¬= 0
         /* SE NAO FOR ";" NEM "END" NEM "UNTIL" NEM EOF */
         THEN DO ;
                REF = 0 ;
                TOPO = 0 ;
                IF GERA
                THEN DO ;
                       IF TIPO = 70 /* "ELSE"? */
                       THEN CBOOL = ETRUE ;
                       ELSE CBOOL = EFALSE ;
                       IF TUOPND ¬= BCT
                       THEN DO ;
                              /* GERA E GUARDA INST COMP BOOL C/CBOOL*/
                              C(1) = COSET ;
                              OP1(1) = 0 ;
                              RB2(1) = 0 ;
                              OP2(1) = 2 ;
                              RB3(1) = BXUOPND ;
                              OP3(1) = XUOPND ;
                              C(2) = COSET ;
                              OP1(2) = 0 ;
                              RB2(2) = 0 ;
                              OP2(2) = 3 ;
                              RB3(2) = 0 ;
                              OP3(2) = ENDZERO ;
                              C(3) = COCOMP ;
                              OP1(3) = 3 ;
                              RB2(3) = BUOPND ;
                              OP2(3) = EUOPND ;
                              RB3(3) = 0 ;
                              OP3(3) = CBOOL ;
                              /* GUARDA INST DESVIO SE IGUAL */
                              C(4) = COSET ;
                              OP1(4) = 0 ;
                              RB2(4) = 0 ;
                              OP2(4) = 3 ;
                              RB3(4) = 0 ;
                              OP3(4) = 0 ;
                              C(5) = CODESV ;
                              OP1(5) = 1 ;
                              RB2(5) = 0 ;
                              OP2(5) = CCEQ ;
                              RB3(5) = 0 ;
```

```
                    OP3(5) = 0 ;
                    TOPO = 5 ;
                   END ;
              ELSE IF EUOPND = CBOOL
                   THEN DO ;
                         /* GUARDA INST DESV INCOND INDEX */
                         C(1) = COSET ;
                         OP1(1) = 0 ;
                         RB2(1) = 0 ;
                         OP2(1) = 3 ;
                         RB3(1) = 0 ;
                         OP3(1) = 0 ;
                         C(2) = CODESV ;
                         OP1(2) = 1 ;
                         RB2(2) = 0 ;
                         OP2(2) = CCQQR ;
                         RB3(2) = 0 ;
                         OP3(2) = 0 ;
                         TOPO = 2 ;
                        END ;
         END ;
   IF TIPO ¬= 70
   THEN DO ; /* NAO E' "ELSE" */
          IF TOPO ¬= 0
          THEN DO ;
                /* GRAVA INST. GUARDADAS */
                IF PROXINT > LSKINT
                THEN DO ;
                      /* PLAPP23 - RESTRICAO DE IMPLE- */
                      /* MENTACAO - NUM.MAX. CTES INT. */
                      /* NOS BLOCOS ATIVOS FOI ULTRA-  */
                      /* PASSADO.                      */
                      ERRO = 135 ;
                      COLUNA = PTR - 1 ;
                      CALL PLAPP01 ; /* IMP MSG ERRO */
                      ERRO = 41 ; /* COMP ABORTADA */
                      CALL PLAPP01 ; /* IMP MSG ERRO */
                      STOP ;
                     END ;
                OP3(TOPO-1) = PROXINT ;
                REF = PROXINT ;
                PROXINT = PROXINT + 1 ;
                DO I = 1 TO TOPO ;
                  CALL PLAPP34(C(I),0,OP1(I),RB2(I),
                               OP2(I),RB3(I),OP3(I)) ;
                END ;
               END ;
          CALL PLAPP17 ; /* RECONHECEDOR DE COMANDO */
          /* GUARDA INSTRUCAO DESV INCONDICIONAL INDEX.*/
          C(1) = COSET ;
          OP1(1) = 0 ;
          RB2(1) = 0 ;
          OP2(1) = 3 ;
          RB3(1) = 0 ;
          OP3(1) = 0 ;
          C(2) = CODESV ;
          OP1(2) = 1 ;
          RB2(2) = 0 ;
          OP2(2) = CCQQR ;
          RB3(2) = 0 ;
```

```
                    OP3(2) = 0 ;
                    TOPO = 2 ;
                   END ;
              IF TIPO = 70 /* "ELSE" ? */
              THEN DO ;
                    CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                    IF TIPO ¬= 10 & TIPO ¬= 67 & TIPO ¬= 63 &
                    TIPO ¬= 70 & TIPO ¬= 0
                    /* NAO E' ";" NEM "END" NEM "UNTIL"  */
                    /* NEM "ELSE" NEM "EOF"              */
                    THEN DO ;
                          IF GERA & TOPO ¬= 0
                          THEN DO ;
                                /* GRAVA INSTRUCOES GUARDADAS */
                                IF PROXINT > LSKINT
                                THEN DO ;
                                     /* PLAPP23 - RESTRICAO DE IM-*/
                                     /* PLEMENTACAO - NUM. MAX.   */
                                     /* CTES INTEIRAS NOS BLOCOS  */
                                     /* ATIVOS FOI ULTRAPASSADO   */
                                     ERRO = 135 ;
                                     COLUNA = PTR - 1 ;
                                     CALL PLAPP01 ; /*IMP.MSG.ERRO*/
                                     ERRO = 41 ; /* COMP.ABORTADA */
                                     CALL PLAPP01 ; /*IMP.MSG.ERRO*/
                                     STOP ;
                                      END ;
                                OP3(TOPO-1) = PROXINT ;
                                DO I = 1 TO TOPO ;
                                  CALL PLAPP34(C(I),0,OP1(I),
                                               RB2(I),OP2(I),RB3(I),
                                               OP3(I)) ;
                                END ;
                                IF REF ¬= 0
                                THEN /* RESOLVE REFERENCIA */
                                     /* ANTERIOR          */
                                     VALINT(REF) = CINST + LC1 ;
                                REF = PROXINT ;
                                PROXINT = PROXINT + 1 ;
                               END ;
                           CALL PLAPP17 ; /* REC. DE COMANDO */
                          END ;
                    END ;
               IF REF ¬= 0
               THEN /* RESOLVE REFERENCIA ANTERIOR */
                    VALINT(REF) = CINST + LC1 ;
              END ;
     END ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
              ('PLAPP23 (RECONHECEDOR DE COMANDO "IF") RETORNOU' ,
              ' COM TIPO =' , TIPO ) (A , A , F(3)) ;
END PLAPP23 ;
```

```
/* PLAPP24 - RECONHECEDOR DE COMANDO 'CASE'                        */
PLAPP24 : PROC RECURSIVE ;
/*****************************************************************/
/*                                                               */
/* ESTA PROCEDURE RECONHECE COMANDO 'CASE'.                      */
/* E' CHAMADA COM A PALAVRA RESERVADA 'CASE' LIDA E RETORNA COM  */
/* O SIMBOLO QUE SEGUE O 'END' QUE ENCERRA O COMANDO.            */
/* E' CHAMADA PELO RECONHECEDOR DE COMANDOS (PLAPP17).           */
/*                                                               */
/*****************************************************************/
1 /* IMP. MENSAGEM DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO TAB. SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MANIPULACAO TAB. CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECONHECEDOR DE COMANDOS */
$CONTINUE WITH PLAMP17 RETURN
1 /* RECONHECEDOR DE EXPRESSAO */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO EM OBJ1 */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TIPEXP, PTMAT E TAMCHAR */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL     ESTADO     BIN FIXED (15,0) STATIC ;
 DCL     TIPCTE     BIN FIXED (15,0) STATIC ;
 DCL     TIPOSEL    BIN FIXED (15,0) ;
 DCL  SBUOPND       BIN FIXED(15,0) ;
 DCL  SEUOPND       BIN FIXED(15,0) ;
 DCL  SBXUOPND      BIN FIXED(15,0) ;
 DCL  SXUOPND       BIN FIXED(15,0) ;
 DCL  REFINI        BIN FIXED(15,0) ;
 DCL  REFEND        BIN FIXED(15,0) ;
 DCL  1 LIST ,
        2 PINT(0:31)        BIN FIXED(31,0) ,
        2 PCHAR(0:31)       CHAR(1) ,
        2 PEND(0:31)        BIN FIXED(15,0) ,
        2 TOPO              BIN FIXED(15,0) ;
```

```
DCL LIMTOPO          BIN FIXED(15,0) STATIC INIT(31) ;
DCL SAVTOPO          BIN FIXED(15,0) STATIC ;
DCL (AUX,I)          BIN FIXED(15,0) STATIC ;
DCL PILHA_CHEIA      ENTRY RETURNS(BIT(1)) ;
DCL    CASE (4)  LABEL INIT (CASE1,CASE2,CASE3,CASE4) AUTOMATIC ;
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('PLAPP24 (RECONHECEDOR DE COMANDO "CASE") INICIOU.') (A) ;
/* PULA "CASE" */
CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
CALL PLAPP30 ; /* RECONHECEDOR DE EXPRESSAO */
IF TIPEXP = 1 | TIPEXP = 4
THEN TIPOSEL = TIPEXP ;
ELSE DO ;
       ERRO = 107 ; /* EXPRESSAO NAO E' INTEIRA NEM CHAR */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       TIPOSEL = 0 ;
     END ;
IF TIPO ¬= 58 /* SE NAO E' 'OF' */
THEN DO ;
       ERRO = 59 ; /* FALTA 'OF'. INSERIDO. */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF GERA
THEN DO ;
       /* GUARDA END DO RESULTADO DA EXPRESSAO SELETORA; GUARDA */
       /* REFERENCIA INICIAL; GERA DESVIO INCONDICIONAL INDEXA- */
       /* DO P/TESTE; GUARDA REFERENCIA P/FIM DO CASE; INICIALI-*/
       /* ZA E GUARDA TOPO                                     */
       SBUOPND = BUOPND ;
       SEUOPND = EUOPND ;
       SBXUOPND = BXUOPND ;
       SXUOPND = XUOPND ;
       IF PROXINT + 1 > LSKINT
       THEN DO ;
              /* PLAPP24 - RESTRICAO DE IMPLEMENTACAO - NUMERO  */
              /* MAXIMO PERMITIDO DE CTES INTEIRAS NOS BLOCOS   */
              /* ATIVOS FOI ULTRAPASSADO                        */
              ERRO = 139 ;
              COLUNA = PTR - 1 ;
              CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
              ERRO = 41 ; /* COMPILACAO ABORTADA */
              CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
              STOP ;
            END ;
       REFINI = PROXINT ;
       CALL PLAPP34(COSET,0,0,0,3,0,REFINI) ;
       CALL PLAPP34(CODESV,0,1,0,CCQQR,0,0) ;
       REFEND = PROXINT + 1 ;
       PROXINT = PROXINT + 2 ;
       SAVTOPO = 0 ;
     END ;
TOPO = 0 ;
ESTADO = 1 ;
DO WHILE (TIPO¬=67 & TIPO¬=0) ; /* ENQUANTO NAO FOR 'END' NEM EOF*/
  IF ESTADO = 1 /* PEGOU 'OF' OU ';' E ESPERA CONSTANTE OU 'END'*/
  THEN DO ;
```

```
          IF TIPO = 13 /* ':' */
          THEN DO ;
                 ERRO = 99 ; /* FALTA ROTULO DO COMANDO */
                 COLUNA = PTR - 1 ;
                 CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               END ;
          DO WHILE (TIPO¬=13 & TIPO¬=10 & TIPO¬=67 & TIPO¬=0);
          /* ENQUANTO NAO FOR ':' NEM ';' NEM 'END' NEM EOF */
          /* RECONHECE CONSTANTE */
          IF TIPO = 20 | TIPO = 21
          THEN DO ; /* PEGOU + OU - */
                 ERRO = 120 ; /*SINAL ANTES DE ROTULO EM COMANDO */
                 COLUNA = PTR - 1 ; /* "CASE". SINAL IGNORADO.   */
                 CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                 CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
               END ;
          IF TIPO >= 1 & TIPO <= 4
          THEN DO ;
                 GOTO CASE(TIPO) ;
CASE1 : /* PEGOU IDENTIFICADOR */
                 IDNOME = IDENT ;
                 CALL PLAPP05 ; /* PROCURA IDENTIFICADOR NA */
                                /* TABELA DE SIMBOLOS       */
                 IF ¬ACHOU
                 THEN DO ;
                        /* COLOCA TIPO DE CONSTANTE INDEFINIDO */
                        /* (NAO DECLARADO) NA TAB. DE SIMBOLOS */
                        IDTIPO(IND) = -1 ;
                        TIPCTE = 0 ; /* TIPO DE CTE INDEFINIDO */
                      END ;
                 ELSE IF IDTIPO(IND) > 4 /* TIPO DA TABELA DE  */
                      THEN DO ;      /* SIMBOLOS NAO E' CONSTANTE*/
                             TIPCTE = 0 ; /*TIPO INDEF. DE CTE */
                             ERRO = 100 ; /* ROTULO INVALIDO */
                             COLUNA = PTR - 1 ;
                             CALL PLAPP01 ; /* IMPRIME MSG ERRO*/
                           END ;
                      ELSE DO ;
                             TIPCTE = IDTIPO(IND) ;
                             IF PILHA_CHEIA
                             THEN STOP ;
                             IF TIPCTE = 1 /* INTEIRO ? */
                             THEN PINT(TOPO) = VALINT(PTCTE(IND)) ;
                             ELSE IF TIPCTE = 4 /* CHAR ? */
                                  THEN PCHAR(TOPO) =
                                             TEXTO(PTALFA(PTCTE(IND)));
                           END ;
                 GOTO ENDCASE ;
CASE2 : /* PEGOU CONSTANTE INTEIRA */
                 TIPCTE = 1 ; /* CONSTANTE INTEIRA OK */
                 IF PILHA_CHEIA
                 THEN STOP ;
                 PINT(TOPO) = NINT ;
                 GOTO ENDCASE ;
CASE3 : /* PEGOU CONSTANTE REAL */
                 TIPCTE = 2 ; /* CONSTANTE REAL OK */
                 GOTO ENDCASE ;
CASE4 : /* PEGOU LITERAL */
                 IF TAM ¬= 1
                 THEN DO ;
```

```
                    ERRO = 121 ; /* DADO ALFANUMERICO C/MAIS */
                                 /* DE UM CARACTER E' PERMI- */
                                 /* TIDO APENAS EM COMANDO   */
                                 /* "WRITE".                 */
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMPRIME MENSAGEM ERRO */
                    TIPCTE = 0 ; /* CONSTANTE INDEFINIDA */
                  END ;
                ELSE DO ;
                    TIPCTE = 4 ;
                    IF PILHA_CHEIA
                    THEN STOP ;
                    PCHAR(TOPO) = STRING(1) ;
                  END ;
ENDCASE :
                IF TIPCTE = 0 | TIPCTE = 1 | TIPCTE = 4
                THEN DO ; /* CTE INDEF.,INTEIRA OU CHAR */
                    IF TIPOSEL ¬= 0 & TIPCTE ¬= 0
                    THEN IF TIPOSEL ¬= TIPCTE
                         THEN DO ;
                                 /* CTE NAO E' DO MESMO TIPO   */
                                 /* DA EXPRESSAO SELETORA.     */
                                 ERRO = 109 ;
                                 COLUNA = PTR - 1 ;
                                 CALL PLAPP01 ; /* IMP.MSG.ERRO*/
                              END ;
                         ELSE DO ;
                                 /* VERIFICA DUPLICIDADE */
                                 I = 0 ;
                                 IF TIPOSEL = 1 /* INTEIRO ? */
                                 THEN DO WHILE
                                          (PINT(I) ¬= PINT(TOPO));
                                        I = I + 1 ;
                                      END ;
                                 ELSE DO WHILE
                                          (PCHAR(I)¬=PCHAR(TOPO));
                                        I = I + 1 ;
                                      END ;
                                 IF I ¬= TOPO
                                 THEN DO ;
                                        /* DUPLICIDADE DE ROTU-*/
                                        /* LO EM COMANDO "CASE"*/
                                        ERRO = 142 ;
                                        COLUNA = PTR - 1 ;
                                        /* IMPRIME MSG ERRO */
                                        CALL PLAPP01 ;
                                      END ;
                                 TOPO = TOPO + 1 ;
                              END ;
                  END ;
                ELSE DO ; /* CTE NAO E' INTEIRA NEM CHAR */
                    ERRO = 108 ;
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO*/
                  END ;
                CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
              END ;
            ELSE IF TIPO = 9
                 THEN DO ; /* ',' */
                         ERRO = 99 ; /* FALTA ROTULO */
```

```
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
              END ;
          ELSE DO ;
                ERRO = 100 ; /* ROTULO INVALIDO */
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
              END ;
      IF TIPO ¬= 9 & TIPO ¬= 13
      THEN DO ; /* NEM ',' NEM ':' */
             ERRO = 54 ; /* SIMBOLO INVALIDO */
                         /* ',' OU ':' ESPERADO*/
             COLUNA = PTR - 1;
             CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
             DO WHILE (TIPO¬=10 & TIPO¬=13 & TIPO¬=9 &
                       TIPO¬=67 & TIPO¬=0) ;
               /* PROCURA ';', ':', ',' OU 'END' OU EOF */
               CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
             END ;
             ERRO = 14 ; /* SIMB.ANTER.IGNORADOS*/
             COLUNA = PTR - 1 ;
             CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
           END ;
      DO WHILE (TIPO = 9) ;
        /* PULA VIRGULAS CONSECUTIVAS */
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        IF TIPO = 9 | TIPO = 10 | TIPO = 67 | TIPO = 13
        THEN DO ; /* ',', ';', 'END' OU ':' */
               ERRO = 99 ; /* FALTA ROTULO */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
             END ;
      END ;
    END ;
    IF TIPO = 13 /* ":" */
    THEN DO ;
           ESTADO = 4 ;
           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
         END ;
    ELSE DO ;     /* 'END' OU ';' */
           ERRO = 76 ; /* FALTA ':'.INSERIDO */
           COLUNA = PTR - 1 ;
           CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
           IF TIPO = 10 /* ';' */
           THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
         END ;
  END ;
ELSE /* ESTADO = 4. PEGOU ':'. ESPERA 'END', ';' OU COMANDO */
    IF TIPO = 10 /* ';' */
    THEN DO ;
           ESTADO = 1 ;
           CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
           IF GERA
           THEN DO ;
                  /* COMPLETA PILHA C/INDICACAO DE QUE END */
                  /* DEVE SER SUBSTITUIDO PELO END DO FIM  */
                  /* DO "CASE" E ATUALIZA SAVTOPO          */
                  DO I = SAVTOPO TO TOPO - 1 ;
                    PEND(I) = -1 ;
```

```
                    END ;
                    SAVTOPO = TOPO ;
                  END ;
            END ;
         ELSE DO ;
                IF GERA
                THEN DO ;
                       /* COMPLETA PILHA C/END DO COMANDO E ATUA-*/
                       /* LIZA SAVTOPO                          */
                       AUX = CINST + LC1 ;
                       DO I = SAVTOPO TO TOPO - 1 ;
                         PEND(I) = AUX ;
                       END ;
                       SAVTOPO = TOPO ;
                     END ;
                CALL PLAPP17 ; /* RECONHECEDOR DE COMANDOS */
                IF TIPO ¬= 10 & TIPO ¬= 67
                THEN DO ; /* NEM ';' NEM 'END' */
                       ERRO = 12 ; /* FALTA ";". INSERIDO */
                       COLUNA = PTR - 1;
                       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                       ESTADO = 1;
                     END ;
                ELSE IF TIPO = 10 /* ';' */
                     THEN DO ;
                            ESTADO = 1;
                            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                          END;
                IF GERA
                THEN DO ; /*GERA INST DESV INCOND INDEX.P/FIM CASE*/
                       CALL PLAPP34(COSET,0,0,0,3,0,REFEND) ;
                       CALL PLAPP34(CODESV,0,1,0,CCQQR,0,0) ;
                     END ;
              END ;
END ;
IF GERA
THEN DO ;
       /* RESOLVE REFERENCIA INICIAL */
       VALINT(REFINI) = CINST + LC1 ;
       /* GERA TABELA DE ROTULOS */
       IF TIPOSEL = 1 /* EXPRESSAO INTEIRA ? */
       THEN DO ;
              IF PROXINT + TOPO > LSKINT
              THEN DO ;
                     /* PLAPP24 - RESTRICAO DE IMPLEMENTACAO -  */
                     /* NUM.MAX.PERMITIDO CTES INTEIRAS NOS BLO-*/
                     /* COS ATIVOS FOI ULTRAPASSADO             */
                     ERRO = 139 ;
                     COLUNA = PTR - 1 ;
                     CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                     ERRO = 41 ; /* COMPILACAO ABORTADA */
                     CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                     STOP ;
                   END ;
              DO I = 0 TO TOPO - 1 ;
                VALINT(PROXINT+I) = PINT(I) ;
              END ;
              SAVTOPO = PROXINT ; /* POS.INICIAL DA TAB ROTULOS */
              PROXINT = PROXINT + TOPO + 1 ;
            END ;
```

```
ELSE DO ; /* EXPRESSAO CHAR */
        CTELIT(1) = ' ' ;
        TAMLIT = 1 ;
        CALL PLAPP15 ; /* INSERE NA TAB CTES ALFA P/ATUA-*/
                       /* LIZAR PONTEIROS                */
        IF ENDCTE + TOPO > LSKCHT
        THEN DO ;
               /* PLAPP24 - RESTRICAO DE IMPLEMENTACAO -  */
               /* NUM.MAX.PERMITIDO DE CTES ALFA.NOS BLO- */
               /* COS ATIVOS FOI ULTRAPASSADO             */
               ERRO = 141 ;
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               ERRO = 41 ; /* COMPILACAO ABORTADA */
               CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
               STOP ;
             END ;
        DO I = 0 TO TOPO - 1 ;
          TEXTO(ENDCTE+I) = PCHAR(I) ;
        END ;
        TAMALFA(POSCTE) = TOPO + 1 ;
        SAVTOPO = ENDCTE ;
      END ;
/* GERA COMPARACAO E DESVIO INCONDICIONAL INDEXADO. A     */
/* COMPARACAO SERA' FEITA POR MEIO DE PESQUISA SEQUENCIAL*/
/* DIRETA, TENDO SIDO COLOCADO O ARGUMENTO DE BUSCA NO    */
/* FIM DA TABELA                                          */
CALL PLAPP34(COSET,0,0,0,2,SBXUOPND,SXUOPND) ;
CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
CALL PLAPP34(COATRIB,0,TIPOSEL,SBUOPND,SEUOPND,0,
             SAVTOPO+TOPO) ;
CALL PLAPP34(COSET,0,0,0,2,0,ENDZERO) ;
CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
AUX = CINST + LC1 ; /* END INST COMPARACAO */
CALL PLAPP34(COCOMP,0,TIPOSEL,0,SAVTOPO,0,SAVTOPO+TOPO) ;
CALL PLAPP34(CODESV,0,0,0,CCEQ,0,CINST+LC1+3) ;
/* GERA CTE INTEIRA 1 */
CTEINT = 1 ;
CALL PLAPP13 ; /* INSERE NA TAB CTES INTEIRAS */
CALL PLAPP34(COADREG,0,0,0,2,0,ENDCTE) ;
CALL PLAPP34(CODESV,0,0,0,CCQQR,0,AUX) ;
TEMINT = TEMINT - 1 ;
CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
CALL PLAPP34(COATRIB,0,1,0,PROXINT,BLOCO+1,TEMINT) ;
CALL PLAPP34(COSET,0,0,0,3,BLOCO+1,TEMINT) ;
CALL PLAPP34(CODESV,0,1,0,CCQQR,0,CINST+LC1+1) ;
/* RESOLVE REFERENCIA AO FIM DO COMANDO */
VALINT(REFEND) = CINST + LC1 ;
REFEND = CINST + LC1 ;
/* VERIFICA SE TAB DE ENDS CABE NA TAB DE CTES INTEIRAS */
IF PROXINT + TOPO > LSKINT
THEN DO ;
        /* PLAPP24 - RESTRICAO DE IMPLEMENTACAO - NUM.MAX.*/
        /* PERMITIDO DE CTES INTEIRAS NOS BLOCOS ATIVOS   */
        /* FOI ULTRAPASSADO                               */
        ERRO = 139 ;
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        ERRO = 41 ; /* COMPILACAO ABORTADA */
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
```

```
            STOP ;
          END ;
        /* GERA TABELA DE ENDERECOS DE DESVIO */
        DO I = 0 TO TOPO - 1 ;
          IF PEND(I) = -1
          THEN VALINT(PROXINT+I) = 0 ;
          ELSE VALINT(PROXINT+I) = PEND(I) - REFEND ;
        END ;
      END ;
/* PULA "END" */
CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
PILHA_CHEIA : PROC RETURNS(BIT(1)) ;
DCL RETCODE          BIT(1) ;
IF TOPO > LIMTOPO
THEN DO ;
        /* PLAPP24 - RESTRICAO DE IMPLEMENTACAO - NUM.MAX.PERMI- */
        /* TIDO DE ROTULOS EM COMANDO "CASE" FOI ULTRAPASSADO    */
        ERRO = 140 ;
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        ERRO = 41 ; /* COMPILACAO ABORTADA */
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        RETCODE = '1'B ;
      END ;
ELSE RETCODE = '0'B ;
RETURN(RETCODE) ;
END PILHA_CHEIA ;
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('PLAPP24 (RECONHECEDOR DE COMANDO "CASE") RETORNOU',
      ' COM TIPO =' , TIPO) (A , A , F(3));
END PLAPP24 ;
```

```
/*  PLAPP25 - RECONHECEDOR DE COMANDO "FOR".                        */
PLAPP25 : PROC RECURSIVE ;
/*****************************************************************/
/*                                                               */
/*  ESTA PROCEDURE RECONHECE COMANDO "FOR".                      */
/*  E' CHAMADA PELO RECONHECEDOR DE COMANDOS (PLAPP17), COM A     */
/*  PALAVRA RESERVADA "FOR" LIDA, E RETORNA NO SIMBOLO APOS O    */
/*  COMANDO.                                                     */
/*                                                               */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MANIPULACAO DAS TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECONHECEDOR DE COMANDOS */
$CONTINUE WITH PLAMP17 RETURN
1 /* RECONHECEDOR DE EXPRESSAO */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO EM OBJ1 */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE   */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TIPEXP, PTMAT E TAMCHAR */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL EVCNTR            BIN FIXED(15,0) AUTOMATIC ;
 DCL BEVCNTR           BIN FIXED(15,0) AUTOMATIC ;
 DCL ENDINC            BIN FIXED(15,0) AUTOMATIC ;
 DCL ENDEXP1           BIN FIXED(15,0) STATIC ;
 DCL BENDEXP1          BIN FIXED(15,0) STATIC ;
 DCL INDEXP1           BIN FIXED(15,0) STATIC ;
 DCL BINDEXP1          BIN FIXED(15,0) STATIC ;
 DCL ENDDESV           BIN FIXED(15,0) AUTOMATIC ;
 DCL ENDTEST           BIN FIXED(15,0) AUTOMATIC ;
 DCL CODDESV           BIN FIXED(15,0) STATIC ;
 IF TRACE (1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP25(RECONHECEDOR DE COMANDO "FOR") INICIOU') (A) ;
 CALL PLAPP04 ; /* RECONHECEDOR LEXICO P/ PULAR O "FOR" */
```

```
IF TIPO ¬= 1 /* NAO E' IDENTIFICADOR */
THEN DO ;
       ERRO = 96 ; /* VAR. DE CONTROLE ESPERADA. */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ;
     END ;
ELSE DO ;
       IDNOME = IDENT ;
       CALL PLAPP05 ; /* PROCURA NA TABELA */
       IF ACHOU
       THEN DO ;
              IF IDTIPO(IND) ¬= 41 & IDTIPO(IND) ¬= 81 &
                 IDTIPO(IND) ¬= 91
              THEN DO ; /* NAO E' VARIAVEL SIMPLES INTEIRA */
                     ERRO = 97 ;
                     COLUNA = PTR - 1 ;
                     CALL PLAPP01 ;
                   END ;
            END ;
        ELSE IDTIPO (IND) = -1 ; /* IDENT. NAO DECLARADO */
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
     END ;
IF TIPO ¬= 16 /* NAO E' ":=" */
THEN DO ;
       ERRO = 90 ; /* ESPERADO ':='. INSERIDO */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF GERA
THEN DO ;
       EVCNTR = ENDER(IND) ;
       BEVCNTR = NIVEL(IND) + 1 ;
     END ;
CALL PLAPP30 ; /* RECONHECEDOR DE EXPRESSAO */
IF TIPEXP > 1
THEN DO ;
       ERRO = 110 ; /* EXPRESSAO NAO E' INTEIRA */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
IF GERA
THEN DO ; /* GUARDA ENDERECO DA EXPRESSAO 1 */
       ENDEXP1 = EUOPND ;
       BENDEXP1 = BUOPND ;
       INDEXP1 = XUOPND ;
       BINDEXP1 = BXUOPND ;
     END ;
IF TIPO = 73
THEN DO ; /* "DOWNTO" */
       IF GERA
       THEN DO ;
              CTEINT = -1 ;
              CODDESV = CCLT ;
            END ;
       CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
     END ;
ELSE IF TIPO = 74
     THEN DO ; /* "TO" */
            IF GERA
```

```
          THEN DO ;
                 CTEINT = 1 ;
                 CODDESV = CCGT ;
               END ;
          CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        END ;
      ELSE DO ; /* "TO" OU "DOWNTO" ESPERADO. INSERIDO "TO" */
          ERRO = 98 ;
          COLUNA = PTR - 1 ;
          CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        END ;
IF GERA
THEN DO ;
       CALL PLAPP13 ; /* INSERE CTE NA TAB DE CTES INTEIRAS */
       ENDINC = ENDCTE ; /* END DA CTE CRIADA */
     END ;
CALL PLAPP30 ; /* RECONHECEDOR DE EXPRESSAO */
IF TIPEXP > 1
THEN DO ;
       ERRO = 110 ; /* EXPRESSAO NAO E' INTEIRA */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ;
     END ;
IF TIPO ¬= 72 /* NAO E' "DO" */
THEN DO ;
       ERRO = 95 ; /* 'DO' ESPERADO. INSERIDO */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ;
     END ;
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TIPO ¬= 10 & TIPO ¬= 67 & TIPO ¬= 63 & TIPO ¬= 70 & TIPO ¬= 0
   /* SE NAO E' ';' NEM 'END' NEM 'UNTIL' NEM 'ELSE' NEM EOF */
THEN DO ;
       IF GERA
       THEN DO ;
              IF TUOPND = IVI | TUOPND = IVD
              THEN DO ; /* EXPRESSAO 2 E' VARIAVEL */
                     /* GERA ATRIBUICAO P/TEMPORARIO */
                     /* GERA INSTRUCAO SET REG. 2 */
                     CALL PLAPP34(COSET,0,0,0,2,BXUOPND,XUOPND);
                     /* GERA INSTRUCAO SET REG. 3 */
                     CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
                     /* GERA INSTRUCAO ATRIB INTEIRA */
                     TEMINT = TEMINT - 1 ; /* END DO TEMPORARIO */
                     CALL PLAPP34(COATRIB,0,1,BUOPND,EUOPND,
                                  BLOCO+1,TEMINT) ;
                     EUOPND = TEMINT ;
                     BUOPND = BLOCO + 1 ;
                   END ;
              /* GERA ATRIB P/VARIAVEL DE CONTROLE */
              /* GERA INSTRUCAO SET REG. 2 */
              CALL PLAPP34(COSET,0,0,0,2,BINDEXP1,INDEXP1) ;
              /* GERA INSTRUCAO SET REG. 3 */
              CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
              /* GERA INSTRUCAO ATRIB INTEIRA */
              CALL PLAPP34(COATRIB,0,1,BENDEXP1,ENDEXP1,BEVCNTR,
                           EVCNTR) ;
              /* GUARDA ENDERECO DA INSTRUCAO DE COMPARACAO */
              ENDTEST = CINST + LC1 ;
              /* GERA INSTRUCAO DE COMPARACAO */
```

```
            /* GERA INSTRUCAO  SETM */
            CALL PLAPP34(COZEREG,0,0,0,1,0,1) ;
            /* GERA INSTRUCAO DE COMPARACAO INTEIRA */
            CALL PLAPP34(COCOMP,0,1,BEVCNTR,EVCNTR,BUOPND,
                         EUOPND) ;
            /* GERA INSTRUCAO DESVIO INDEXADO */
            IF PROXINT > LSKINT
            THEN DO ;
                   /* PLAPP25 - RESTRICAO DE IMPLEMENTACAO -  */
                   /* NUMERO MAXIMO DE CONSTANTES INTEIRAS    */
                   /* NOS BLOCOS ATIVOS FOI ULTRAPASSADO      */
                   ERRO = 134 ;
                   COLUNA = PTR - 1 ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   ERRO = 41 ; /* COMPILACAO ABORTADA */
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   STOP ;
                 END ;
            ENDDESV = PROXINT ;
            PROXINT = PROXINT + 1 ;
            CALL PLAPP34(COSET,0,0,0,3,0,ENDDESV) ;
            CALL PLAPP34(CODESV,0,1,0,CODDESV,0,0) ;
          END ;
      CALL PLAPP17 ; /* RECONHECEDOR DE COMANDO */
      IF GERA
      THEN DO ;
            /* INCREMENTA VAR DE CONTROLE E VOLTA P/COMPARACAO */
            /* GERA INSTRUCAO SETM, ADI E DESV INCONDICIONAL */
            CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
            CALL PLAPP34(COADI,0,ENDINC,BEVCNTR,EVCNTR,
                         BEVCNTR,EVCNTR) ;
            CALL PLAPP34(CODESV,0,0,0,CCQQR,0,ENDTEST) ;
            /* RESOLVE REFERENCIA ANTERIOR */
            VALINT(ENDDESV) = CINST + LC1 ;
          END ;
    END ;
IF TRACE (1)
THEN PUT FILE (SPRINT) SKIP EDIT
     ('PLAPP25 (RECONHECEDOR DE COMANDO "FOR") RETORNOU',
              ' COM TIPO =' , TIPO ) (A , A , F(3));
END PLAPP25 ;
```

```
/* PLAPP26 - RECONHECEDOR DE COMANDO 'READ'                        */
PLAPP26 : PROC ;
/*****************************************************************/
/*                                                               */
/* ESTA PROCEDURE RECONHECE COMANDO 'READ'.                      */
/* E' CHAMADA COM A PALAVRA RESERVADA "READ" LIDA E RETORNA      */
/* COM O SIMBOLO IMEDIATAMENTE APOS O COMANDO.                   */
/* E' CHAMADA PELO RECONHECEDOR DE COMANDOS (PLAPP17).           */
/*                                                               */
/*****************************************************************/
1 /* IMP. MENSAGEM DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MANIPULACAO DAS TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECONHECEDOR DE INDICES DE ARRAY */
$CONTINUE WITH PLAMP29 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO EM OBJ1 */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM E PROXDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL AUX                  BIN FIXED(15,0) STATIC ;
 DCL SALVA                BIN FIXED(15,0) STATIC ;
 DCL ENDIND               BIN FIXED(15,0) STATIC ;
 DCL TEND                 BIN FIXED(15,0) STATIC ;
 DCL BIND                 BIN FIXED(15,0) STATIC ;
 DCL SALTIP               BIN FIXED(15,0) STATIC ;
 DCL PTL                  BIN FIXED(15,0) STATIC ;
 DCL NUMOPND              BIN FIXED(15,0) STATIC ;
 DCL (L,M,N,J)            BIN FIXED(15,0) STATIC ;
 DCL BAUX                 BIN FIXED(15,0) STATIC ;
 DCL 1 LIST STATIC ,
       2 LISTA(32) ,
         3 ENDEXP             BIN FIXED(15,0) ,
         3 RBASE              BIN FIXED(15,0) ,
         3 TENDEXP            BIN FIXED(15,0) ,
         3 NELEM              BIN FIXED(15,0) ,
```

```
      2 PTL                BIN FIXED(15,0) ,
      2 LIMPTL             BIN FIXED(15,0) INIT(32) ;
DCL MOD                  BUILTIN ;
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
        ('PLAPP26 (RECONHECEDOR DE COMANDO "READ") INICIOU.')(A);
IF GERA
THEN DO ;
       SALTIP = TIPO ;
       PTL = 0 ;
       NUMOPND = 0 ;
     END ;
CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TIPO ¬= 11 /* NAO E' "(" */
THEN DO ;
       ERRO = 92 ; /* FALTA "(". INSERIDO */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
     END ;
ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TIPO = 12 | TIPO = 10 | TIPO = 67
THEN DO ; /* ")", ";" OU "END" LOGO APOS "(" */
       ERRO = 63 ; /* LISTA DE VARIAVEIS VAZIA */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       IF TIPO = 12
       THEN CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
     END ;
ELSE DO ;
      DO WHILE (TIPO ¬= 12 & TIPO ¬= 10 & TIPO ¬= 67 & TIPO¬=0) ;
       /* ENQUANTO NAO FOR ')' NEM ';' NEM 'END' NEM EOF */
       IF TIPO = 9 /* SE PEGOU "," */
       THEN DO ;
              ERRO = 82 ; /* FALTA VARIAVEL */
              COLUNA = PTR - 1 ;
              CALL PLAPP01 ; /* IMP. MSG ERRO */
              CALL PLAPP04 ; /* REC. LEXICO */
            END ;
       ELSE DO ;
              IF TIPO = 1 /* IDENTIFICADOR */
              THEN DO ;
                     IDNOME = IDENT ;
                     CALL PLAPP05 ; /* PROCURA NA TABELA */
                                    /* SIMBOLOS          */
                     IF ¬ACHOU
                     THEN IDTIPO(IND) = -1 ; /* IDENT.NAO DECLARADO*/
                     AUX = IDTIPO (IND) ;
                     IF AUX >=  40 & AUX <=  44 /* VAR.SIMPLES   */ |
                        AUX >=  50 & AUX <=  54 /* VAR.INDEXADA  */ |
                        AUX >=  80 & AUX <=  84 /* PARAM.VAR.    */ |
                                                /* SIMP.P/END.   */
                        AUX >=  90 & AUX <=  94 /* PARAM.VAR.    */ |
                                                /* SIMP.P/VALOR  */
                        AUX >= 100 & AUX <= 104 /* PARAM.VAR.    */ |
                                                /* INDEX.P/END.  */
                        AUX >= 110 & AUX <= 114 /* PARAM.VAR.    */ |
                                                /* INDEX.P/VALOR*/
                        AUX = -1 /* IDENTIFICADOR NAO DECLARADO */
                     THEN DO ;
                            SALVA = IND ;
```

```
                        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                        IF GERA
                        THEN DO ;
                                IF PTL = LIMPTL
                                THEN DO ;
                                        /* PLAPP26 - RESTRICAO    */
                                        /* DE IMPLEMENTACAO - NUM*/
                                        /* MAX.PERMITIDO DE ELEM.*/
                                        /* EM LISTA DE COMANDO    */
                                        /* READ FOI ULTRAPASSADO */
                                        ERRO = 138 ;
                                        COLUNA = PTR - 1 ;
                                        /* IMPRIME MSG DE ERRO */
                                        CALL PLAPP01 ;
                                        /* COMPILACAO ABORTADA */
                                        ERRO = 41 ;
                                        /* IMPRIME MSG DE ERRO */
                                        CALL PLAPP01 ;
                                        STOP ;
                                     END ;
                                PTL = PTL + 1 ;
                                N = MOD(AUX,10) ;
                                IF N = 1
                                THEN DO ; /* INTEIRO */
                                        M = IVD ;
                                        L = IVI ;
                                     END ;
                                ELSE IF N = 2
                                     THEN DO ; /* REAL */
                                             M = RVD ;
                                             L = RVI ;
                                          END ;
                                     ELSE IF N = 3
                                          THEN DO ; /* BOOL. */
                                                  M = BVD ;
                                                  L = BVI ;
                                               END ;
                                          ELSE DO ; /* CHAR */
                                                  M = CVD ;
                                                  L = CVI ;
                                               END ;
                             END ;
/* * * * * FORAM RETIRADOS OS BRANCOS INICIAIS * * * * * * * * * */
IF AUX >= 50 & AUX <= 54 | AUX > 100 | AUX = -1
THEN DO ; /* NOME DE ARRAY (OU IDENTIFICADOR NAO DECLARADO) */
        IF TIPO = 14 /* ABRE COLCHETE ? */
        THEN DO ;
                /* RECONHECE INDICES DE ARRAY */
                CALL PLAPP29(SALVA,ENDIND,TEND,BIND) ;
                IF GERA
                THEN DO ;
                        IF TEND = 3 /* INDIRETO ? */
                        THEN DO ;
                                M = L ;
                                /* GERA END.ABSOLUTO DO ELEMENTO */
                                TEMINT = TEMINT - 1 ;
                                BAUX = BLOCO + 1 ; /* BASE DA PROC. */
                                /* JOGA VALOR DO REG.BASE DO ARRAY  */
                                /* NUM TEMPORARIO                   */
                                /* DETERMINA TIPO DO REG.BASE */
```

```
                        IF M = IVI
                        THEN N = 1 ; /* INTEIRO VARIAVEL */
                        ELSE IF M = RVI
                             THEN N = 2 ; /* REAL VARIAVEL */
                             ELSE IF M = BVI
                                  THEN N = 3 ; /* BOOL VAR. */
                                  ELSE N = 4 ; /* CHAR VAR. */
                        CALL PLAPP34(COSTOB,0,N,0,NIVEL(SALVA)+
                                  1,BAUX,TEMINT) ;
                        CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                        /* GERA CTE C/END.DO ARRAY */
                        CTEINT = ENDER(SALVA) ;
                        CALL PLAPP13 ; /* INSERE TAB.CTES */
                                       /* INTEIRAS        */
                        CALL PLAPP34(COADI,0,ENDCTE,BAUX,
                                  TEMINT,BAUX,TEMINT) ;
                        CALL PLAPP34(COADI,BAUX,TEMINT,BIND,
                                  ENDIND,BAUX,TEMINT) ;
                        L = TEMINT ; /* END. DO OPERANDO */
                      END ;
                   ELSE DO ;
                        BAUX = BIND ;
                        IF TEND = 4 /* PARAMETRO ? */
                        THEN M = L ; /*TIPO DO END.=INDIRETO*/
                        L = ENDIND ; /* END. DO OPERANDO */
                      END ;
                 N = 1 ;
               END ;
          END ;
     ELSE IF GERA
          THEN DO ;
                 N = TABDIM(PTCTE(SALVA)+1) ; /* NUM.ELEM.ARRAY*/
                 IF AUX < 100 | AUX > 104
                 THEN DO ; /* VARIAVEL INDEXADA OU PARAMETRO  */
                           /* VARIAVEL INDEXADA POR VALOR     */
                        L = ENDER(SALVA) ; /* END.DO OPERANDO */
                        BAUX = NIVEL(SALVA) + 1 ; /* BASE DO  */
                                                  /* OPERANDO */
                      END ;
                 ELSE DO ; /* PARAM.VAR.INDEXADA P/ENDERECO */
                        M = L ; /* ENDERECO INDIRETO */
                        TEMINT = TEMINT - N ;
                        L = TEMINT ; /* END. DO OPERANDO */
                        BAUX = BLOCO + 1 ; /* BASE DA PROCEDURE*/
                        /* ZERA TODOS OS REGISTRADORES INDEXAD.*/
                        CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                        CALL PLAPP34(COATRIB,0,1,NIVEL(SALVA)+1,
                                  ENDER(SALVA),BAUX,TEMINT) ;
                        /* GERA LOOP P/CALCULAR END.ABSOLUTO */
                        /* DOS DEMAIS ELEMENTOS DO ARRAY     */
                        CALL PLAPP34(COSET,0,0,0,3,0,ENDUM) ;
                        CALL PLAPP34(COADI,BAUX,TEMINT,0,ENDUM,
                                  BAUX,TEMINT) ;
                        CALL PLAPP34(COADREG,0,0,0,1,0,ENDUM) ;
                        CALL PLAPP34(COADREG,0,0,0,3,0,ENDUM) ;
                        /* TABDIM(PTCTE(SALVA)+2) E' O END.DA */
                        /* CTE C/O NUMERO DE ELEM. DO ARRAY   */
                        CALL PLAPP34(COCREG,0,0,0,3,0,
                                  TABDIM(PTCTE(SALVA)+2)) ;
                        /* GERA DESVIO, SE ¬=, P/INSTRUCAO ADI */
```

```
                        CALL PLAPP34(CODESV,0,0,0,CCNEQ,0,
                                     CINST+LC1-4) ;
                    END ;
                END ;
        END ;
ELSE IF GERA
        THEN DO ;
                IF AUX > 80 & AUX <= 84 /* PARAM.VAR.SIMPLES P/END. */
                THEN M = L ; /* END. INDIRETO */
                L = ENDER(SALVA) ; /* END. DO OPERANDO */
                BAUX = NIVEL(SALVA) + 1 ; /* BASE DO OPERANDO */
                N = 1 ;
              END ;
 IF GERA
 THEN DO ;
          TENDEXP(PTL) = M ;
          ENDEXP(PTL) = L ;
          RBASE(PTL) = BAUX ;
          NELEM(PTL) = N ;
          NUMOPND = NUMOPND + N ;
        END ;
                        END ;
                    ELSE DO ;
                           ERRO = 101 ; /* IDENT.INVALIDO      */
                                        /* P/ LISTA DE 'READ' */
                           COLUNA = PTR - 1 ;
                           CALL PLAPP01 ; /* IMP. MSG. ERRO */
                           CALL PLAPP04 ; /* REC.LEXICO */
                        END ;
                  END ;
              IF TIPO ¬= 9 & TIPO ¬= 12
              THEN DO ;
                     IF TIPO = 1 /* IDENTIFICADOR */
                     THEN DO ;
                            ERRO = 84 ; /* FALTA ",". INSERIDA */
                            COLUNA = PTR - 1 ;
                            CALL PLAPP01 ; /* IMP MSG ERRO */
                          END ;
                     ELSE DO ;
                            ERRO = 91 ; /* SIMBOLO INVALIDO */
                            COLUNA = PTR - 1 ;
                            CALL PLAPP01 ; /* IMP MSG ERRO */
                            DO WHILE(TIPO¬=9 & TIPO¬=12 & TIPO ¬= 10 &
                                     TIPO¬=67 & TIPO¬=0) ;
                             /*PROCURA ',', ')', ';' OU 'END' OU EOF*/
                             CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                            END ;
                            ERRO = 14 ; /* SIMB. ANTERIORES */
                                        /* IGNORADOS        */
                            COLUNA = PTR - 1 ;
                            CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                          END ;
                   END ;
              IF TIPO = 9 /* SE PEGOU "," */
              THEN DO ;
                     CALL PLAPP04 ; /* REC. LEXICO */
                     IF TIPO = 10 | TIPO = 12 | TIPO = 67
                     THEN DO ; /* PEGOU ";", ")' OU "END"  */
                               /* DEPOIS DE ","            */
                          ERRO = 82 ; /* FALTA VARIAVEL */
```

```
                        COLUNA = PTR - 1 ;
                        CALL PLAPP01 ; /* IMP. MSG ERRO */
                      END ;
                  END ;
            END ;
      END ;
      IF TIPO ¬= 12
      THEN DO ; /* NAO E' ")" */
            ERRO = 103 ; /* FALTA ")". INSERIDO */
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ;/* IMPRIME MENSAGEM DE ERRO */
          END ;
      ELSE CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
      IF GERA
      THEN DO ;
            IF SALTIP = 91 /* READ ? */
            THEN SALTIP = COREAD ;
            ELSE IF SALTIP = 92 /* READP ? */
                 THEN SALTIP = COREADP ;
                 ELSE IF SALTIP = 93 /* READD ? */
                      THEN SALTIP = COREADD ;
                      ELSE SALTIP = COREADPD ;
            /* GERA INST DE READ - OPERANDO 1 = ARQUIVO (LEI- */
            /* TORA DE CARTOES), OPERANDO 2 = NUM OPERANDOS A */
            /* LER, OPERANDO 3 = PONTEIRO P/LISTA QUE DESCRE- */
            /* VE OS OPERANDOS                                */
            CALL PLAPP34(SALTIP,0,0,0,NUMOPND,0,PROXINT) ;
            IF PROXINT + NUMOPND * 3 > LSKINT + 1
            THEN DO ;
                   /* PLAPP26 - RESTRICAO DE IMPLEMENTACAO -  */
                   /* NUM.MAX.PERMITIDO DE CIES INTEIRAS NOS  */
                   /* BLOCOS ATIVOS FOI ULTRAPASSADO          */
                   ERRO = 136 ;
                   COLUNA = PTR - 1 ;
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   ERRO = 41 ; /* COMPILACAO ABORTADA */
                   CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
                   STOP ;
                 END ;
            DO N = 1 TO PTL ;
              DO J = 0 TO NELEM(N) - 1 ;
                /* TIPO DO END.DO OPERANDO */
                VALINT(PROXINT) = TENDEXP(N) ;
                /* END DO OPERANDO */
                VALINT(PROXINT+1) = ENDEXP(N) + J ;
                /* NUM.DO REG.BASE DO OPERANDO */
                VALINT(PROXINT+2) = RBASE(N) ;
                PROXINT = PROXINT + 3 ;
              END ;
            END ;
          END ;
    END ;
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
        ('PLAPP26 (RECONHECEDOR DE COMANDO "READ") RETORNOU COM',
         ' TIPO =' , TIPO) (A , A , F(3)) ;
END PLAPP26 ;
```

```
/* PLAPP27 - RECONHECEDOR DE COMANDO 'WRITE'                          */
PLAPP27 : PROC ;
/*********************************************************************/
/*                                                                   */
/* ESTA PROCEDURE RECONHECE COMANDO 'WRITE'.                         */
/* E' CHAMADA COM A PALAVRA RESERVADA "WRITE" LIDA E RETORNA         */
/* COM O SIMBOLO IMEDIATAMENTE APOS O COMANDO.                       */
/* E' CHAMADA PELO RECONHECEDOR DE COMANDOS (PLAPP17).               */
/*                                                                   */
/*********************************************************************/
1 /* IMP. MENSAGEM DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DAS TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECCNHECEDOR DE EXPRESSAO */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REG. INSTRUCOES EM 'OBJ1' */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABIMT, TABREAL E TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM E PROXDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO TIPEXP, PTMAT E TAMCHAR */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL SALTIP    BIN FIXED(15,0) STATIC ;
 DCL 1 LIST STATIC ,
       2 LISTA(32) ,
         3 TAMEXP          BIN FIXED(15,0) ,
         3 ENDEXP          BIN FIXED(15,0) ,
         3 RBASE           BIN FIXED(15,0) ,
         3 TENDEXP         BIN FIXED(15,0) ,
         3 NELEM           BIN FIXED(15,0) ,
       2 PTL             BIN FIXED(15,0) ,
       2 LIMPTL          BIN FIXED(15,0) INIT(32) ;
 DCL (NUMOPND,N,J)       BIN FIXED(15,0) STATIC ;
 DCL BAUX                BIN FIXED(15,0) ;
 IF TRACE(1)
 THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP27 (RECONHECEDOR DE COMANDO "WRITE") INICIOU.') (A) ;
 IF GERA
 THEN DO ;
        PTL = 0 ;
```

```
        NUMOPND = 0 ;
        SALTIP = TIPO ; /* GUARDA TIPO DO 'WRITE' */
      END ;
CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
IF TIPO ¬= 11 /* NAO E' "(" */
THEN DO ;
        ERRO = 92 ; /* FALTA "(". INSERIDO */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ;
      END ;
ELSE CALL PLAPP04 ; /* PULA "(" */
IF TIPO = 12 | TIPO = 10 | TIPO = 67 /* ")", ";" OU "END" */
THEN DO ; /* ")", ";" OU "END" LOGO APOS "(" */
        ERRO = 63 ; /* LISTA DE VARIAVEIS VAZIA */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ;
        IF TIPO = 12
        THEN CALL PLAPP04 ; /* PULA ")" */
      END ;
ELSE DO ;
        DO WHILE (TIPO ¬= 10 & TIPO ¬= 12 & TIPO ¬= 67 & TIPO¬=0) ;
          /* ENQUANTO NAO FOR ';' NEM ')' NEM 'END' NEM EOF */
          IF TIPO = 9   /* SE PEGOU "," */
          THEN DO ;
                 ERRO = 82 ; /* FALTA VARIAVEL */
                 COLUNA = PTR - 1 ;
                 CALL PLAPP01 ;
                 CALL PLAPP04 ; /* PULA VIRGULA */
               END ;
          ELSE DO ;
                 CALL PLAPP30 ; /* REC. EXPRESSAO */
                 IF GERA
                 THEN DO ; /* GUARDA OPERANDOS P/COMANDO IMPRESSAO*/
                        IF PTL = LIMPTL
                        THEN DO ; /* PLAPP27 - RESTRICAO DE IMPLE- */
                               /* MENTACAO - NUMERO MAXIMO PERMITIDO*/
                               /* DE ELEMENTOS EM LISTA DE COMANDO  */
                               /* "WRITE" FOI ULTRAPASSADO.         */
                               ERRO = 123 ;
                               COLUNA = PTR - 1 ;
                               CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                               STOP ;
                             END ;
                      PTL = PTL + 1 ;
                      TAMEXP(PTL) = TAMCHAR ;
                      TENDEXP(PTL) = TUOPND ;
                      IF TUOPND = IVI | TUOPND = RVI |
                         TUOPND = BVI | TUOPND = CVI
                      THEN DO ; /* ENDERECO INDIRETO */
                             IF TIPEXP > 100
                             THEN DO ; /* MATRICIAL (PARAMETRO) */
                                    N = TABDIM(PTMAT+1) ; /* NUM.DE*/
                                          /* ELEMENTOS DO ARRAY */
                                    TEMINT = TEMINT - N ;
                                    ENDEXP(PTL) = TEMINT ;
                                    BAUX = BLOCO + 1 ; /* BASE DA */
                                                 /* PROCEDURE*/
                                    RBASE(PTL) = BAUX ;
                                    CALL PLAPP34(COZEREG,0,1,0,1,
                                         0,1) ; /*ZERA TODOS REGS */
```

```
                CALL PLAPP34(COATRIB,0,1,BXUOPND,
                     XUOPND,BAUX,TEMINT) ;
                /* GERA LOOP P/CALCULAR END.   */
                /* ABSOLUTO DOS DEMAIS ELEMEN- */
                /* TOS DO ARRAY                */
                CALL PLAPP34(COSET,0,0,0,3,
                          0,ENDUM) ;
                CALL PLAPP34(COADI,BAUX,TEMINT,
                     0,ENDUM,BAUX,TEMINT) ;
                CALL PLAPP34(COADREG,0,0,0,1,0,
                          ENDUM) ;
                CALL PLAPP34(COADREG,0,0,0,3,0,
                          ENDUM) ;
                /* GERA CTE C/NUM.ELEM. ARRAY */
                CTEINT = N ;
                CALL PLAPP13 ; /* INSERE NA */
                       /* TAB CTES INTEIRAS */
                CALL PLAPP34(COCREG,0,0,0,3,0,
                          ENDCTE) ;
                /* GERA DESVIO, SE DIFERENTE,*/
                /* P/ INSTRUCAO ADI          */
                CALL PLAPP34(CODESV,0,0,0,CCNEQ,
                     0,CINST+LC1-4) ;
           END ;
      ELSE IF EUOPND ¬= 0
           THEN DO ; /* ELEM. DE ARRAY    */
                     /* (NAO PARAM. POR END) */
                     /* C/ INDICES VARIAVEIS */
                     /* GERA END.ABSOLUTO DO */
                     /* ELEMENTO             */
                     TEMINT = TEMINT - 1 ;
                     BAUX = BLOCO + 1 ;/* BASE */
                               /* DA PROCEDURE */
                     /* JOGA VALOR DO REG. BASE*/
                     /* DO ARRAY NUM TEMPORARIO*/
                     /* DETERMINA TIPO DO REG. */
                     /* BASE                   */
                     IF TUOPND = IVI
                     THEN J = 1 ;/* INTEIRO-VAR*/
                     ELSE IF TUOPND = RVI
                          THEN J = 2 ; /* REAL */
                                     /* VAR  */
                          ELSE IF TUOPND = BVI
                               THEN /* BOOL-VAR*/
                                    J = 3 ;
                               ELSE /* CHAR-VAR*/
                                    J = 4 ;
                     CALL PLAPP34(COSTOB,0,J,0,
                          BUOPND,BAUX,TEMINT) ;
                     CALL PLAPP34(COZEREG,0,1,0,
                          1,0,1) ;
                     /* GERA CTE C/END DO ARRAY*/
                     CTEINT = EUOPND ; /*END DO*/
                                /* ARRAY */
                     CALL PLAPP13 ; /* INSERE */
                      /* NA TAB DE CTES INT. */
                     CALL PLAPP34(COADI,0,ENDCTE,
                       BAUX,TEMINT,BAUX,TEMINT) ;
                     CALL PLAPP34(COADI,BAUX,
                          TEMINT,BXUOPND,XUOPND,
```

```
                                        BAUX,TEMINT) ;
                                    ENDEXP(PTL) = TEMINT ;
                                    RBASE(PTL) = BAUX ;
                                   END ;
                               ELSE DO ;
                                    ENDEXP(PTL) = XUOPND ;
                                    RBASE(PTL) = BXUOPND ;
                                   END ;
                          END ;
                     ELSE DO ;
                            ENDEXP(PTL) = EUOPND ;
                            RBASE(PTL) = BUOPND ;
                          END ;
                     IF TIPEXP < 50
                     THEN DO ; /* ESCALAR */
                            NELEM(PTL) = 1 ;
                            NUMOPND = NUMOPND + 1 ;
                          END ;
                     ELSE DO ; /* OPERANDO MATRICIAL */
                           /* CALCULA NUM.DE ELEMENTOS DE ARRAY */
                           N = TABDIM(PTMAT+1) ;
                           NELEM(PTL) = N ;
                           NUMOPND = NUMOPND + N ;
                          END ;
                    END ;
               IF TIPO ¬= 9 & TIPO ¬= 12
               THEN DO ;
                      ERRO = 102 ; /* SIMBOLO INVALIDO. */
                      COLUNA = PTR - 1 ; /* ',' OU ')' ESPERADO */
                      CALL PLAPP01 ;
                      DO WHILE(TIPO¬=9 & TIPO¬=12 & TIPO¬=10 &
                               TIPO¬=67 & TIPO¬=0) ;
                     /* PROCURA ',' , ')' , ';' OU 'END' OU EOF */
                        CALL PLAPP04 ;
                      END ;
                      ERRO = 14 ; /* SIMB. ANTERIORES IGNORADOS */
                      COLUNA = PTR - 1 ;
                      CALL PLAPP01 ;
                    END ;
               IF TIPO = 9 /* SE PEGOU "," */
               THEN DO ;
                      CALL PLAPP04 ; /* REC. LEXICO */
                      IF TIPO = 10 | TIPO = 12 | TIPO = 67
                      THEN DO ; /* PEGOU ";", ")" OU "END" */
                                /* DEPOIS DA ","           */
                             ERRO = 82 ; /* FALTA VARIAVEL */
                             COLUNA = PTR - 1 ;
                             CALL PLAPP01 ; /* IMP. MSG ERRO */
                            END ;
                    END ;
             END ;
        END ;
   IF TIPO ¬= 12
   THEN DO ; /* NAO E' ")" */
          ERRO = 103 ; /* FALTA ")". INSERIDO */
          COLUNA = PTR - 1 ;
          CALL PLAPP01 ;
        END ;
   ELSE CALL PLAPP04 ; /* PULA ")" */
   IF GERA
```

```
        THEN DO ;
             IF SALTIP = 81 /* WRITE */
             THEN SALTIP = COWRITE ;
             ELSE IF SALTIP = 82 /* WRITELN */
                  THEN SALTIP = COWRITELN ;
                  ELSE IF SALTIP = 83 /* WRITEPG */
                       THEN SALTIP = COWRITEPG ;
                       ELSE IF SALTIP = 84 /* WRITED */
                            THEN SALTIP = COWRITED ;
                            ELSE IF SALTIP = 85 /* WRITELND */
                                 THEN SALTIP = COWRITELND ;
                                 ELSE SALTIP = COWRITEPGD ;
             /* GERA INST. DE WRITE - OPERANDO 1 = ARQUIVO (IM- */
             /* PRESSORA), OPERANDO 2 = NUM. OPERANDOS A IMPRI- */
             /* MIR, OPERANDO 3 = PONTEIRO P/LISTA QUE DESCREVE */
             /* OS OPERANDOS                                    */
             CALL PLAPP34(SALTIP,0,0,0,NUMOPND,0,PROXINT) ;
             IF PROXINT + NUMOPND * 4 > LSKINT + 1
             THEN DO ; /* PLAPP27 - RESTRICAO DE IMPLEMENTACAO -*/
                       /* NUMERO MAXIMO PERMITIDO DE CONSTANTES */
                       /* INTEIRAS NO PROGRAMA FOI ULTRAPASSADO.*/
                  ERRO = 125 ;
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMP. MSG. ERRO */
                  ERRO = 41 ; /* COMPILACAO ABORTADA */
                  CALL PLAPP01 ; /* IMP. MSG. ERRO */
                  STOP ;
                END ;
             DO N = 1 TO PTL ;
              DO J = 0 TO NELEM(N) - 1 ;
                /* TIPO DO ENDERECO DO OPERANDO */
                VALINT(PROXINT) = TENDEXP(N) ;
                /* TAMANHO DO OPERANDO */
                VALINT(PROXINT+1) = TAMEXP(N) ;
                /* ENDERECO DO OPERANDO */
                VALINT(PROXINT+2) = ENDEXP(N) + J ;
                /* NUMERO DO REGISTRADOR BASE DO OPERANDO */
                VALINT(PROXINT+3) = RBASE(N) ;
                PROXINT = PROXINT + 4 ;
              END ;
             END ;
            END ;
      END ;
 IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
      ('PLAPP27 (RECONHECEDOR DE COMANDO "WRITE") RETORNOU',
             ' COM TIPO =' , TIPO) (A , A , F(3));
END PLAPP27 ;
```

```
/* PLAPP28 - RECONHECEDOR DE SECAO DE PARAMETROS REAIS.          */
PLAPP28 : PROC (PT,ENDRESUL) RECURSIVE ;
/*****************************************************************/
/*                                                               */
/*  ESTA PROCEDURE RECONHECE SECAO DE PARAMETROS REAIS DE        */
/*  PROCEDURES E FUNCOES.                                        */
/*  E' CHAMADA COM O PRIMEIRO SIMBOLO APOS O NOME DA PROCEDU-    */
/*  RE OU FUNCAO E RETORNA NO SIMBOLO SEGUINTE AO ")" QUE        */
/*  ENCERRA A SECAO DE PARAMETROS REAIS.                         */
/*  RECEBE COMO PARAMETRO O PONTEIRO PARA O NOME DA PROCEDU-     */
/*  RE OU FUNCAO NA TABELA DE SIMBOLOS ("PT") E DEVOLVE EM       */
/*  "ENDRESUL" O ENDERECO DO TEMPORARIO QUE CONTERA' O RESUL-    */
/*  TADO, NO CASO DE FUNCAO (TEMPORARIO CRIADO NA GERACAO DE     */
/*  CODIGOS).                                                    */
/*                                                               */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MANIPULACAO DE TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECONHECEDOR DE EXPRESSOES */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCOES */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL, TABALFA, ENDZERO, */
  /* ENDUM, EFALSE, ETRUE, ENDCTE E POSCTE    */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABPARM E PRXPARM */
$CONTINUE WITH PLAMV09 RETURN
1 /* MACRO TIPEXP */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL PT          BIN FIXED(15,0) ;
 DCL ENDRESUL    BIN FIXED(15,0) ;
 DCL NPARAM      BIN FIXED(15,0) AUTOMATIC ;
 DCL NPFORM      BIN FIXED(15,0) AUTOMATIC ;
 DCL TIPIDEN     BIN FIXED(15,0) AUTOMATIC ;
 DCL PCNT        BIN FIXED(15,0) AUTOMATIC ;
 DCL TOPO        BIN FIXED(15,0) AUTOMATIC ;
```

```
DCL 1 PILHA (16) AUTOMATIC ,
      2 END            BIN FIXED(15,0) ,
      2 BASE           BIN FIXED(15,0) ;
DCL LIMTOPO   BIN FIXED(15,0) STATIC INIT (16) ;
DCL FIM       BIT(1) AUTOMATIC ;
DCL CALL30    BIT(1) AUTOMATIC ;
DCL OK        BIT(1) STATIC ;
DCL END1      BIN FIXED(15,0) STATIC ;
DCL RBASE     BIN FIXED(15,0) STATIC ;
DCL TIPB      BIN FIXED(15,0) STATIC ;
DCL RBASE1    BIN FIXED(15,0) STATIC ;
DCL EMPIL     BIT(1) STATIC ;
DCL AUX       BIN FIXED(15,0) STATIC ;
DCL MOD       BUILTIN ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
     ('PLAPP28 (RECONHECEDOR DE SECAO DE PARAMETROS REAIS) ' ,
      'INICIOU COM PT = ' , PT)( A , A , F(3) ) ;
NPARAM = 0 ; /* ZERA NUMERO DE PARAMETROS REAIS */
TOPO = 0 ;
/* OBTEM NUMERO DE PARAMETROS FORMAIS */
IF PTCTE(PT) = 0 | IDTIPO(PT) = -1
THEN NPFORM = 0 ; /* PARAMETRO PROCEDURE OU PARAMETRO  */
                  /* FUNCAO OU IDENTIFICADOR NAO DECLARADO */
ELSE DO ;
       PONT = PTCTE(PT) + 1 ;
       NPFORM = TABPARM(PONT) ; /* NUM.DE PARAMETROS FORMAIS */
       PONT = PONT + 1 ; /* POSICIONA NO CODIGO DO PRIM.PARAM. */
     END ;
IF TIPO = 11 | NPFORM ¬= 0
THEN DO ; /* PEGOU "(" OU PROC OU FUNCAO COM PARAMETROS */
      IF TIPO = 11
      THEN CALL PLAPP04 ; /* REC. LEXICO */
      ELSE DO ;
             ERRO = 92 ; /* FALTA "(". INSERIDO */
             COLUNA = PTR - 1 ;
             CALL PLAPP01 ; /* IMPRIME MSG ERRO */
           END ;
      /* RECONHECE LISTA DE PARAMETROS */
      FIM = '0'B ;
      DO WHILE (¬FIM) ;
       /* PRIMEIRA VEZ OU ENQUANTO NAO FOR ";",")","END" NEM EOF*/
       IF TIPO = 9
       THEN DO ; /* VIRGULA */
              ERRO = 112 ; /* FALTA PARAMETRO */
              COLUNA = PTR - 1 ;
              CALL PLAPP01 ; /* IMPRIME MSG ERRO */
              CALL PLAPP04 ; /* REC. LEXICO */
            END ;
       ELSE DO ;
              CALL30 = '1'B ;
              IF TIPO = 1
              THEN DO ; /* IDENTIFICADOR */
                     IDNOME = IDENT ;
                     CALL PLAPP05 ; /* BUSCA NA TABELA DE SIMBOLOS*/
                     IF ¬ACHOU
                     THEN DO ;
                            IDTIPO(IND) = -1 ;/*IDENT.NAO DECLARADO*/
                            TIPIDEN = -1 ;
                          END ;
```

```
         ELSE TIPIDEN = IDTIPO(IND) ;
         IF TIPIDEN = 10 | TIPIDEN >= 20 & TIPIDEN <= 24
           | TIPIDEN = 60 | TIPIDEN >= 70 & TIPIDEN <= 74
         THEN  /* PAR.REAL E' IDENT.DE PROC OU FUNCAO */
               /* OU E' PAR. PROC OU FUNCAO           */
            CALL VERIFICA ; /* TESTA COMPATIBILIDADE  */
                 /* ENTRE PARAM.REAL E O FORMAL COR- */
                 /* RESPONDENTE, SE EXISTIR.         */
      END ;
IF CALL30
THEN DO ;
       CALL PLAPP30 ; /* REC. EXPRESSAO */
       TIPIDEN = TIPEXP ;
      END ;
NPARAM = NPARAM + 1 ; /* INCREMENTA NUM.PAR.REAIS */
IF NPFORM >= NPARAM
THEN DO ;
       IF TIPIDEN >= 0
       THEN DO ;
            AUX = TABPARM(PONT) ;
            IF TIPIDEN <= 4
            THEN OK = AUX = 80 | AUX = 90 |
                      (AUX >= 80 & AUX <= 94) &
                      (TIPIDEN = 0 | TIPIDEN +80 = AUX
                      | TIPIDEN + 90 = AUX) ;
            ELSE OK = AUX=100 | AUX=110 | AUX >= 100 &
                      (TIPIDEN = 100 | AUX = TIPIDEN |
                      AUX = TIPIDEN + 10) ;
            IF ¬OK
            THEN DO ; /* PAR.REAL INCOMPATIVEL COM */
                   ERRO = 111 ; /* PAR.FORMAL      */
                   COLUNA = PTR - 1 ;
                   CALL PLAPP01 ; /* IMP.MSG ERRO */
                 END ;
          END ;
       /* POSICIONA PONT NO PROXIMO PARAM.FORMAL.   */
       /* SE O PARAM.FORMAL RECEM ANALISADO ERA     */
       /* ARRAY, DEVEM SER PULADOS TAMBEM O NUMERO  */
       /* DE DIMENSOES, O NUMERO DE ELEMENTOS E     */
       /* SUAS DIMENSOES EM TABPARM.                */
       IF AUX < 100
       THEN PONT = PONT + 1 ;
       ELSE PONT = PONT + 2 * TABPARM(PONT+1) + 3 ;
      END ;
IF TIPIDEN >= 0
THEN DO ; /* PARAM. NAO E' PROC NEM FUNCAO */
       IF TOPO = LIMTOPO
       THEN DO ; /* PLAPP28 - RESTRICAO DE IMPLE-  */
            /* MENTACAO - NUMERO MAXIMO PERMITIDO */
            /* DE ELEMENTOS NA SECAO DE PARAM.    */
            /* REAIS FOI ULTRAPASSADO.            */
            ERRO = 130 ;
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
            ERRO = 41 ; /* COMPILACAO ABORTADA */
            CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
            STOP ;
           END ;
       TOPO = TOPO + 1 ;
       IF GERA
```

```
THEN DO ;
        IF (TUOPND = IVI | TUOPND = RVI |
            TUOPND = BVI | TUOPND = CVI) &
            EUOPND = 0
        THEN DO ; /* ELEM. DE PARAM. VAR. INDE- */
              /* XADA OU PARM. VAR. SIMPLES  OU */
              /* OPERANDO MATRICIAL PARAMETRO   */
              END(TOPO) = XUOPND ;
              BASE(TOPO) = BXUOPND ;
             END ;
        ELSE DO ;
              IF TUOPND = IVI | TUOPND = RVI |
                 TUOPND = BVI | TUOPND = CVI
              THEN DO ; /* ELEM. DE ARRAY (OU   */
                    /* DE PARAM. ARRAY POR VA-  */
                    /* LOR) C/INDICES VARIAVEIS */
                    /* GERA CTE C/ END. DO ARRAY*/
                    CTEINT = EUOPND ;
                    CALL PLAPP13 ; /* INSERE NA */
                         /* TAB CTES INTEIRAS   */
                    /* GERA INST.P/SOMAR END.DO */
                    /* ARRAY C/DESLOCAMENTO DO  */
                    /* ELEMENTO.                */
                    CALL PLAPP34(COZEREG,0,1,0,
                                 1,0,1) ;
                    CALL PLAPP34(COADI,0,ENDCTE,
                         BXUOPND,XUOPND,BXUOPND,
                         XUOPND) ;
                    TIPB = TIPEXP ;
                    RBASE = BUOPND ;
                    END1 = XUOPND ;
                    RBASE1 = BXUOPND ;
                   END ;
              ELSE DO ;
                    IF TUOPND=ICT | TUOPND=RCT |
                       TUOPND=BCT | TUOPND=CCT
                    THEN DO ; /* CONSTANTE */
                          IF TIPEXP = 1
                          THEN DO ;
                                TEMINT=TEMINT-1;
                                END1 = TEMINT ;
                               END ;
                          ELSE IF TIPEXP = 2
                               THEN DO ;
                                     TEMREAL =
                                       TEMREAL-1;
                                     END1=TEMREAL;
                                    END ;
                               ELSE IF TIPEXP=3
                                    THEN DO ;
                                          TEMBOOL=
                                          TEMBOOL-1;
                                          END1 =
                                           TEMBOOL;
                                         END ;
                                    ELSE DO ;
                                          TEMCHAR=
                                          TEMCHAR-1;
                                          END1 =
                                           TEMCHAR ;
```

```
                                                    END ;
                                        /*GERA INST. P/MOVER CTE*/
                                        /* PARA TEMPORARIO      */
                                        CALL PLAPP34(COZEREG,0,0,
                                                     0,1,0,1) ;
                                        BUOPND = BLOCO + 1 ;
                                        CALL PLAPP34(COATRIB,0,
                                             TIPEXP,0,EUOPND,
                                             BUOPND,END1);
                                        EUOPND = END1 ;
                                       END ;
                                  /* GERA CTE COM ENDERECO   */
                                  /* DO PARAMETRO.           */
                                  CTEINT = EUOPND ;
                                  CALL PLAPP13 ; /* INSERE NA */
                                       /* TAB CTES INTEIRAS */
                                  IF EUOPND < 0
                                  THEN TIPB = TIPEXP+4; /* BASE */
                                             /* RELATIVA A TEMP */
                                  ELSE TIPB = TIPEXP ; /* BASE  */
                                             /* RELATIVA A VAR  */
                                  RBASE = BUOPND ;
                                  END1 = ENDCTE ;
                                  RBASE1 = 0 ;
                                 END ;
                            /* GERA INST.P/SOMAR END.DO PARAM. */
                            /* COM VALOR DO REG. BASE          */
                            TEMINT = TEMINT - 1 ;
                            CALL PLAPP34(COSTOB,0,TIPB,0,RBASE,
                                         BLOCO+1,TEMINT) ;
                            CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                            CALL PLAPP34(COADI,RBASE1,END1,
                                 BLOCO+1,TEMINT,BLOCO+1,TEMINT);
                            END(TOPO) = TEMINT ;
                            BASE(TOPO) = BLOCO + 1 ;
                           END ;
                      END ;
               END ;
          IF TIPO = 9 /* VIRGULA */
          THEN CALL PLAPP04 ; /* REC. LEXICO */
          ELSE IF TIPO = 12 /* ")" */
               THEN FIM = '1'B ;
               ELSE DO ;
                      ERRO = 102 ; /* SIMBOLO INVALIDO */
                           /* ESPERADO "," OU ")".     */
                      COLUNA = PTR - 1 ;
                      CALL PLAPP01 ; /* IMP. MSG ERRO */
                      IF TIPO = 10 | TIPO = 67 | TIPO = 0
                      THEN FIM = '1'B ;
                      ELSE DO ;
                             ERRO = 117 ; /* INSERIDA "," */
                             CALL PLAPP01 ; /* IMP.MSG ERRO */
                            END ;
                    END ;
        END ;
    END ;
    IF TIPO = 12 /* ")" */
    THEN CALL PLAPP04 ; /* REC. LEXICO */
    ELSE DO ;
         ERRO = 103 ; /* FALTA ")". INSERIDO */
```

```
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MSG ERRO */
          END ;
      IF NPARAM ¬= NPFORM & IDTIPO(PT) ¬= 60 &
         (IDTIPO(PT) < 70 | IDTIPO(PT) > 74)
      THEN DO ; /* NUM.PARAM.REAIS ¬= NUM.PARAM.FORMAIS */
            ERRO = 93 ;
            COLUNA = PTR - 1 ;
            CALL PLAPP01 ; /* IMPRIME MSG ERRO */
          END ;
    END ;
IF GERA
THEN DO ; /* CHAMADA P/ PROCEDURE OU FUNCAO */
      EMPIL = '0'B ;
      IF IDTIPO(PT) = 10 | IDTIPO(PT) = 60 /* PROC */
      THEN END1 = 1 ;
      ELSE DO ;
            END1 = 2 ;
            TIPB = MOD(IDTIPO(PT),10)+4 ; /* TIPO DA FUNCAO + 4 */
            IF TIPB = 5
            THEN DO ; /* FUNCAO INTEIRA */
                  TEMINT = TEMINT - 1 ;
                  ENDRESUL = TEMINT ;
                 END ;
            ELSE IF TIPB = 6
                 THEN DO ; /* FUNCAO REAL */
                       TEMREAL = TEMREAL - 1 ;
                       ENDRESUL = TEMREAL ;
                      END ;
                 ELSE IF TIPB = 7
                      THEN DO ; /* FUNCAO BOOLEANA */
                            TEMBOOL = TEMBOOL - 1 ;
                            ENDRESUL = TEMBOOL ;
                           END ;
                      ELSE DO ; /* FUNCAO CHAR */
                            TEMCHAR = TEMCHAR - 1 ;
                            ENDRESUL = TEMCHAR ;
                           END ;
          END ;
      IF IDTIPO(PT) <= 24
      THEN DO ; /* CHAMADA P/PROC OU FUNCAO (NAO PARAMETRO) */
            /* GERA INST.P/COLOCAR NA AREA DE LIGACAO O END. */
            /* ABSOLUTO DA LISTA DE PARAMETROS               */
            AUX = TABPARM(PTCTE(PT)) ; /* END. AREA DE LIGACAO */
            CALL PLAPP34(COSTOB,0,5,0,BLOCO+1,0,AUX) ;
            TEMINT = TEMINT-END1-TOPO; /* END. LISTA DE PARAM. */
            CTEINT = TEMINT ;
            CALL PLAPP13 ; /* INSERE TAB CTES INTEIRAS */
            CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
            CALL PLAPP34(COADI,0,AUX,0,ENDCTE,0,AUX) ;
           END ;
      ELSE DO ; /* CHAMADA DE PROC OU FUNCAO (PARAMETRO) */
            /* GERA INST.P/COLOCAR NA AREA DE LIGACAO O END. */
            /* ABSOLUTO DA LISTA DE PARAMETROS.              */
            TEMINT = TEMINT - 1 ;
            CALL PLAPP34(COSTOB,0,5,0,BLOCO+1,BLOCO+1,TEMINT) ;
            CTEINT = TEMINT-END1-TOPO; /* END.DA LISTA DE PARAM.*/
            CALL PLAPP13 ; /* INSERE NA TEB. CTES INTEIRAS */
            CALL PLAPP34(COSETM,0,ENDZERO,0,ENDZERO,
                         NIVEL(PT)+1,ENDER(PT)-1) ;
```

```
            CALL PLAPP34(COADI,BLOCO+1,TEMINT,0,ENDCTE,0,0) ;
            TEMINT = TEMINT-END1-TOPO ;
          END ;
      IF END1 = 2
      THEN DO ; /* FUNCAO */
            /* GERA INST.P/COLOCAR END.ABSOLUTO DO TEMPORARIO */
            /* QUE CONTERA' O RESULTADO DA FUNCAO NA LISTA DE */
            /* PARAMETROS.                                    */
            CALL PLAPP34(COSTOB,0,TIPB,0,BLOCO+1,BLOCO+1,TEMINT+1);
            CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
            CTEINT = ENDRESUL ; /* END. DO TEMPORARIO */
            CALL PLAPP13 ; /* INSERE NA TAB CTES INTEIRAS */
            CALL PLAPP34(COADI,0,ENDCTE,BLOCO+1,TEMINT+1,
                         BLOCO+1,TEMINT+1) ;
          END ;
      END1 = TEMINT + END1 - 1 ;
      RBASE1 = BLOCO + 1 ;
      CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
      DO AUX = 1 TO TOPO ;
       CALL PLAPP34(COATRIB,0,1,BASE(AUX),END(AUX),RBASE1,
                    END1+AUX) ;
      END ;
      /* COMPLETA LISTA DE PARAMETROS (END.DE RETORNO) E GERA */
      /* INSTRUCAO DE DESVIO P/ PROC OU FUNCAO                */
      IF IDTIPO(PT) <= 24
      THEN DO ; /* PROC OU FUNCAO (NAO PARAMETRO) */
            IF NIVEL(PT) < BLOCO
            THEN DO ; /* GERA INSTRUCOES P/ EMPILHAR BASES */
                  EMPIL = '1'B ;
                  CTEINT = CINST + LC1 + 2 ; /* END.RETORNO-1 */
                 END ;
            ELSE CTEINT = CINST + LC1 + 1; /* END.DE RETORNO - 1 */
            CALL PLAPP13 ; /* INSERE NA TABELA CTES INTEIRAS */
            CALL PLAPP34(COATRIB,0,1,0,ENDCTE,RBASE1,TEMINT) ;
            IF EMPIL
            THEN DO ;
                  CTEINT = NIVEL(PT) + 1 ; /* NIVEL DA PROC OU */
                                           /* FUNCAO CHAMADA   */
                  CALL PLAPP13 ; /* INSERE NA TAB CTE INTEIRAS */
                  CALL PLAPP34(COEMPB,0,0,0,0,0,ENDCTE) ;
                 END ;
            CALL PLAPP34(CODESV,0,0,0,CCQQR,0,ENDER(PT)) ;
           END ;
      ELSE DO ; /* PROC OU FUNCAO (PARAMETRO) */
            CTEINT = CINST + LC1 + 3 ; /* END. DE RETORNO - 1 */
            CALL PLAPP13 ; /* INSERE NA TAB CTES INTEIRAS */
            CALL PLAPP34(COATRIB,0,1,0,ENDCTE,RBASE1,TEMINT) ;
            CALL PLAPP34(COSET,0,0,0,3,NIVEL(PT)+1,ENDER(PT)) ;
            EMPIL = '1'B ; /* GERA INSTRUCAO P/EMPILHAR BASES */
            CALL PLAPP34(COEMPB,0,0,0,0,NIVEL(PT)+1,ENDER(PT)-2) ;
            CALL PLAPP34(CODESV,0,1,0,CCQQR,0,1) ;
           END ;
      IF EMPIL
      THEN /* GERA INST. P/DESEMPILHAR BASES */
           CALL PLAPP34(CODESB,0,0,0,0,0,0) ;
     END ;
1VERIFICA : PROC ;
 DCL PT1          BIN FIXED(15,0) STATIC ;
 DCL N            BIN FIXED(15,0) STATIC ;
 DCL NOTERRO      BIT(1) STATIC ;
```

```
DCL AUX1          BIN FIXED(15,0) STATIC ;
 IF NPFORM > NPARAM
 THEN DO ; /* EXISTE O PARAM. FORMAL */
        AUX = TABPARM(PONT) ;
        IF ((TIPIDEN = 10 | TIPIDEN = 60 ) & AUX = 60) |
           AUX = 70 & (TIPIDEN >= 20 & TIPIDEN <= 24 |
           TIPIDEN >= 70 & TIPIDEN <= 74 ) |
           (TIPIDEN = 20 | TIPIDEN = 70) & AUX >= 70 & AUX <= 74 |
           AUX > 70 & AUX <= 74 &
           (TIPIDEN = AUX | TIPIDEN + 50 = AUX)
        THEN DO ; /* PARAM. REAL DE MESMO TIPO QUE PARAM.FORMAL */
               CALL30 = '0'B ;
               /* TESTA SE PARAM. DA PROC OU FUNCAO QUE ESTA' */
               /* SENDO PASSADA COMO PARAMETRO SAO TODOS PA-  */
               /* RAMETROS POR VALOR.                         */
               /* PERCORRE TABPARM(PTCTE(IND)+1)              */
               IF PTCTE(IND) ¬= 0
               THEN DO ;
                      PT1 = PTCTE(IND) + 1 ;
                      N = TABPARM(PT1) ;
                      PT1 = PT1 + 1 ;
                      NOTERRO = '1'B ;
                      DO N = 1 TO N WHILE (NOTERRO) ;
                         AUX1 = TABPARM(PT1) ;
                         NOTERRO = AUX1 >=90 & AUX1 <= 94 |
                                   AUX1 >= 110 & AUX1 <= 114 ;
                         IF AUX1 >= 110
                         THEN PT1 = PT1 + 2 * TABPARM(PT1+1) + 3 ;
                         ELSE PT1 = PT1 + 1 ;
                      END ;
                      IF ¬NOTERRO
                      THEN DO ; /* PROC OU FUNCAO PASSADOS COMO   */
                             /* PARAMETRO SO' PODEM TER PARAMETROS */
                             /* POR VALOR.                         */
                             ERRO = 126 ;
                             COLUNA = PTR - 1 ;
                             CALL PLAPP01 ; /* IMPRIME MSG DE ERRO */
                           END ;
                      IF NIVEL(IND) > NIVEL(PT)
                      THEN DO ;
                             /* PARAMETRO PROC OU FUNCAO NAO PODE */
                             /* TER NIVEL MAIOR QUE O DA PROC OU  */
                             /* FUNCAO CHAMADA.                   */
                             ERRO = 148 ;
                             COLUNA = PTR - 1 ;
                             CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                           END ;
                    END ;
               ELSE DO ;
                      /* PARAMETRO REAL NAO PODE SER PARAMETRO */
                      /* PROCEDURE NEM PARAMETRO FUNCAO.       */
                      ERRO = 149 ;
                      COLUNA = PTR - 1 ;
                      CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                    END ;
               TIPIDEN = -2 ; /* P/NAO TESTAR DE NOVO  */
                              /* COMPATIBILIDADE       */
               CALL PLAPP04 ; /* REC. LEXICO */
               IF TIPO = 11
               THEN DO ; /* "(" */
```

```
                    CALL PLAPP28(IND,ENDRESUL) ;
                    /*PARAM. REAL INCOMPATIVEL C/PARAM.FORMAL */
                    ERRO = 111 ;
                    COLUNA = PTR - 1 ;
                    CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                  END ;
              IF GERA
              THEN DO ;
                    IF TOPO + 3 > LIMTOPO
                    THEN DO ; /* PLAPP28 - RESTRICAO DE IMPLE- */
                          /* MENTACAO - NUMERO MAXIMO PERMITI- */
                          /* DO DE ELEMENTOS NA SECAO DE PARAM.*/
                          /* REAIS FOI ULTRAPASSADO.          */
                          ERRO = 130 ;
                          COLUNA = PTR - 1 ;
                          CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                          ERRO = 41 ; /* COMPILACAO ABORTADA */
                          CALL PLAPP01 ; /* IMPRIME MSG ERRO */
                          STOP ;
                         END ;
                    TOPO = TOPO + 1 ;
                    /* GERA CTE C/END. PROC OU FUNCAO - 1 */
                    CTEINT = ENDER(IND) - 1 ;
                    CALL PLAPP13 ; /* INSERE TAB CTES INT.*/
                    END(TOPO) = ENDCTE ;
                    BASE(TOPO) = 0 ;
                    /* GERA CTE C/END LIGACAO */
                    CTEINT = TABPARM(PTCTE(IND)) ;
                    CALL PLAPP13 ; /* INSERE TAB CTES INT.*/
                    TOPO = TOPO + 1 ;
                    END(TOPO) = ENDCTE ;
                    BASE(TOPO) = 0 ;
                    /* GERA CTE C/ NIVEL DA PROC OU FUNCAO */
                    CTEINT = NIVEL(IND) + 1 ;
                    CALL PLAPP13 ; /* INSERE NA TABELA DE CTE */
                                   /* INTEIRAS.               */
                    TOPO = TOPO + 1 ;
                    END(TOPO) = ENDCTE ;
                    BASE(TOPO) = 0 ;
                  END ;
            END ;
      END ;
END VERIFICA ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
     ('PLAPP28 (RECONHECEDOR DE SECAO DE PARAMETROS REAIS)' ,
           ' RETORNOU COM TIPO =' , TIPO) (A,A,F(3)) ;
END PLAPP28 ;
```

```
/* PLAPP29 - RECONHECEDOR DE INDICES DE ARRAYS                    */
PLAPP29 : PROC (PTI,ENDIND,TIPEND,BASE) RECURSIVE ;
/*****************************************************************/
/*                                                               */
/*  ESTA PROCEDURE RECONHECE INDICES DE ARRAYS.                  */
/*  E' CHAMADA NO ABRE COLCHETES QUE SEGUE O NOME DO ARRAY E     */
/*  RETORNA NO SIMBOLO SEGUINTE AO FECHA COLCHETES QUE ENCER-    */
/*  RA OS INDICES DO ARRAY.                                      */
/*  RECEBE COMO PARAMETRO A POSICAO DO NOME DO ARRAY NA TABELA   */
/*  DE SIMBOLOS ("PTI") E DEVOLVE O ENDERECO DO ELEMENTO, O TI-  */
/*  PO DESTE ENDERECO E O NUMERO DO REGISTRADOR BASE A SER USA-  */
/*  DO ("ENDIND", "TIPEND" E "BASE").                            */
/*  SE TODOS OS OPERANDOS DA LISTA DE INDICES FOREM CONSTANTES   */
/*  (OU IDENTIFICADORES DE CONSTANTES) E O ARRAY NAO FOR PARA-    */
/*  METRO POR ENDERECO, "ENDIND" CONTERA' O VALOR DO ENDERECO    */
/*  BASE DO ARRAY, SOMADO COM O DESLOCAMENTO CALCULADO (EM FA-   */
/*  SE DE COMPILACAO) PARA O ELEMENTO, "TIPEND" = 2 (ENDERECO    */
/*  DIRETO) E "BASE" = NUMERO DO REGISTRADOR BASE RELATIVO AO    */
/*  ARRAY. CASO OS INDICES NAO SEJAM CONSTANTES E O ARRAY TAM-   */
/*  BEM NAO SEJA PARAMETRO POR ENDERECO, "ENDIND" CONTERA' O     */
/*  ENDERECO DO TEMPORARIO COM O DESLOCAMENTO CALCULADO EM FA-   */
/*  SE DE EXECUCAO E "TIPEND" = 3 (ENDERECO INDIRETO) E          */
/*  "BASE" = NUMERO DO REGISTRADOR BASE RELATIVO A ESTE TEMPO-   */
/*  RARIO. NO CASO DE ARRAY PARAMETRO POR ENDERECO, "ENDIND"     */
/*  CONTERA' O ENDERECO DO TEMPORARIO COM O ENDERECO ABSOLUTO    */
/*  DO ELEMENTO (CALCULADO EM FASE DE EXECUCAO), "TIPEND" = 4    */
/*  (PARAMETRO POR ENDERECO) E "BASE" = NUMERO DO REGISTRADOR    */
/*  BASE RELATIVO A ESTE TEMPORARIO.                             */
/*                                                               */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO TABELAS DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* RECONHECEDOR DE EXPRESSOES */
$CONTINUE WITH PLAMP30 RETURN
1 /* GRAVA REGISTRO DE INSTRUCAO EM "OBJ1" */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT, TABREAL, TABALFA, ENDZERO, ENDUM, EFALSE,      */
  /* ETRUE, ENDCTE E POSCTE                                       */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM,MAXDIM E PROXDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO TIPEXP,PTMAT E TAMCHAR */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
```

```
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL    PT1          BIN FIXED(15,0) ;
 DCL    PT           BIN FIXED(15,0) ;
 DCL    ENDIND       BIN FIXED(15,0) ;
 DCL    TIPEND       BIN FIXED(15,0) ;
 DCL    BASE         BIN FIXED(15,0) ;
 DCL    INDEC        BIN FIXED(15,0) ;
 DCL    ENDLISTA1    BIN FIXED(15,0) ;
 DCL 1 LIST ,
       2 LISTAIND(64) ,
         3 LISTATIP    BIN FIXED(15,0) ,
         3 LISTAEND    BIN FIXED(15,0) ,
         3 LISTABAS    BIN FIXED(15,0) ,
       2 NUMIND      BIN FIXED(15,0) ;
 DCL LIMIND         BIN FIXED(15,0) STATIC INIT(64) ;
 DCL CTE            BIT(1) ;
 DCL AUX            BIN FIXED(31,0) STATIC ;
 DCL I              BIN FIXED(15,0) STATIC ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
          ('PLAPP29 (RECONHECEDOR DE INDICES DE ARRAYS) INICIOU ' ,
           'COM TIPO = ' , TIPO )(A,A,F(3)) ;
 PT = PT1 ; /* SALVA PONT. PARA TAB. SIMBOLOS */
 CALL PLAPP04 ; /* REC.LEXICO P/ PULAR O ABRE COLCHETES */
 IF IDTIPO(PT) ¬= -1 & PTCTE(PT) ¬= 0
 THEN INDEC = TABDIM (PTCTE(PT)) ; /* NUM.DE DIMENSOES DECLARADO */
 ELSE INDEC = 0 ; /* NUM.DE DIMENSOES INDEFINIDO */
 NUMIND = 0 ; /* ZERA NUMERO DE INDICES RECONHECIDO */
 CTE = '1'B ; /* PARA INDICAR INDICES CTES */
 IF TIPO = 15 | TIPO = 10 | TIPO = 67
 THEN DO ; /* SE E' FECHA COLCHETES OU ';' OU 'END' */
        ERRO = 57 ; /* FALTA INDICE DO ARRAY */
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ;
        IF TIPO = 15
        THEN CALL PLAPP04 ; /* PULA FECHA COLCHETES */
      END ;
 ELSE DO ;
        DO WHILE (TIPO¬=10 & TIPO¬=15 & TIPO¬=67 & TIPO¬=0);
          /* ENQUANTO NAO  FOR ';' NEM FECHA COLCHETES */
          /* NEM 'END' NEM EOF                         */
          IF TIPO = 9 /* PEGOU ',' */
          THEN DO ;
                 ERRO = 57 ; /* FALTA INDICE */
                 COLUNA = PTR - 1 ;
                 CALL PLAPP01 ;
                 CALL PLAPP04 ; /* PULA ',' */
               END ;
          ELSE DO ;
                 CALL PLAPP30 ; /* RECONHECEDOR DE EXPRESSAO */
                 IF TIPEXP > 1
                 THEN DO ; /* EXPRESSAO NAO E' INTEIRA */
                        ERRO = 110 ;/*INDICE NAO E' VALOR INTEIRO*/
                        COLUNA = PTR - 1 ;
                        CALL PLAPP01 ;
                      END ;
```

```
      NUMIND = NUMIND + 1; /*INCREMENTAR NUM.DE INDICES*/
      IF GERA
      THEN DO ;
            IF NUMIND > LIMIND
            THEN DO ;
                  /* PLAPP29 - RESTRICAO DE IMPLEMENT.*/
                  /* NUMERO MAXIMO PERMITIDO DE INDI- */
                  /* CES DE ARRAY FOI ULTRAPASSADO.   */
                  ERRO = 127 ;
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMP. MSG DE ERRO */
                  ERRO = 41 ; /* COMP. ABORTADA */
                  COLUNA = 1 ;
                  CALL PLAPP01 ; /* IMP. MSG ERRO */
                  STOP ;
                 END ;
            IF TUOPND = IVI
            THEN DO ; /* VARIAVEL COM END. INDIRETO */
                  /* GERA CTE COM END. DO OPERANDO */
                  CTEINT = EUOPND ;
                  CALL PLAPP13 ; /* INSERE TAB CTE INT */
                  /* GERA INST. ZEREG TODOS REGS. */
                  CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                  /* GERA INST. ADI - OPERANDO 1 = END */
                  /* CTE C/END DO OPERANDO, OPERANDO   */
                  /* 2 = END TEMP C/DESLOCAMENTO CALCU-*/
                  /* LADO P/ELEMENTO, OPERANDO 3 = END */
                  /* DO MESMO TEMPORARIO               */
                  CALL PLAPP34(COADI,0,ENDCTE,BXUOPND,
                              XUOPND,BXUOPND,XUOPND) ;
                  /* SOMA VALOR DO REG.BASE AO TEMPOR.*/
                  TEMINT = TEMINT - 1 ;
                  CALL PLAPP34(COSTO8,0,1,0,BUOPND,
                              BLOCO+1,TEMINT) ;
                  CALL PLAPP34(COADI,BLOCO+1,TEMINT,
                              BXUOPND,XUOPND,BXUOPND,
                              XUOPND) ;
                  LISTAEND(NUMIND) = XUOPND ;
                  LISTABAS(NUMIND) = BXUOPND ;
                 END ;
            ELSE DO ;
                  LISTAEND(NUMIND) = EUOPND ;
                  LISTABAS(NUMIND) = BUOPND ;
                 END ;
            LISTATIP(NUMIND) = TUOPND ;
            /* DESLIGA CTE SE INDICE NAO FOR CONSTANTE */
            CTE = CTE & TUOPND = ICT ;
           END ;
      IF TIPO ¬= 9 & TIPO ¬= 15
      THEN DO ; /* NAO E' ',' NEM FECHA COLCHETES */
            ERRO = 62 ; /* SIMBOLO INVALIDO. ',' OU */
            COLUNA = PTR - 1; /* FECHA COLCHETES ES- */
            CALL PLAPP01 ; /* PERADO. */
            DO WHILE (TIPO¬=9 & TIPO¬=15 & TIPO¬=10 &
                      TIPO¬=67 & TIPO¬=0) ;
              /* PROCURA ',' , FECHA COLCHETES , */
              /* ';' OU 'END' OU EOF             */
              CALL PLAPP04 ;
            END ;
            ERRO = 14 ; /* SIMB. ANTERIORES IGNORADOS */
```

```
                COLUNA = PTR - 1 ;
                CALL PLAPP01 ;
              END ;
          IF TIPO = 9   /* ',' */
          THEN DO ;
                CALL PLAPP04 ; /* REC. LEXICO */
                IF TIPO = 10 | TIPO = 15 | TIPO = 67
                THEN DO ; /* PEGOU ";", FECHA COLCHETES */
                          /* OU "END" DEPOIS DE ","     */
                     ERRO = 57 ; /* FALTA INDICE */
                     COLUNA = PTR - 1 ;
                     CALL PLAPP01 ; /* IMP. MSG ERRO */
                    END ;
              END ;
        END ;
END ;
IF TIPO ¬= 15
THEN DO ; /* NAO E' FECHA COLCHETES */
       ERRO = 58 ; /* FALTA FECHA COLCHETES. INSERIDO */
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ;
     END ;
ELSE CALL PLAPP04 ; /* PULA FECHA COLCHETES */
IF IDTIPO(PT) ¬= -1 & NUMIND ¬= INDEC
THEN DO ; /*NUM.DE DIMENSOES ¬= NUM.DE DIMENSOES DECLARADO*/
       ERRO = 113 ;
       COLUNA = PTR - 1 ;
       CALL PLAPP01 ; /* IMPRIME MSG ERRO */
     END ;
IF GERA
THEN DO ;
      /* ENDERECO DA LISTA DE CTES DO ARRAY */
      ENDLISTA1 = TABDIM(PTCTE(PT)+2)+1 ;
      IF ¬CTE
      THEN DO ; /* PELO MENOS 1 INDICE NAO CTE */
            /* GERA INSTRUCAO "CDE" - OPERANDO 1 = END DA */
            /* LISTA DE CTES P/CALCULO DE DESLOCAMENTO DO */
            /* ELEMENTO, OPERANDO 2 = END DA LISTA DE ENDS*/
            /* DOS INDICES, OPERANDO 3 = END TEMP QUE CON-*/
            /* TERA O DESLOCAMENTO CALCULADO              */
            TEMINT = TEMINT - 1 ;
            CALL PLAPP34(COCDE,0,ENDLISTA1,0,PROXINT,
                         BLOCO+1,TEMINT) ;
            IF PROXINT + NUMIND * 3 - 1 > LSKINT
            THEN DO ;
                  /* PLAPP29 - RESTRICAO DE IMPLEMENTACAO */
                  /* NUMERO MAXIMO DE CTES INTEIRAS NOS   */
                  /* BLOCOS ATIVOS FOI ULTRAPASSADO.      */
                  ERRO = 128 ;
                  COLUNA = PTR - 1 ;
                  CALL PLAPP01 ; /* IMP MSG ERRO */
                  ERRO = 41 ; /* COMPILACAO ABORTADA */
                  COLUNA = 1 ;
                  CALL PLAPP01 ; /* IMP MSG ERRO */
                  STOP ;
                 END ;
            DO I = 1 TO NUMIND ;
              VALINT(PROXINT) = LISTATIP(I) ;
              VALINT(PROXINT+1) = LISTAEND(I) ;
              VALINT(PROXINT+2) = LISTABAS(I) ;
```

```
                    PROXINT = PROXINT + 3 ;
                  END ;
                  ENDIND = TEMINT ;
                  BASE = BLOCO + 1 ;
                  IF IDTIPO(PT) < 100 | IDTIPO(PT) > 104
                  THEN TIPEND = 3 ; /* ENDERECO INDIRETO */
                  ELSE DO ;
                         /* SOMA END.ABSOLUTO DO ARRAY AO TEMP.*/
                         /* ZERA REGS.INDEXADORES */
                         CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                         CALL PLAPP34(COADI,NIVEL(PT)+1,ENDER(PT),
                                      BASE,TEMINT,BASE,TEMINT) ;
                         TIPEND = 4 ; /* PARAMETRO P/ENDERECO */
                       END ;
                END ;
           ELSE DO ; /* TODOS OS INDICES CTES */
                  /* CALCULA DESLOCAMENTO DO ELEMENTO */
                  ENDLISTA1 = ENDLISTA1 + NUMIND * 2 ;
                  AUX = VALINT(ENDLISTA1) +
                     VALINT(LISTAEND(NUMIND))-VALINT(ENDLISTA1-1);
                  DO I = NUMIND-1 TO 1 BY -1 ;
                    ENDLISTA1 = ENDLISTA1 - 2 ;
                    AUX = AUX + (VALINT(LISTAEND(I)) -
                         VALINT(ENDLISTA1-1)) * VALINT(ENDLISTA1);
                  END ;
                  IF IDTIPO(PT) < 100 | IDTIPO(PT) > 104
                  THEN DO ; /* NAO E' PARAM.ARRAY P/ENDERECO */
                        /* SOMA DESLOCAMENTO COM ENDERECO */
                        /* BASE DO ARRAY                  */
                        ENDIND = AUX + ENDER(PT) - 1 ;
                        TIPEND = 2 ; /* ENDERECO DIRETO */
                        BASE = NIVEL(PT) + 1 ; /* BASE DO ARRAY*/
                       END ;
                  ELSE DO ;
                         /* GERA INSTRUCOES PARA CALCULO DO */
                         /* ENDERECO ABSOLUTO DO ELEMENTO   */
                         CTEINT = AUX ; /* DESLOC.CALCULADO */
                         /* INSERE NA TAB.DE CTES INTEIRAS */
                         CALL PLAPP13 ;
                         TEMINT = TEMINT - 1 ;
                         /* ZERA INDEXADORES */
                         CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                         /* SOMA END.ABSOLUTO DO ARRAY COM */
                         /* DESLOCAMENTO DO ELEMENTO       */
                         CALL PLAPP34(COADI,NIVEL(PT)+1,ENDER(PT),
                                      0,ENDCTE,BLOCO+1,TEMINT) ;
                         ENDIND = TEMINT ;
                         TIPEND = 4 ; /* PARAMETRO POR ENDERECO*/
                         BASE = BLOCO + 1 ;
                       END ;
                END ;
           END ;
      END ;
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
        ('PLAPP29 (RECONHECEDOR DE INDICES DE ARRAYS) RETORNOU',
         ' COM TIPO =' , TIPO)(A , A , F(3)) ;
END PLAPP29 ;
```

```
/* PLAPP30 - RECONHECEDOR DE EXPRESSAO                              */
PLAPP30 : PROC RECURSIVE ;
/*******************************************************************/
/*                                                                 */
/* ESTA PROCEDURE RECONHECE EXPRESSAO.                             */
/* E' CHAMADA COM O PRIMEIRO ELEMENTO DA EXPRESSAO LIDO            */
/* E RETORNA COM O PRIMEIRO SIMBOLO ESTRANHO A MESMA,              */
/* QUE SERA' ANALISADO PELA ROTINA QUE CHAMOU ESTE RECO-           */
/* NHECEDOR.                                                       */
/* ESTA PROCEDURE COLOCA EM "TIPEXP" O TIPO DA EXPRESSAO           */
/* RECONHECIDA, CONFORME A TABELA ABAIXO:                          */
/*                                                                 */
/*    TIPEXP     EXPRESSAO                                         */
/*     -1        EXPRESSAO DE TIPO INDEFINIDO                      */
/*                                                                 */
/*      0        EXPRESSAO NAO MATRICIAL DE TIPO INDEFINIDO        */
/*      1        EXPRESSAO NAO MATRICIAL INTEIRA                   */
/*      2        EXPRESSAO NAO MATRICIAL REAL                      */
/*      3        EXPRESSAO NAO MATRICIAL BOOLEANA                  */
/*      4        EXPRESSAO NAO MATRICIAL CHAR                      */
/*                                                                 */
/*    100        EXPRESSAO MATRICIAL DE TIPO INDEFINIDO            */
/*    101        EXPRESSAO MATRICIAL INTEIRA                       */
/*    102        EXPRESSAO MATRICIAL REAL                          */
/*    103        EXPRESSAO MATRICIAL BOOLEANA                      */
/*    104        EXPRESSAO MATRICIAL CHAR                          */
/* SE A EXPRESSAO RECONHECIDA FOR MATRICIAL ("TIPEXP >=            */
/* 100), SERA' RETORNADO EM "PTMAT" O PONTEIRO PARA A TA-          */
/* BELA DE DIMENSOES, CORRESPONDENDO AS DIMENSOES DA MA-           */
/* TRIZ RESULTANTE, PARA SER USADO PELO COMANDO DE ATRI-           */
/* BUICAO (PARA VERIFICAR COERENCIA DAS MATRIZES).                 */
/* ALEM DISSO, SE A EXPRESSAO RESULTANTE FOR DO TIPO CHAR          */
/* ("TIPEXP" = 4 OU "TIPEXP" = 104), SERA' RETORNADO EM            */
/* "TAMCHAR" O NUMERO DE CARACTERES DOS ELEMENTOS DA EX-           */
/* PRESSAO, PARA SER USADO PELO RECONHECEDOR DE COMANDO            */
/* DE ATRIBUICAO (NO PLAPP17) PARA VERIFICAR COERENCIA E           */
/* NA GERACAO DE CODIGO PARA O COMANDO "WRITE".                    */
/*                                                                 */
/*******************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* RECONHECEDOR LEXICO */
$CONTINUE WITH PLAMP04 RETURN
1 /* MANIPULACAO DA TABELA DE SIMBOLOS */
$CONTINUE WITH PLAMP05 RETURN
1 /* MANIPULACAO DAS TAB.DE CONSTANTES */
$CONTINUE WITH PLAMP13 RETURN
1 /* REC. SECAO PARAMETROS REAIS */
$CONTINUE WITH PLAMP28 RETURN
1 /* REC. INDICES DE ARRAY */
$CONTINUE WITH PLAMP29 RETURN
1 /* GRAVA REG. INSTRUCAO EM OBJ1 */
$CONTINUE WITH PLAMP34 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TABSIMB */
$CONTINUE WITH PLAMV04 RETURN
1 /* MACRO TRACE */
```

```
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO TABINT,TABREAL,TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO TABDIM E PTDIM */
$CONTINUE WITH PLAMV08 RETURN
1 /* MACRO TIPEXP E PTMAT */
$CONTINUE WITH PLAMV10 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO 'OBJ1' */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL TOPO                   BIN FIXED(15,0) ;
 DCL OPER(64)               BIN FIXED(15,0) ;
 DCL PRIOR(64)              BIN FIXED(15,0) ;
 DCL OPAND(64)              BIN FIXED(15,0) ;
 DCL TIPOPAND(64)           BIN FIXED(15,0) ;
 DCL BOPAND(64)             BIN FIXED(15,0) ;
 DCL TIPEND(64)             BIN FIXED(15,0) ;
 DCL ENDINDEX(64)           BIN FIXED(15,0) ;
 DCL BINDEX(64)             BIN FIXED(15,0) ;
 DCL TCHAR(64)              BIN FIXED(15,0) ;
 DCL PRIDIN                 BIN FIXED(15,0) ;
 DCL BASE                   BIN FIXED(15,0) ;
 DCL DIRETO                 BIT(1) ;
 DCL AUXTIPO                BIN FIXED(15,0) STATIC ;
 DCL AUXOPER                BIN FIXED(15,0) STATIC ;
 DCL AUXOPAND               BIN FIXED(15,0) STATIC ;
 DCL AUXMUM                 BIN FIXED(15,0) STATIC ;
 DCL AUXIND                 BIN FIXED(15,0) STATIC ;
 DCL AUX                    BIN FIXED(15,0) STATIC ;
 DCL (A,B)                  BIN FIXED(31,0) STATIC ;
 DCL (AR,BR)                BIN FLOAT(21) STATIC ;
 DCL (AC,BC)                CHAR(1) STATIC ;
 DCL CODOP(16:25)           BIN FIXED(15,0) STATIC ;
 DCL CODBR(35:40)           BIN FIXED(15,0) STATIC ;
 DCL AUXERRO                BIT (1) ;
 DCL MOD                    BUILTIN ;
 DCL PILHA_CHEIA ENTRY RETURNS(BIT(1)) ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('PLAPP30 (RECONHECEDOR DE EXPRESSAO) INICIOU COM TIPO =',
       TIPO) (A , F(3)) ;
 CODOP(16) = COADR ;
 CODOP(17) = COSUBR ;
 CODOP(18) = CODIVR ;
 CODOP(19) = COMLTR ;
 CODOP(20) = COADI ;
 CODOP(21) = COSUBI ;
 CODOP(23) = COMLTI ;
 CODOP(24) = CODIVI ;
 CODOP(25) = COMOD ;
 CODBR(35) = CCEQ ;
 CODBR(36) = CCLE ;
 CODBR(37) = CCGE ;
```

```
CODBR(38) = CCNEQ ;
CODBR(39) = CCLT ;
CODBR(40) = CCGT ;
TOPO = 1 ;
OPER(1) = 26 ;
PRIOR(1) = 0 ;
BASE = 0 ;
CALL RECEXP ; /* RECONHECEDOR DE EXPRESSAO */
PRIDIN = 0 ; /* PARA FORCAR REDUCAO DA EXPRESSAO */
IF TRACE(3)
THEN DO ;
       AUX = TOPO ;
       CALL PRTPILHA ; /* IMPRIME PILHA */
     END ;
CALL REDUZ ; /* EFETUA REDUCAO DA EXPRESSAO */
IF TRACE(3)
THEN DO ;
       IF TOPO ¬= AUX
       THEN CALL PRTPILHA ; /* IMPRIME PILHA */
       ELSE DO ;
              PUT FILE(SPRINT) SKIP EDIT('NAO HOUVE REDUCAO')(A) ;
              PUT FILE(SPRINT) SKIP ;
            END ;
     END ;
TIPEXP = TIPOPAND(1) ; /* TIPO RESULTANTE DA EXPRESSAO */
PTMAT = TIPEND(1) ; /* TIPO DO ENDERECO DA EXPRESSAO OU   */
                    /* PONT. P/ TABDIM, SE EXP. MATRICIAL */
TAMCHAR = TCHAR(1) ; /* TAMANHO DO LITERAL, SE EXP. CHAR */
EUOPND = OPAND(1) ; /* ENDERECO DA EXPRESSAO */
BUOPND = BOPAND(1) ; /* NUM.DO REG. BASE DA EXPRESSAO */
XUOPND = ENDINDEX(1) ; /* ENDERECO USADO P/ INDEXACAO */
BXUOPND = BINDEX(1) ; /* NUM.DO REG.BASE DO ENDERECO P/INDEXACAO */
/* CALCULA TIPO DO OPERANDO */
IF TIPEXP = 1 /* EXP. INTEIRA */
THEN IF EUOPND < 0
     THEN TUOPND = ITE ; /* INTEIRO TEMPORARIO */
     ELSE IF PTMAT = 1
          THEN TUOPND = ICT ; /* INTEIRO CTE */
          ELSE IF PTMAT = 2
               THEN TUOPND = IVD ; /* INT. VAR. DIRETO */
               ELSE TUOPND = IVI ; /* INT. VAR. INDIRETO */
ELSE
   IF TIPEXP = 2 /* EXP REAL */
   THEN IF EUOPND < 0
        THEN TUOPND = RTE ; /* REAL TEMPORARIO */
        ELSE IF PTMAT = 1
             THEN TUOPND = RCT ; /* REAL CTE */
             ELSE IF PTMAT = 2
                  THEN TUOPND = RVD ; /* REAL VAR. DIRETO */
                  ELSE TUOPND = RVI ; /* REAL VAR. INDIRETO */
  ELSE IF TIPEXP = 3 /* EXP BOOLEANA */
       THEN IF EUOPND < 0
            THEN TUOPND = BTE ; /* BOOLEANO TEMPORARIO */
            ELSE IF PTMAT = 1
                 THEN TUOPND = BCT ; /* BOOL. CTE */
                 ELSE IF PTMAT = 2
                      THEN TUOPND = BVD ; /* BOOL. VAR. DIR.*/
                      ELSE TUOPND = BVI ; /* BOOL.VAR.INDIR.*/
      ELSE IF TIPEXP = 4 /* EXP CHAR */
           THEN IF PTMAT = 1
```

```
                THEN TUOPND = CCT ; /* CHAR CTE */
                ELSE IF PTMAT = 2
                     THEN TUOPND = CVD ; /* CHAR VAR.DIRETO*/
                     ELSE TUOPND = CVI ; /* CHAR VAR.IND. */
           ELSE IF TIPEXP = 51 /* EXP MAT. INTEIRA */
                THEN IF EUOPND < 0
                     THEN TUOPND = ITE ; /* INTEIRO TEMP. */
                     ELSE TUOPND = IVD ; /* INT VAR DIR */
                ELSE IF TIPEXP = 52 /* EXP MAT REAL */
                     THEN IF EUOPND < 0
                          THEN TUOPND = RTE ; /* REAL TEMP */
                          ELSE TUOPND = RVD ; /* REAL VAR DIR*/
                     ELSE IF TIPEXP = 53 /* EXP MAT BOOL */
                          THEN IF EUOPND < 0
                               THEN TUOPND = BTE ;/* BOOL TEMP */
                               ELSE TUOPND = BVD ;/* BOOL VAR DIR */
                          ELSE IF TIPEXP = 54 /* EXP MAT CHAR */
                               THEN TUOPND = CVD ;/* CHAR VAR DIR */
                               ELSE IF TIPEXP = 101 /*MAT(PARAM) INT*/
                                    THEN TUOPND = IVI ;/* INT INDIR */
                                    ELSE IF TIPEXP = 102 /* MAT */
                                                 /* (PARAM) REAL */
                                         THEN TUOPND = RVI ; /* REAL*/
                                                 /* INDIRETO */
                                    ELSE IF TIPEXP = 103 /* MAT */
                                                 /* (PARAM) BOOL */
                                         THEN TUOPND = BVI;/*BOOL*/
                                                 /* INDIRETO */
                                    ELSE IF TIPEXP = 104
                                           /*MAT (PARAM) CHAR*/
                                         THEN TUOPND = CVI ;
                                           /* CHAR INDIRETO */
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('PLAPP30 (RECONHECEDOR DE EXPRESSAO) RETORNOU COM TIPO =' ,
      TIPO , ' E TIPEXP =' , TIPEXP) (A , F(3) , A , F(3)) ;
1RECEXP : PROC RECURSIVE ;
 IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
     ('RECEXP (RECONHECEDOR DE EXPRESSAO) INICIOU COM TIPO =' ,
      TIPO)(A , F(3)) ;
CALL EXPSIMP ; /* RECONHECEDOR DE EXPRESSAO SIMPLES */
 IF TIPO >= 35 & TIPO <= 40
THEN DO ; /* < , > , = , >= , <= OU ¬= */
       PRIDIN = BASE + 1 ;
       IF TRACE(3)
       THEN DO ;
              AUX = TOPO ;
              CALL PRTPILHA ; /* IMPRIME PILHA */
            END ;
       CALL REDUZ ; /* EFETUA REDUCAO DA EXPRESSAO */
       IF TRACE(3)
       THEN IF TOPO ¬= AUX
            THEN CALL PRTPILHA ; /* IMPRIME PILHA */
            ELSE DO ;
                   PUT FILE(SPRINT) SKIP EDIT
                         ('NAO HOUVE REDUCAO')(A);
                   PUT FILE(SPRINT) SKIP ;
                 END ;
       IF PILHA_CHEIA
```

```
        THEN STOP ; /*COMPILACAO ABORTADA.RESTRICAO IMPLEMENTACAO*/
        TOPO = TOPO + 1 ;
        OPER(TOPO) = TIPO ;
        PRIOR(TOPO) = PRIDIN ;
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        CALL EXPSIMP ; /* RECONHECEDOR DE EXPRESSAO SIMPLES */
      END ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('RECEXP (RECONHECEDOR DE EXPRESSAO) RETORNOU COM TIPO =' ,
        TIPO)(A , F(3)) ;
EXPSIMP : PROC RECURSIVE ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('EXPSIMP (RECONHECEDOR DE EXPRESSAO SIMPLES) ' ,
       'INICIOU COM TIPO = ' , TIPO)(A,A,F(3)) ;
 IF TIPO = 20 | TIPO = 21
 THEN DO ; /* PEGOU '+' OU '-' UNARIO */
        IF TIPO = 21
        THEN DO ; /* '-' */
               IF PILHA_CHEIA
               THEN STOP ; /*COMPILACAO ABORTADA.RESTRICAO IMP.*/
               /* GERA OPERANDO CONSTANTE = 0 */
               OPAND(TOPO) = ENDZERO ; /* END.DA CTE ZERO */
               TIPEND(TOPO) = 1 ; /* P/INDICAR CTE */
               TIPOPAND(TOPO) = 1 ; /* INTEIRO NAO MATRICIAL */
               BOPAND(TOPO) = '0'B ; /*REG BASE = 0 */
               ENDINDEX(TOPO) = ENDZERO ; /* ENDERECO E' DIRETO */
               BINDEX(TOPO) = '0'B ; /* REG BASE = 0 */
               TOPO = TOPO + 1 ;
               OPER(TOPO) = TIPO ;
               PRIOR(TOPO) = BASE + 2 ;
             END ;
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
      END ;
 CALL RECTERM ; /* RECONHECEDOR DE TERMO */
 DO WHILE(TIPO = 20 | TIPO = 21 | TIPO = 31) ;
   /* ENQUANTO FOR '+' , '-' OU '|' */
   PRIDIN = BASE + 2 ;
   IF TRACE(3)
   THEN DO ;
          AUX = TOPO ;
          CALL PRTPILHA ; /* IMPRIME PILHA */
        END ;
   CALL REDUZ ; /* REDUZ EXPRESSAO SE NECESSARIO */
   IF TRACE(3)
   THEN IF TOPO ¬= AUX
        THEN CALL PRTPILHA ; /* IMPRIME PILHA */
        ELSE DO ;
               PUT FILE(SPRINT) SKIP EDIT('NAO HOUVE REDUCAO')(A) ;
               PUT FILE(SPRINT) SKIP ;
             END ;
   IF PILHA_CHEIA
   THEN STOP ; /* COMPILACAO ABORTADA.RESTRICAO IMPLEMENTACAO */
   TOPO = TOPO + 1 ;
   OPER(TOPO) = TIPO ;
   PRIOR(TOPO) = PRIDIN ;
   CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
   CALL RECTERM ; /* RECONHECEDOR DE TERMO */
 END ;
```

```
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('EXPSIMP (RECONHECEDOR DE EXPRESSAO SIMPLES) ' ,
       'RETORNOU COM TIPO = ' , TIPO)(A,A,F(3));
1RECTERM : PROC RECURSIVE ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('RECTERM (RECONHECEDOR DE TERMO) INICIOU COM ' ,
       'TIPO = ' , TIPO)(A , A , F(3)) ;
 CALL RECFATOR ; /* RECONHECEDOR DE FATOR */
 DO WHILE(TIPO >= 22 & TIPO <= 30) ;
   /* ENQUANTO FOR "/" , "*" , DIV , MOD OU "&" */
   PRIDIN = BASE + 3 ;
   IF TRACE(3)
   THEN DO ;
          AUX = TOPO ;
          CALL PRTPILHA ; /* IMPRIME PILHA */
        END ;
   CALL REDUZ ; /* REDUZ EXPRESSAO SE NECESSARIO */
   IF TRACE(3)
   THEN IF TOPO ¬= AUX
        THEN CALL PRTPILHA ; /* IMPRIME PILHA */
        ELSE DO ;
               PUT FILE(SPRINT) SKIP EDIT('NAO HOUVE REDUCAO')(A) ;
               PUT FILE(SPRINT) SKIP ;
             END ;
   IF PILHA_CHEIA
   THEN STOP ; /* COMPILACAO ABORTADA.RESTRICAO IMPLEMENTACAO */
   TOPO = TOPO + 1 ;
   OPER(TOPO) = TIPO ;
   PRIOR(TOPO) = PRIDIN ;
   CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
   CALL RECFATOR ; /* RECONHECEDOR DE FATOR */
 END ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('RECTERM (RECONHECEDOR DE TERMO) RETORNOU COM ' ,
       'TIPO = ' , TIPO)(A , A , F(3)) ;
1RECFATOR : PROC RECURSIVE ;
 DCL FIM                          BIT(1) ;
 DCL (TIPOIDEN , INDEND , TEND) BIN FIXED(15,0) ;
 DCL SALVA                        BIN FIXED(15,0) ;
 DCL RBASE                        BIN FIXED(15,0) ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('RECFATOR (RECONHECEDOR DE FATOR) INICIOU COM ' ,
       'TIPO = ' , TIPO)(A , A , F(3)) ;
 FIM = '0'B ;
 DIRETO = '1'B ;
 DO WHILE (¬FIM) ;
   FIM = '1'B ;
   TCHAR(TOPO) = 1 ; /* SE NAO FOR LITERAL O TAM. SERA' SEMPRE 1 */
   IF TIPO = 1
   THEN DO ; /* IDENTIFICADOR */
          IDNOME = IDENT ;
          CALL PLAPP05 ; /* BUSCA NA TABELA DE SIMBOLOS */
          IF ¬ACHOU
          THEN DO ;
                 IDTIPO(IND) = -1 ; /* IDENT. INDEFINIDO */
                 TIPOIDEN = -1 ;
```

```
      END ;
ELSE TIPOIDEN = IDTIPO(IND) ;
SALVA = IND ; /* GUARDA PONTEIRO P/ TABELA DE SIMBOLOS */
IF TIPOIDEN = -1 | TIPOIDEN = 10 | TIPOIDEN = 30 |
   TIPOIDEN = 60
THEN DO ; /*IDENT.NAO DECLARADO, DE PROCEDURE, LABEL  */
          /* OU PARAMETRO PROCEDURE                   */
        TIPOPAND(TOPO) = -1 ; /* OPERANDO INDEFINIDO */
        IF TIPOIDEN ¬= -1
        THEN DO ; /* LABEL, PROC OU PARAMETRO PROCEDURE */
               ERRO = 114 ; /* OPERANDO INVALIDO */
               COLUNA = PTR - 1 ;
               CALL PLAPP01 ; /* IMP. MSG. DE ERRO */
             END ;
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        IF TIPOIDEN ¬= 30 /* ¬= LABEL */
        THEN IF TIPO = 11 /* "(" ? */
             THEN CALL PLAPP28(SALVA,INDEND) ;
                  /* CHAMADO REC.SECAO PARAM.REAIS SO */
                  /* P/ANALISE PARCIAL LISTA PARAM.   */
             ELSE IF TIPOIDEN = -1 & TIPO = 14
                  /* IDENT.INDEFINIDO SEGUIDO DE ABRE */
                  /* COLCHETE ("#")                   */
                  THEN CALL PLAPP29(SALVA,INDEND,TEND,
                                    RBASE) ;
                  /* CHAMADO REC.INDICES ARRAY SO PARA*/
                  /* ANALISE PARCIAL                  */
     END ;
ELSE TIPOPAND(TOPO) = MOD(TIPOIDEN , 10) ;
     /* OPERANDO (INDEFINIDO, INTEIRO, REAL, BOOLEAN */
     /* OU CHAR) NAO MATRICIAL                       */
IF TIPOIDEN >= 50 & TIPOIDEN <= 54 |
   TIPOIDEN >= 100 & TIPOIDEN <= 104 |
   TIPOIDEN >= 110 & TIPOIDEN <= 114
THEN DO ; /* IDENT.DE ARRAY, DE PARAMETRO ARRAY OU */
          /* DE PARAMETRO ARRAY POR VALOR          */
        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
        IF TIPO = 14
        THEN DO ; /* ABRE COLCHETE (#) */
               /* CHAMA REC.INDICES DE ARRAY */
               CALL PLAPP29(IND,INDEND,TEND,RBASE) ;
               IF TEND = 2
               THEN DO ; /* ENDERECO DIRETO */
                      OPAND(TOPO) = INDEND ;/*END.ELEMENTO*/
                      BOPAND(TOPO) = RBASE ;/* BASE ELEM. */
                      TIPEND(TOPO) = 2 ; /* END.DIRETO */
                    END ;
               ELSE DO ; /* END.INDIR. OU PARAM.POR END.*/
                      TIPEND(TOPO) = 3 ; /*END.INDIRETO*/
                      DIRETO = '0'B ;
                      IF TEND = 3
                      THEN DO ; /* NAO E' PARAM.POR ENDER.*/
                             OPAND(TOPO) = ENDER(SALVA) ;
                                   /* ENDER.DO ARRAY */
                             BOPAND(TOPO) = NIVEL(SALVA) + 1;
                                   /* BASE DO ARRAY */
                           END ;
                      ELSE DO ; /* PARAM. POR ENDERECO */
                             OPAND(TOPO) = 0 ;
                             BOPAND(TOPO) = 0 ;
```

```
                    END ;
                ENDINDEX(TOPO) = INDEND ;
                BINDEX(TOPO) = RBASE ;
               END ;
          END ;
     ELSE DO ; /* OPERANDO MATRICIAL */
           IF TIPOIDEN >= 100 & TIPOIDEN <= 104
           THEN DO ; /* PARAM. POR ENDERECO */
                OPAND(TOPO) = 0 ;
                ENDINDEX(TOPO) = ENDER(SALVA) ; /* END.DO */
                                                /* PARAM. */
                BINDEX(TOPO) = NIVEL(SALVA) + 1 ; /* BASE */
                                                /* DO PARAM. */
                TIPOPAND(TOPO) = TIPOPAND(TOPO) + 100 ;
                                 /* PARAM.MAT.POR ENDERECO */
               END ;
           ELSE DO ;
                OPAND(TOPO) = ENDER(SALVA) ; /* END.ARRAY */
                BOPAND(TOPO) = NIVEL(SALVA) + 1 ; /* BASE */
                                                /* DO ARRAY */
                ENDINDEX(TOPO) = ENDZERO ;
                BINDEX(TOPO) = 0 ;
                TIPOPAND(TOPO) = TIPOPAND(TOPO) + 50;/*MAT*/
               END ;
           TIPEND(TOPO) = PTCTE(SALVA) ; /* PONT. P/ TAB. */
                                         /* DE DIMENSOES  */
           DIRETO = '0'B ;
          END ;
      END ;
ELSE IF TIPOIDEN >= 20 & TIPOIDEN <= 24 |
        TIPOIDEN >= 70 & TIPOIDEN <= 74
     THEN DO ; /* IDENTIFICADOR DE FUNCAO OU */
               /* PARAMETRO FUNCAO           */
          TIPEND(TOPO) = 2 ; /* ENDERECO DIRETO */
          CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
          /* CHAMA REC. SECAO DE PARAMETROS REAIS */
          CALL PLAPP28(SALVA,INDEND) ;
          OPAND(TOPO) = INDEND ; /* END.DO TEMPORA-*/
                                 /* RIO QUE CONTERA*/
                                 /* O RESULTADO DA */
                                 /* FUNCAO         */
          BOPAND(TOPO) = BLOCO + 1 ;
         END ;
     ELSE IF TIPOIDEN >= 0 & TIPOIDEN <= 4 |
             TIPOIDEN >= 40 & TIPOIDEN <= 44 |
             TIPOIDEN >= 90 & TIPOIDEN <= 94
          THEN DO ; /* IDENT.DE CTE, VAR.SIMPLES, OU */
                    /* PAR.VARIAVEL SIMPLES POR VALOR */
               IF TIPOIDEN <= 4
               THEN DO ;
                     TIPEND(TOPO) = 1 ; /* CTE */
                     BOPAND(TOPO) = '0'B ;/*REG.BASE=0*/
                     IF TIPOIDEN = 4 /* ID.CTE CHAR */
                     THEN TCHAR(TOPO)=TAMALFA(PTCTE(SALVA));
                    END ;
               ELSE DO ;
                     TIPEND(TOPO) = 2 ; /* DIRETO */
                     BOPAND(TOPO) = NIVEL(SALVA) + 1 ;
                    END ;
              OPAND(TOPO) = ENDER(SALVA) ;
```

```
                        CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                     END ;
                  ELSE IF TIPOIDEN >= 80 & TIPOIDEN <= 84
                     THEN DO ; /* PARAM.VAR. SIMPLES */
                           TIPEND(TOPO) = 3 ; /* INDIRETO */
                           OPAND(TOPO) = 0 ;
                           BOPAND(TOPO) = '0'B ;
                           ENDINDEX(TOPO) = ENDER(SALVA) ;
                                 /* ENDER.DO PARAMETRO */
                           BINDEX(TOPO) = NIVEL(SALVA) + 1 ;
                                 /* BASE DO PARAMETRO */
                           DIRETO = '0'B ;
                           CALL PLAPP04 ; /* REC. LEXICO */
                          END ;
      END ;
ELSE IF TIPO >= 2 & TIPO <= 4
      THEN DO ; /* NUM.INTEIRO, REAL OU LITERAL */
            TIPEND(TOPO) = 1 ; /* CONSTANTE */
            IF TIPO = 2
            THEN DO ; /* NUMERO INTEIRO */
                  CTEINT = NINT ;
                  CALL PLAPP13 ; /* INSERE TAB.CTE.INTEIRA */
                  TIPOPAND(TOPO) = 1 ; /* OPERANDO INTEIRO  */
                                       /* NAO MATRICIAL     */
                 END ;
            ELSE IF TIPO = 3
                 THEN DO ; /* NUMERO REAL */
                       CTEREAL = NREAL ;
                       CALL PLAPP14 ; /* INSERE TAB.CTE.REAL */
                       TIPOPAND(TOPO) = 2 ; /* OPERANDO REAL */
                                            /* NAO MATRICIAL */
                      END ;
                 ELSE DO ; /* LITERAL */
                       CTELIT = STRING ;
                       TAMLIT = TAM ;
                       CALL PLAPP15 ; /* INSERE TAB.CTE.ALFA */
                       TIPOPAND(TOPO) = 4 ; /* OPERANDO CHAR */
                                            /* NAO MATRICIAL */
                       TCHAR(TOPO) = TAM ; /* TAMANHO DO LIT.*/
                      END ;
            BOPAND(TOPO) = '0'B ; /* REG.BASE = 0 */
            OPAND(TOPO) = ENDCTE ;
            CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
           END ;
      ELSE IF TIPO = 32
           THEN DO ; /* "¬" */
                 FIM = '0'B ;
                 PRIDIN = BASE + 4 ;
                 IF PILHA_CHEIA
                 THEN STOP ; /* COMPILACAO ABORTADA */
                 TOPO = TOPO + 1 ;
                 OPER(TOPO) = TIPO ;
                 PRIOR(TOPO) = PRIDIN ;
                 CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                END ;
           ELSE IF TIPO = 11
                THEN DO ; /* "(" */
                      BASE = BASE + 5 ;
                      CALL PLAPP04 ; /* RECONHECEDOR LEXICO */
                      CALL RECEXP ; /* REC. DE EXPRESSAO */
```

```
                        IF TIPO ¬= 12
                        THEN DO ; /* ¬= ")" */
                                ERRO = 103 ; /* FALTA ")".INSERIDO*/
                                COLUNA = PTR - 1 ;
                                CALL PLAPP01 ; /* IMP. MSG. ERRO */
                             END ;
                          ELSE CALL PLAPP04 ; /* REC. LEXICO */
                          BASE = BASE - 5 ;
                       END ;
                  ELSE IF TIPO = 8 | TIPO = 13 | TIPO = 14
                       THEN DO ;/* ".", ":" OU ABRE COLCHETE (#) */
                              FIM = '0'B ;
                              ERRO = 115 ; /* SIMBOLO INVALIDO.   */
                                           /* SERA IGNORADO       */
                              COLUNA = PTR - 1 ;
                              CALL PLAPP01 ; /* IMP. MSG. ERRO */
                              CALL PLAPP04 ; /* REC. LEXICO */
                            END ;
                       ELSE DO ; /* OPERADOR, PALAVRA RESERVADA  */
                                 /* OU FIM VALIDO P/EXPRESSAO    */
                                 /* (",", ";", ")", FECHA COLCHE-*/
                                 /* TE ( ) OU ":=")              */
                              ERRO = 116 ; /* FALTA OPERANDO */
                              COLUNA = PTR - 1 ;
                              CALL PLAPP01 ; /* IMP. MSG. ERRO */
                              TIPOPAND(TOPO) = -1 ; /* OPERANDO */
                                                    /*INDEFINIDO*/
                            END ;
 END ;
 IF DIRETO
 THEN DO ;
       ENDINDEX(TOPO) = ENDZERO ; /* END.DA CTE ZERO, P/INDEXACAO */
       BINDEX(TOPO) = '0'B ; /* REG.BASE = 0 */
      END ;
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
      ('RECFATOR (RECONHECEDOR DE FATOR) RETORNOU COM ' ,
       'TIPO = ' , TIPO)(A , A , F(3)) ;
 END RECFATOR ;
 END RECTERM ;
 END EXPSIMP ;
 END RECEXP ;
1REDUZ : PROC ;
  DCL VERDIM              ENTRY RETURNS(BIT(1)) ;
  DCL CONVERT             ENTRY (BIN FIXED(15,0)) ;
  DCL ARITREAL            ENTRY ;
  AUXERRO = '0'B ;
  DO TOPO = TOPO TO 2 BY -1 WHILE(PRIOR(TOPO) >= PRIDIN) ;
   AUXTIPO = 999 ;
   AUXOPER = OPER(TOPO) ;
   AUXOPAND = TIPOPAND(TOPO) ;
   AUXMUM = TIPOPAND(TOPO-1) ;
   IF AUXOPER = 32 /* "¬" */
   THEN IF AUXOPAND = 3 | AUXOPAND <= 0
        THEN DO ;
               AUXTIPO = 3 ; /* EXPRESSAO BOOLEANA */
               IF GERA
               THEN DO ;
                      IF TIPEND(TOPO) ¬= 1 /* VARIAVEL? */
                      THEN DO ; /* GERA INSTRUCAO NOT */
```

```
                        /* GERA INSTRUCAO SET REG. 2 */
                        CALL PLAPP34(COSET,0,0,0,2,BINDEX(TOPO),
                                     ENDINDEX(TOPO)) ;
                        TEMBOOL = TEMBOOL - 1;/*END.TEMP.BOOL*/
                        /* GERA INSTRUCAO NOT */
                        CALL PLAPP34(CONOT,0,0,BOPAND(TOPO),
                             OPAND(TOPO),BLOCO+1,TEMBOOL) ;
                        OPAND(TOPO-1) = TEMBOOL ;
                        BOPAND(TOPO-1) = BLOCO + 1 ;
                        TIPEND(TOPO-1) = 2 ; /* VAR. DIRETO */
                       END ;
                  ELSE DO ;/* REDUCAO DE OPERANDO CONSTANTE */
                        IF OPAND(TOPO) = EFALSE /* CTE FALSE?*/
                        THEN OPAND(TOPO-1) = ETRUE ;
                        ELSE OPAND(TOPO-1) = EFALSE ;
                        TIPEND(TOPO-1) = 1 ; /* CTE */
                        BOPAND(TOPO-1) = '0'B ; /*BASE DA CTE*/
                       END ;
                  ENDINDEX(TOPO-1) = ENDZERO ; /* END.CTE ZERO*/
                  BINDEX(TOPO-1) = '0'B ; /* BASE DA CTE */
                 END ;
            END ;
       ELSE TIPOPAND(TOPO-1) = AUXOPAND ;
  ELSE IF AUXOPAND < 103 & AUXOPAND ¬= 53 & AUXOPAND ¬= 54 &
          AUXMUM < 103 & AUXMUM ¬= 53 & AUXMUM ¬= 54
/* * FORAM RETIRADOS BRANCOS DO INICIO DA LINHA * * * */
THEN IF AUXOPAND = 4 | AUXMUM = 4
     THEN DO ;
          IF AUXOPER >= 35 & (AUXOPAND <= 0 | AUXOPAND = 4)
             & (AUXMUM <= 0 | AUXMUM = 4)
          THEN DO ;
               AUXTIPO = 3 ; /* EXPRESSAO BOOLEANA */
               IF GERA
               THEN DO ;
                    IF TIPEND(TOPO) = 1 /* CTE? */
                    THEN IF TCHAR(TOPO) ¬= 1
                         THEN GERA = '0'B ;
                    IF TIPEND(TOPO-1) = 1
                    THEN IF TCHAR(TOPO-1) ¬= 1
                         THEN GERA = '0'B ;
                    IF ¬GERA
                    THEN DO ;
                         /* DADO ALFANUMERICO COM MAIS DE */
                         /* 1 CARACTER E' PERMITIDO APENAS*/
                         /* EM COMANDO 'WRITE'.           */
                         ERRO = 121 ;
                         COLUNA = PTR - 1;
                         CALL PLAPP01;/*IMP MSG ERRO*/
                        END ;
                   END ;
              IF GERA
              THEN DO ;
                   IF TIPEND(TOPO)¬=1 | TIPEND(TOPO-1)¬=1
                   THEN DO ; /* UM DOS OPERANDOS E' VARIAVEL */
                        /* GERA INST SET REG. 2 */
                        CALL PLAPP34(COSET,0,0,0,2,
                             BINDEX(TOPO-1),ENDINDEX(TOPO-1));
                        /* GERA INST SET REG. 3 */
                        CALL PLAPP34(COSET,0,0,0,3,
                             BINDEX(TOPO),ENDINDEX(TOPO)) ;
```

```
                        /* GERA INST COMP ALFA */
                        CALL PLAPP34(COCOMP,0,4,
                             BOPAND(TOPO-1),OPAND(TOPO-1),
                             BOPAND(TOPO),OPAND(TOPO)) ;
                        TEMBOOL = TEMBOOL - 1 ;
                        /* GERA INST ATRIB TEMP = FALSE */
                        CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                        CALL PLAPP34(COATRIB,0,3,0,EFALSE,
                                     BLOCO+1,TEMBOOL) ;
                        /* GERA INST DESV SE COMPARACAO RE- */
                        /* SULTOU "FALSE"                   */
                        CALL PLAPP34(CODESV,0,0,0,
                             CODBR(AUXOPER),0,CINST+LC1+3) ;
                        /* GERA INST ATRIB TEMP = TRUE */
                        CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                        CALL PLAPP34(COATRIB,0,3,0,ETRUE,
                                     BLOCO+1,TEMBOOL) ;
                        OPAND(TOPO-1) = TEMBOOL ;
                        BOPAND(TOPO-1) = BLOCO + 1 ;
                        TIPEND(TOPO-1) = 2 ; /* VAR. DIRETO */
                      END ;
                   ELSE DO ; /* REDUCAO DE OPERANDOS CTES */
                        AC = TEXTO(TOPO-1) ;
                        BC = TEXTO(TOPO) ;
                        IF AC < BC
                        THEN IF AUXOPER = 35 | AUXOPER = 37 |
                                AUXOPER = 40
                             THEN OPAND(TOPO-1) = ETRUE ;
                             ELSE OPAND(TOPO-1) = EFALSE ;
                        ELSE IF AC = BC
                             THEN IF AUXOPER >= 38
                                  THEN OPAND(TOPO-1) = ETRUE ;
                                  ELSE OPAND(TOPO-1) = EFALSE;
                             ELSE IF AUXOPER<=36 | AUXOPER=39
                                  THEN OPAND(TOPO-1) = ETRUE ;
                                  ELSE OPAND(TOPO-1) = EFALSE;
                        BOPAND(TOPO-1) = '0'B ;/*BASE DE CTE*/
                        TIPEND(TOPO-1) = 1 ; /* CTE */
                      END ;
                   ENDINDEX(TOPO-1) = ENDZERO ; /*END CTE ZERO*/
                   BINDEX(TOPO-1) = '0'B ; /* BASE DE CTE */
                 END ;
            END ;
      END ;
        ELSE IF AUXOPAND = 3 | AUXMUM = 3
             THEN DO ;
                    IF AUXOPER >= 30 & (AUXOPAND <= 0 |
                     AUXOPAND = 3) & (AUXMUM <= 0 | AUXMUM = 3)
-/* * * * * * * *  RETIRADOS BRANCOS DO INICIO DA LINHA  * * * * */
-THEN DO ;
      AUXTIPO = 3 ; /* EXPRESSAO BOOL. */
      IF GERA
      THEN DO ; /* AMBOS OPERANDOS BOOL. */
             IF TIPEND(TOPO) ¬= 1 | TIPEND(TOPO-1) ¬= 1
             THEN DO ; /* UM DOS OPERANDOS E' VARIAVEL */
                    TEMBOOL = TEMBOOL - 1 ;
                    IF AUXOPER >= 35
                    THEN DO ; /* OPERADOR RELACIONAL */
                           /* GERA INST SET REG. 2, REG. 3 E */
                           /* COMP BOOLEANA                  */
```

```
                    CALL PLAPP34(COSET,0,0,0,2,
                         BINDEX(TOPO-1),ENDINDEX(TOPO-1)) ;
                    CALL PLAPP34(COSET,0,0,0,3,
                         BINDEX(TOPO),ENDINDEX(TOPO)) ;
                    CALL PLAPP34(COCOMP,0,3,BOPAND(TOPO-1),
                         OPAND(TOPO-1),BOPAND(TOPO),
                         OPAND(TOPO)) ;
                    /* GERA INST ATRIB TEMP = FALSE */
                    CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                    CALL PLAPP34(COATRIB,0,3,0,EFALSE,
                         BLOCO+1,TEMBOOL) ;
                    /* GERA INST DESV SE COMPARACAO */
                    /* RESULTOU 'FALSE'             */
                    CALL PLAPP34(CODESV,0,0,0,
                         COD8R(AUXOPER),0,CINST+LC1+3) ;
                    /* GERA INST ATRIB TEMP = TRUE */
                    CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                    CALL PLAPP34(COATRIB,0,3,0,ETRUE,
                         BLOCO+1,TEMBOOL) ;
                  END ;
                ELSE DO ; /* "&" OU "|" */
                    /* GERA INST "AND" OU "OR" */
                    /* GERA INST SET REGS */
                    CALL PLAPP34(COSETM,BINDEX(TOPO-1),
                         ENDINDEX(TOPO-1),BINDEX(TOPO),
                         ENDINDEX(TOPO),0,ENDZERO) ;
                    IF AUXOPER = 30
                    THEN AUXOPER = COAND ;
                    ELSE AUXOPER = COOR ;
                    CALL PLAPP34(AUXOPER,BOPAND(TOPO-1),
                         OPAND(TOPO-1),BOPAND(TOPO),
                         OPAND(TOPO),BLOCO+1,TEMBOOL) ;
                  END ;
                OPAND(TOPO-1) = TEMBOOL ;
                BOPAND(TOPO-1) = BLOCO + 1 ;
                TIPEND(TOPO-1) = 2 ; /* VARIAVEL DIRETO */
              END ;
            ELSE DO ; /* REDUCAO DE OPERANDOS CTES */
                A = OPAND(TOPO-1) ;
                B = OPAND(TOPO) ;
                IF AUXOPER = 30 /* "&" ? */
                THEN IF A = ETRUE & B = ETRUE
                     THEN A = ETRUE ;
                     ELSE A = EFALSE ;
                ELSE IF AUXOPER = 31 /* "|" ? */
                     THEN IF A = ETRUE | B = ETRUE
                          THEN A = ETRUE ;
                          ELSE A = EFALSE ;
                     ELSE IF A = B
                          THEN IF AUXOPER <= 37
                               THEN A = EFALSE ;
                               ELSE A = ETRUE ;
                          ELSE IF AUXOPER = 35
                               THEN A = ETRUE ;
                               ELSE IF AUXOPER = 38
                                    THEN A = EFALSE ;
                                    ELSE IF AUXOPER = 37 |
                                            AUXOPER = 40
                                         THEN A = B ;
                OPAND(TOPO-1) = A ;
```

```
                    BOPAND(TOPO-1) = 0 ; /* BASE DE CTE */
                    TIPEND(TOPO-1) = 1 ; /* CTE */
                  END ;
              ENDINDEX(TOPO-1) = ENDZERO ; /* END CTE ZERO */
              BINDEX(TOPO-1) = 0 ; /* BASE DE CTE */
            END ;
      END ;
                      END ;
ELSE IF AUXOPAND = 52 | AUXOPAND = 102
      THEN DO ; /* AUXMUM < 3 | (AUXMUM >= 50 & AUXMUM < 53) | */
                /* (AUXMUM >= 100 & AUXMUM < 103)                */
            IF AUXOPER = 20 | AUXMUM < 50 & AUXOPER = 23 |
               AUXMUM > 2 & AUXOPER = 21
            THEN DO ;
                   AUXTIPO = 52 ; /* EXP MAT REAL */
                   GERA = '0'B ;
                   PUT FILE(SPRINT) SKIP EDIT ('INIBIDA A GERACAO',
                       ' DE CODIGO, POIS NAO FORAM IMPLEMENTADAS ',
                       'AS OPERACOES MATRICIAIS PREVISTAS')(A,A,A);
                 END ;
            END ;
/* * * * * * *       RETIRADOS BRANCOS NO INICIO DA LINHA    * * * */
ELSE IF AUXMUM = 52 | AUXMUM = 102
      THEN DO ;
            IF AUXOPER <= 21 | AUXOPAND < 3 & AUXOPER < 24
            THEN DO ;
                   AUXTIPO = 52 ; /* EXP MAT REAL */
                   GERA = '0'B ;
                   PUT FILE(SPRINT) SKIP EDIT ('INIBIDA A GERACAO ',
                       'DE CODIGO, POIS NAO FORAM IMPLEMENTADAS AS',
                       ' OPERACOES MATRICIAIS PREVISTAS')(A,A,A) ;
                 END ;
            END ;
      ELSE IF AUXOPAND >= 50
           THEN IF AUXMUM >= 50
                THEN DO ;
                      IF AUXOPER <= 21
                      THEN IF AUXMUM = 51 | AUXMUM = 101 |
                              AUXOPAND = 51 | AUXOPAND = 101
                           THEN DO ;
                                  AUXTIPO = 51 ; /* EXP.MAT.INT */
                                  GERA = '0'B ;
                                  PUT FILE(SPRINT) SKIP EDIT
                                     ('INIBIDA A GERACAO DE CODIGO' ,
                                     'POIS NAO FORAM IMPLEMENTADAS ',
                                     'AS OPERACOES MATRICIAIS ',
                                     'PREVISTAS')(A,A,A,A) ;
                                END ;
                           ELSE AUXTIPO = 50 ; /* EXP.MAT.INDEF.*/
                     END ;
                ELSE DO ;
                      IF AUXOPER = 20 | AUXOPER = 23
                      THEN IF AUXMUM > 0
                           THEN DO ;
                                  AUXTIPO = AUXMUM + 50 ; /* EXP. */
                                    /* MAT. DO MESMO TIPO DE AUXMUM*/
                                  GERA = '0'B ;
                                  PUT FILE(SPRINT) SKIP EDIT
                                     ('INIBIDA A GERACAO DE CODIGO ',
                                     'POIS NAO FORAM IMPLEMENTADAS ',
```

```
                                    'AS OPERACOES MATRICIAIS ' ,
                                    'PREVISTAS')(A,A,A,A) ;
                                END ;
                            ELSE AUXTIPO = MOD(AUXOPAND,10) + 50 ;
                        END ;
                ELSE IF AUXMUM >= 50
                     THEN DO ;
                          IF AUXOPER < 30
                          THEN IF AUXOPER = 22
                               THEN AUXTIPO = 52 ; /* EXP MAT REAL */
                               ELSE IF AUXOPER < 24
                                    THEN IF AUXOPAND < 1
                                         THEN AUXTIPO=MOD(AUXMUM,10)+50;
                                         ELSE AUXTIPO = AUXOPAND + 50 ;
                                    ELSE IF AUXOPAND ¬= 2
                                         THEN AUXTIPO = 51 ; /* EXP */
                                                             /* MAT INTEIRA */
                          GERA = '0'B ;
                          PUT FILE(SPRINT) SKIP EDIT('INIBIDA A GERACAO',
                              ' DE CODIGO POIS NAO FORAM IMPLEMENTADAS ',
                              'AS OPERACOES MATRICIAIS PREVISTAS')(A,A,A);
                        END ;
/* * NESTE PONTO FORAM RETIRADOS OS BRANCOS DO INICIO DA LINHA */
ELSE IF AUXOPAND = 2 | AUXMUM = 2 /* UM DOS OPERANDOS REAL? */
     THEN IF AUXOPER >= 35 /* '¬=', '>', '<', '=', '>=' OU '<='? */
          THEN DO ;
                 AUXTIPO = 3 ; /* EXP. BOOLEANA */
                 IF GERA
                 THEN DO ; /* OPERANDOS NAO INVALIDOS */
                      IF TIPEND(TOPO) ¬= 1 | TIPEND(TOPO-1) ¬= 1
                      THEN DO ; /*UM DOS OPERANDOS E' VARIAVEL */
                           IF AUXOPAND ¬= 2 /* INTEIRO? */
                           THEN CALL CONVERT(TOPO) ;
                           IF AUXMUM ¬= 2 /* INTEIRO? */
                           THEN CALL CONVERT(TOPO-1) ;
                           TEMBOOL = TEMBOOL - 1 ;
                           /* GERA INST. SET REG 2 */
                           CALL PLAPP34(COSET,0,0,0,2,
                                BINDEX(TOPO-1),ENDINDEX(TOPO-1));
                           /* GERA INST. SET REG 3 */
                           CALL PLAPP34(COSET,0,0,0,3,
                                BINDEX(TOPO),ENDINDEX(TOPO)) ;
                           /* GERA INST. COMPARACAO REAL */
                           CALL PLAPP34(COCOMP,0,2,
                                BOPAND(TOPO-1),OPAND(TOPO-1),
                                BOPAND(TOPO),OPAND(TOPO)) ;
                           /* GERA INST. ATRIB. TEMP=FALSE */
                           CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                           CALL PLAPP34(COATRIB,0,3,0,EFALSE,
                                BLOCO+1,TEMBOOL);
                           /* GERA INST. DESVIO SE COMPARACAO */
                           /* RESULTOU FALSE                  */
                           CALL PLAPP34(CODESV,0,0,0,
                                CODBR(AUXOPER),0,CINST+LC1+3) ;
                           /* GERA INST. ATRIB TEMP=TRUE */
                           CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                           CALL PLAPP34(COATRIB,0,3,0,ETRUE,
                                BLOCO+1,TEMBOOL) ;
                           OPAND(TOPO-1) = TEMBOOL ;
                           BOPAND(TOPO-1) = BLOCO + 1 ;
```

```
                        TIPEND(TOPO-1) = 2 ; /* VAR. DIRETO */
                      END ;
                ELSE DO ; /* REDUCAO DE OPERANDOS CTES */
                      IF AUXMUM ¬= 2 /* INTEIRO? */
                      THEN AR = VALINT(OPAND(TOPO-1)) ;
                      ELSE AR = VALREAL(OPAND(TOPO-1)) ;
                      IF AUXOPAND ¬= 2 /* INTEIRO? */
                      THEN BR = VALINT(OPAND(TOPO)) ;
                      ELSE BR = VALREAL(OPAND(TOPO)) ;
                      IF AR < BR
                      THEN IF AUXOPER = 35 | AUXOPER = 37 |
                              AUXOPER = 40
                           THEN A = ETRUE ;
                           ELSE A = EFALSE ;
                      ELSE IF AR = BR
                           THEN IF AUXOPER >= 38
                                THEN A = ETRUE ;
                                ELSE A = EFALSE ;
                           ELSE IF AUXOPER<=36 | AUXOPER=39
                                THEN A = ETRUE ;
                                ELSE A = EFALSE ;
                      OPAND(TOPO-1) = A ;
                      BOPAND(TOPO-1) = 0 ; /* BASE DE CTE */
                      TIPEND(TOPO-1) = 1 ; /*CTE */
                    END ;
                ENDINDEX(TOPO-1) = ENDZERO ; /* END CTE 0 */
                BINDEX(TOPO-1) = 0 ; /* BASE DE CTE */
              END ;
        END ;
    ELSE DO ; /* OPERADORES ARITMETICOS */
           IF AUXOPER < 24 /* IDEM, MENOS 'DIV' E 'MOD'? */
           THEN CALL ARITREAL ; /* OPERACOES REAIS */
         END ;
/* FORAM RETIRADOS OS BRANCOS DO INICIO DA LINHA */
ELSE IF AUXOPAND = 1 | AUXMUM = 1 /* UM DOS OPERANDOS INT.? */
   THEN IF AUXOPER >= 35 /* "¬=",">","<","=",">=" OU "<="? */
        THEN DO ;
           AUXTIPO = 3 ; /* EXP BOOLEANA */
           IF GERA
           THEN DO ; /* AMBOS OPERANDOS INTEIROS */
                IF TIPEND(TOPO) ¬= 1 | TIPEND(TOPO-1) ¬= 1
                THEN DO ; /* UM DOS OPERANDOS E' VAR. */
                     TEMBOOL = TEMBOOL - 1 ;
                     /* GERA INST. SET REG 2 */
                     CALL PLAPP34(COSET,0,0,0,2,
                          BINDEX(TOPO-1),ENDINDEX(TOPO-1));
                     /* GERA INST. SET REG 3 */
                     CALL PLAPP34(COSET,0,0,0,3,
                          BINDEX(TOPO),ENDINDEX(TOPO));
                     /* GERA INST. COMPARACAO INTEIRA */
                     CALL PLAPP34(COCOMP,0,1,
                          BOPAND(TOPO-1),OPAND(TOPO-1),
                          BOPAND(TOPO),OPAND(TOPO)) ;
                     /* GERA INST. ATRIB TEMP=FALSE */
                     CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                     CALL PLAPP34(COATRIB,0,3,0,EFALSE,
                          BLOCO+1,TEMBOOL) ;
                     /* GERA INST. DESVIO SE COMPARACAO */
                     /* RESULTOU FALSE                  */
                     CALL PLAPP34(CODESV,0,0,0,
```

```
                                    CODBR(AUXOPER),0,CINST+LC(1+3) ;
                                  /* GERA INST. ATRIB TEMP=TRUE */
                                  CALL PLAPP34(COZEREG,0,1,0,1,0,1) ;
                                  CALL PLAPP34(COATRIB,0,3,0,ETRUE,
                                       BLOCO+1,TEMBOOL) ;
                                  OPAND(TOPO-1) = TEMBOOL ;
                                  BOPAND(TOPO-1) = BLOCO + 1 ;
                                  TIPEND(TOPO-1) = 2 ; /* VAR DIRETO */
                                 END ;
                            ELSE DO ; /* REDUCAO DE OPERANDOS CTES */
                                  A = VALINT(OPAND(TOPO-1)) ;
                                  B = VALINT(OPAND(TOPO)) ;
                                  IF A < B
                                  THEN IF AUXOPER = 35 | AUXOPER = 37 |
                                          AUXOPER = 40
                                       THEN A = ETRUE ;
                                       ELSE A = EFALSE ;
                                  ELSE IF A = B
                                       THEN IF AUXOPER >= 38
                                            THEN A = ETRUE ;
                                            ELSE A = EFALSE ;
                                       ELSE IF AUXOPER<=36 | AUXOPER=39
                                            THEN A = ETRUE ;
                                            ELSE A = EFALSE ;
                                  OPAND(TOPO-1) = A ;
                                  BOPAND(TOPO-1) = 0 ; /*BASE DE CTE*/
                                  TIPEND(TOPO-1) = 1 ; /* CTE */
                                 END ;
                             ENDINDEX(TOPO-1) = ENDZERO ; /* END CTE 0 */
                             BINDEX(TOPO-1) = 0 ; /* BASE DE CTE */
                            END ;
                    END ;
               ELSE DO ; /* "+","-","/","*","DIV","MOD","&","|" OU "¬"*/
                      IF AUXOPER < 30 /* IDEM,MENOS "&","|" E "¬"? */
                      THEN IF AUXOPER = 22 /* "/" ? */
                           THEN CALL ARITREAL ; /* DIVISAO REAL */
/* FORAM RETIRADOS OS BRANCOS DO INICIO DA LINHA */
ELSE DO ; /* OPERACOES INTEIRAS */
       AUXTIPO = 1 ; /* EXPRESSAO INTEIRA */
       IF GERA
       THEN DO ;
              IF TIPEND(TOPO) ¬= 1 | TIPEND(TOPO-1) ¬= 1
              THEN DO ; /* UM DOS OPERANDOS E' VARIAVEL */
                     /* GERA INST. SETM */
                     CALL PLAPP34(COSETM,BINDEX(TOPO-1),
                                  ENDINDEX(TOPO-1),BINDEX(TOPO),
                                  ENDINDEX(TOPO),0,ENDZERO) ;
                     TEMINT = TEMINT - 1 ; /* END. DO TEMPORARIO */
                     /* GERA INST. ARITMETICA INTEIRA */
                     CALL PLAPP34(CODOP(AUXOPER),BOPAND(TOPO-1),
                                  OPAND(TOPO-1),BOPAND(TOPO),
                                  OPAND(TOPO),BLOCO+1,TEMINT) ;
                     OPAND(TOPO-1) = TEMINT ;
                     BOPAND(TOPO-1) = BLOCO + 1 ;
                     TIPEND(TOPO-1) = 2 ; /* VAR. DIRETO */
                    END ;
              ELSE DO ; /* REDUCAO DE OPERANDOS CTES */
                     A = VALINT(OPAND(TOPO-1)) ;
                     B = VALINT(OPAND(TOPO)) ;
                     IF AUXOPER = 20 /* OPERADOR "+" ? */
```

```
                    THEN CTEINT = A + B ;
                    ELSE IF AUXOPER = 21 /* OPERADOR "-" ? */
                         THEN CTEINT = A - B ;
                         ELSE IF AUXOPER = 23 /* OPERADOR "*" ? */
                              THEN CTEINT = A * B ;
                              ELSE /* OPERADOR "DIV" OU "MOD" */
                                   IF B = 0
                                   THEN DO ; /* DIVISAO POR 0 */
                                          ERRO = 129 ;
                                          COLUNA = PTR - 1 ;
                                          /* IMP. MSG. ERRO */
                                          CALL PLAPP01 ;
                                        END ;
                                   ELSE IF AUXOPER = 24
                                        THEN CTEINT = A / B ;
                                        ELSE CTEINT = MOD(A,B) ;
                    /* INSERE CTE CALCULADA NA TAB DE CTES INT. */
                    CALL PLAPP13 ;
                    OPAND(TOPO-1) = ENDCTE ; /* END CTE RESULT. */
                    BOPAND(TOPO-1) = 0 ; /* BASE DE CTE */
                    TIPEND(TOPO-1) = 1 ; /* CONSTANTE */
                  END ;
               ENDINDEX(TOPO-1) = ENDZERO ; /* END CTE ZERO */
               BINDEX(TOPO-1) = 0 ; /* BASE DE CTE */
             END ;
      END ;
                END ;
/* * * FORAM RETIRADOS BRANCOS DO INICIO DA LINHA * * * */
ELSE /* AUXMUM <= 0 & AUXOPAND <= 0 */
      IF AUXOPER >= 30
      THEN AUXTIPO = 3 ; /* EXP BOOLEANA */
      ELSE IF AUXOPER >= 24
           THEN AUXTIPO = 1 ; /* EXP INTEIRA */
           ELSE IF AUXOPER = 22
                THEN AUXTIPO = 2 ; /* EXP REAL */
                ELSE IF AUXOPAND = 0
                     THEN AUXTIPO = 0 ; /* EXP INDEFINIDA */
                     ELSE AUXTIPO = -1 ; /* EXP SEM TIPO */
   IF AUXTIPO = 999
   THEN AUXERRO = '1'B ; /* EXPRESSAO COM ERRO */
   ELSE DO ;
          TIPOPAND (TOPO-1) = AUXTIPO ;
          IF AUXOPAND >= 50 & AUXMUM >= 50
          THEN AUXERRO = AUXERRO | VERDIM ;
        END ;
   GERA = GERA & ¬AUXERRO ;
  END ;
  IF AUXERRO
  THEN DO ; /* MISTURA DE MODO INVALIDA OU OPERACAO MATRICIAL     */
            /* ENTRE ARRAYS DE DIMENSOES DIFERENTES NA EXPRESSAO */
         ERRO = 118 ;
         COLUNA = PTR - 1 ;
         CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
       END ;
VERDIM : PROC RETURNS(BIT(1)) ;
 DCL RETCODE              BIT(1) STATIC ;
 DCL (I , J , PT1 , PT2) BIN FIXED(15,0) STATIC ;
 PT1 = TIPEND(TOPO) ;
 PT2 = TIPEND(TOPO-1) ;
 RETCODE = TABDIM(PT1) ¬= TABDIM(PT2) ;
```

```
J = TABDIM(PT1) ;
PT1 = PT1 + 1 ;
PT2 = PT2 + 1 ;
DO I = 1 TO J WHILE(¬RETCODE) ;
  RETCODE = TABDIM(PT1+1)-TABDIM(PT1) ¬= TABDIM(PT2+1)-TABDIM(PT2);
  PT1 = PT1 + 2 ;
  PT2 = PT2 + 2 ;
END ;
RETURN(RETCODE) ;
END VERDIM ;
ARITREAL : PROC ;
AUXTIPO = 2 ; /* EXP REAL */
IF GERA
THEN DO ;
       IF TIPEND(TOPO) ¬= 1 | TIPEND(TOPO-1) ¬= 1
       THEN DO ; /* UM DOS OPERANDOS E' VARIAVEL */
              IF AUXOPAND ¬= 2 /* INTEIRO? */
              THEN CALL CONVERT(TOPO) ;
              IF AUXMUM ¬= 2
              THEN CALL CONVERT(TOPO-1) ;
              /* GERA INST CARGA REGS */
              CALL PLAPP34(COSETM,BINDEX(TOPO-1),ENDINDEX(TOPO-1),
                           BINDEX(TOPO),ENDINDEX(TOPO),0,ENDZERO) ;
              TEMREAL = TEMREAL - 1 ; /* END DO TEMPORARIO */
              /* GERA INST ARITMETICA REAL */
              CALL PLAPP34(CODOP(AUXOPER-4),BOPAND(TOPO-1),
                           OPAND(TOPO-1),BOPAND(TOPO),
                           OPAND(TOPO),BLOCO+1,TEMREAL) ;
              OPAND(TOPO-1) = TEMREAL ;
              BOPAND(TOPO-1) = BLOCO + 1 ;
              TIPEND(TOPO-1) = 2 ; /* VARIAVEL DIRETO */
            END ;
       ELSE DO ; /* REDUCAO DE OPERANDOS CTES */
              IF AUXMUM ¬= 2 /* INTEIRO? */
              THEN AR = VALINT(OPAND(TOPO-1)) ;
              ELSE AR = VALREAL(OPAND(TOPO-1)) ;
              IF AUXOPAND ¬= 2 /* INTEIRO ? */
              THEN BR = VALINT(OPAND(TOPO)) ;
              ELSE BR = VALREAL(OPAND(TOPO)) ;
              IF AUXOPER = 20 /* OPERADOR "+"? */
              THEN CTEREAL = AR + BR ;
              ELSE IF AUXOPER = 21 /* OPERADOR "-"? */
                   THEN CTEREAL = AR - BR ;
                   ELSE IF AUXOPER = 22 /* OPERADOR "/"? */
                        THEN IF BR = 0
                             THEN DO ; /* DIVISAO POR ZERO */
                                    ERRO = 129 ;
                                    COLUNA = PTR - 1 ;
                                    CALL PLAPP01 ;/*IMP MSG ERRO*/
                                  END ;
                             ELSE CTEREAL = AR / BR ;
                        ELSE CTEREAL = AR * BR ;
              /* INSERE CTE CALCULADA NA TAB CTES REAIS */
              CALL PLAPP14 ;
              OPAND(TOPO-1) = ENDCTE ; /* END CTE RESULTANTE */
              BOPAND(TOPO-1) = 0 ; /* BASE DE CTE */
              TIPEND(TOPO-1) = 1 ; /* CONSTANTE */
            END ;
       ENDINDEX(TOPO-1) = ENDZERO ; /* END CTE ZERO */
       BINDEX(TOPO-1) = 0 ; /* BASE DE CTE */
```

```
      END ;
 END ARITREAL ;
1CONVERT : PROC(PT) ;
 DCL PT           BIN FIXED(15,0) ;
 IF TIPEND(PT) = 1 /* CTE ? */
 THEN DO ; /* CRIA CTE REAL */
        CTEREAL = VALINT(OPAND(PT)) ;
        CALL PLAPP14 ; /* INSERE NA TAB CTES REAIS */
        OPAND(PT) = ENDCTE ; /* END DA CTE GERADA */
      END ;
 ELSE DO ; /* GERA CONVERSAO REAL */
        /* GERA INSTRUCAO SET REG. 2 */
        CALL PLAPP34(COSET,0,0,0,2,BINDEX(PT),ENDINDEX(PT)) ;
        /* GERA INSTRUCAO SET REG. 3 */
        CALL PLAPP34(COSET,0,0,0,3,0,ENDZERO) ;
        TEMREAL = TEMREAL - 1 ; /* END TEMP REAL */
        /* GERA INSTRUCAO CONVERSAO REAL */
        CALL PLAPP34(COCVT,0,2,BOPAND(PT),OPAND(PT),BLOCO+1,
                     TEMREAL) ;
        OPAND(PT) = TEMREAL ;
        BOPAND(PT) = BLOCO + 1 ;
        ENDINDEX(PT) = ENDZERO ;
        BINDEX(PT) = 0 ; /* BASE DE CTE */
      END ;
 END CONVERT ;
 END REDUZ ;
1PRTPILHA : PROC ;
 DCL (AUX,I)          BIN FIXED(15,0) ;
 DCL TOPNDCHAR(-1:4)  CHAR(5) INIT('N.DEC' , 'INDEF' , 'INT.' ,
                                   'REAL' , 'BOOL.' , 'CHAR') ;
 DCL (CHAR51,CHAR52)  CHAR(5) ;
 DCL (CHAR3,TENDCHAR) CHAR(3) ;
 DCL OPCHAR1(20:26) CHAR(3) INIT('+' , '-' , '/' , '*' , 'DIV' ,
                                 'MOD' , '#') ;
 DCL OPCHAR2(30:40) CHAR(3) INIT('&' , '|' , '¬' , '?' , '?' , '¬=',
                                 '>' , '<' , '=' , '>=' , '<=') ;
 PUT FILE(SPRINT) SKIP EDIT
     ('I' , 'OPER' , 'PRIDIN' , 'OPAND' , 'TIPOPAND' , 'TIPEND')
     (COL(3) , A , COL(6) , A , COL(12) , A , COL(19) , A ,
      COL(27) , A , COL(43) , A) ;
 DO I = 1 TO TOPO ;
   IF OPER(I) >= 20 & OPER(I) <= 26
   THEN CHAR3 = OPCHAR1(OPER(I)) ;
   ELSE IF OPER(I) >= 30 & OPER(I) <= 40
        THEN CHAR3 = OPCHAR2(OPER(I)) ;
        ELSE CHAR3 = '???' ;
   AUX = TIPOPAND(I) ;
   IF AUX < -1 | AUX > 4 & AUX < 50 | AUX > 54 & AUX < 100 | AUX>104
   THEN CHAR51 = '?????' ;
   ELSE CHAR51 = TOPNDCHAR(MOD(AUX,10)) ;
   IF AUX > 4
   THEN IF AUX < 100
        THEN CHAR52 = 'MATR.' ;
        ELSE CHAR52 = 'PMAT.' ;
   ELSE CHAR52 = 'SIMP.' ;
   IF AUX >= 50
   THEN TENDCHAR = 'DIM' ;
   ELSE IF TIPEND(I) < 1 | TIPEND(I) > 3
        THEN TENDCHAR = 'ERR' ;
        ELSE IF TIPEND(I) = 1
```

```
            THEN TENDCHAR = 'CTE' ;
            ELSE IF TIPEND(I) = 2
                 THEN TENDCHAR = 'DIR' ;
                 ELSE TENDCHAR = 'IND' ;
   PUT FILE(SPRINT) SKIP EDIT
       (I , OPER(I) , CHAR3 , PRIOR(I) , OPAND(I) , TIPOPAND(I) ,
        CHAR51 , CHAR52 , TIPEND(I) , TENDCHAR)(F(3) , X(1) , F(3) ,
        X(1) , A , X(1) , F(3) , X(3) , F(3) , X(3) , F(3) , X(1) ,
        A , X(1) , A , X(1) , F(3) , X(1) , A) ;
 END ;
 END PRTPILHA ;
1PILHA_CHEIA : PROC RETURNS(BIT(1)) ;
 DCL RETCODE BIT(1) ;
 IF TOPO = 64
 THEN DO ; /* PLAPP30 - RESTRICAO DE IMPLEMENTACAO - NUMERO */
           /* MAXIMO DE OPERADORES NA EXPRESSAO FOI ULTRA-  */
           /* PASSADO.                                      */
        ERRO = 70 ;
        COLUNA = PTR - 1 ;
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        ERRO = 41 ; /* COMPILACAO ABORTADA */
        CALL PLAPP01 ; /* IMPRIME MENSAGEM DE ERRO */
        RETCODE = '1'B ;
      END ;
 ELSE RETCODE = '0'B ;
 RETURN(RETCODE) ;
 END PILHA_CHEIA ;
 END PLAPP30 ;
```

```
/* PLAPP31 - GERA PROGRAMA OBJETO EM DISCO.                          */
PLAPP31 : PROC ;
/*******************************************************************/
/* ESTA PROCEDURE GERA O PROGRAMA OBJETO EM DISCO, EM FORMATO      */
/* EXECUTAVEL, NO ARQUIVO 'OBJ2', A PARTIR DO ARQUIVO 'OBJ1',      */
/* QUE CONTEM AS QUADRUPLAS COM AS INSTRUCOES DO PROGRAMA          */
/* COMPILADO.                                                      */
/* CONTEUDO DE 'OBJ2':                                             */
/* INDICACAO DO TAMANHO DE CADA AREA A SER ALOCADA PARA EXE-       */
/* CUCAO DO PROGRAMA (AREAS PARA CONSTANTES, VARIAVEIS E TEM-      */
/* PORARIOS INTEIROS, REAIS, BOOLEANOS E CHAR), NUMERO DE CONS-    */
/* TANTES INTEIRAS, REAIS, BOOLEANAS E CHAR, VALORES DAS CONS-     */
/* TANTES USADAS NO PROGRAMA, AS INSTRUCOES A SEREM EXECUTADAS     */
/* E NUMERO MAXIMO OCORRIDO DE PROCEDURES (E/OU FUNCOES) EM-       */
/* BUTIDAS.                                                        */
/*******************************************************************/
1 /* MACRO BLOCO E BLOCMAX */
$CONTINUE WITH PLAMV02 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO TABINT, TABREAL, TABALFA */
$CONTINUE WITH PLAMV07 RETURN
1 /* MACRO GERA E GERAEND */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1 /* MACRO OBJ2 */
$CONTINUE WITH PLAMV14 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO REGAREA */
$CONTINUE WITH PLAMV16 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL (I,J,K)          BIN FIXED(15,0) ;
DCL QFIX(5,4)         BIN FIXED(31,0) ;
DCL QFLOAT(20)        BIN FLOAT(21) BASED (P) ;
DCL QFIXED(20)        BIN FIXED(31,0) BASED (P) ;
DCL QCHAR(80)         CHAR(1) BASED (P) ;
DCL 1 QUADRA(5) BASED (P) ,
      2 COP           BIT (8) ,
      2 OPND(3)       CHAR (5) ;
DCL P                 POINTER ;
DCL ADDR              BUILTIN ;
P = ADDR (QFIX) ;
DCL EOF               BIT(1) INIT('0'B) ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
   ('PLAPP31 (GERA PROGRAMA OBJETO EM DISCO) INICIOU') (A) ;
ON ENDFILE (OBJ1) EOF = '1'B ;
OPEN FILE (OBJ2) OUTPUT TITLE ('2') ;
CO = COAREA ; /* QUADRUPLA C/ TAM.AREAS DO PROGRAMA DO USUARIO */
CINT = 200 ; /* TAM.AREA P/CONSTANTES E VARIAVEIS INTEIRAS */
CREAL = 200 ; /* TAM.AREA P/CONSTANTES E VARIAVEIS REAIS */
CBOOL = 200 ; /* TAM.AREA P/CONSTANTES E VARIAVEIS BOOLEANAS */
CCHAR = 200 ; /* TAM.AREA P/CONSTANTES E VARIAVEIS CHAR */
TINT = 200 ; /* TAM.AREA P/TEMPORARIOS INTEIROS */
TREAL = 200 ; /* TAM.AREA P/TEMPORARIOS REAIS */
TBOOL = 200 ; /* TAM.AREA P/TEMPORARIOS BOOLEANOS */
TCHAR = 200 ; /* TAM.AREA P/TEMPORARIOS CHAR */
```

```
NCINT = PROXINT - 1 ; /* NUMERO DE CONSTANTES INTEIRAS */
NCREAL = PROXRE - 1 ; /* NUMERO DE CONSTANTES REAIS */
NCBOOL = 2 ; /* NUMERO DE CONSTANTES BOOLEANAS */
          /* NUMERO DE CONSTANTES CHAR (EM CARACTERES) */
NCCHAR = PTALFA(ULTALFA) + TAMALFA(ULTALFA) - 1 ;
NQUAD = CINST ; /* TAM.AREA P/INSTRUCOES */
NBMAX = BLOCMAX + 1 ;
I = 5 ;
/* SAO SEMPRE GERADAS CONSTANTES INTEIRAS */
COP(I) = COCINT ; /* LISTA DE CTES INTEIRAS */
QFIX(I,2) = PROXINT - 1 ;
J = I * 4 - 1 ;
DO K = 1 TO PROXINT - 1 ;
  IF J > 20
  THEN DO ;
         WRITE FILE (OBJ2) FROM (QFIX) ;
         J = 1 ;
       END ;
  QFIXED(J) = VALINT(K) ;
  J = J + 1 ;
END ;
I = (J + 2) / 4 + 1 ;
IF I > 5
THEN DO ;
       WRITE FILE (OBJ2) FROM (QFIX) ;
       I = 1 ;
     END ;
IF PROXRE > 1
THEN DO ; /* FORAM GERADAS CONSTANTES REAIS */
       COP(I) = COCREAL ; /* LISTA DE CTES REAIS */
       QFIX(I,2) = PROXRE - 1 ;
       J = I * 4 - 1 ;
       DO K = 1 TO PROXRE - 1 ;
         IF J > 20
         THEN DO ;
                WRITE FILE (OBJ2) FROM (QFIX) ;
                J = 1 ;
              END ;
         QFLOAT(J) = VALREAL(K) ;
         J = J + 1 ;
       END ;
       I = (J + 2) / 4 + 1 ;
       IF I > 5
       THEN DO ;
              WRITE FILE (OBJ2) FROM (QFIX) ;
              I = 1 ;
            END ;
     END ;
COP(I) = COCBOOL ; /* LISTA DE CTES BOOLEANAS */
QFIX(I,2) = 2 ;
QFIX(I,3) = 0 ; /* FALSE */
QFIX(I,4) = 1 ; /* TRUE */
I = I + 1 ;
IF I > 5
THEN DO ;
       WRITE FILE (OBJ2) FROM (QFIX) ;
       I = 1 ;
     END ;
K = PTALFA(ULTALFA) + TAMALFA(ULTALFA) - 1 ;
IF K >= 1
```

```
THEN DO ; /* FORAM GERADAS CTES ALFANUMERICAS */
        J = I * 16 - 15 ;
        COP(I) = COCCHAR ; /* LISTA DE CTES ALFANUMERICAS */
        QFIX(I,2) = K ;
        J = J + 8 ;
        DO K = 1 TO K ;
          IF J > 80
          THEN DO ;
                 WRITE FILE (OBJ2) FROM (QFIX) ;
                 J = 1 ;
               END ;
          QCHAR(J) = TEXTO(K) ;
          J = J + 1 ;
        END ;
        I = (J + 14) / 16 + 1 ;
        IF I > 5
        THEN DO ;
               WRITE FILE (OBJ2) FROM (QFIX) ;
               I = 1 ;
             END ;
      END ;
IF I ¬= 1
THEN WRITE FILE (OBJ2) FROM (QFIX) ;
OPEN FILE (OBJ1) INPUT TITLE ('1') ;
/* COPIA QUADRUPLAS DE OBJ1 PARA OBJ2 */
READ FILE (OBJ1) SET (P) ;
DO WHILE (¬EOF) ;
  WRITE FILE (OBJ2) FROM (QCHAR) ;
  READ FILE (OBJ1) SET (P) ;
END ;
CLOSE FILE (OBJ1) ;
CLOSE FILE (OBJ2) ;
IF TRACE(1)
THEN PUT FILE (SPRINT) SKIP EDIT
     ('PLAPP31 (GERA PROGRAMA OBJETO EM DISCO) RETORNOU') (A) ;
END PLAPP31 ;
```

```
/* PLAPP32 - EXECUTA PROGRAMA COMPILADO                              */
PLAPP32 : PROC ;
/*********************************************************************/
/*                                                                   */
/*  ESTA PROCEDURE CARREGA E EXECUTA O PROGRAMA COMPILADO, CON-      */
/*  TIDO NO ARQUIVO "OBJ2".                                          */
/*                                                                   */
/*********************************************************************/
1 /* IMPRIME MENSAGENS DE EXECUCAO */
$CONTINUE WITH PLAMP33 RETURN
1 /* MACRO TRACE */
$CONTINUE WITH PLAMV05 RETURN
1 /* MACRO GERA E GERAEND */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO EXEC E LIST */
$CONTINUE WITH PLAMV12 RETURN
1 /* MACRO OBJ2 */
$CONTINUE WITH PLAMV14 RETURN
1 /* MACRO CODOPER */
$CONTINUE WITH PLAMV15 RETURN
1 /* MACRO REGAREA */
$CONTINUE WITH PLAMV16 RETURN
1 /* MACRO SCARDS E CARD */
$CONTINUE WITH PLAMV18 RETURN
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL QFIX(5,4)                  BIN FIXED(31,0) BASED(P) ;
 DCL QFIXED(20)                 BIN FIXED(31,0) BASED(P) ;
 DCL QFLOAT(20)                 BIN FLOAT(21) BASED(P) ;
 DCL QCHAR(80)                  CHAR(1) BASED(P) ;
 DCL 1 QUADRA(5) BASED(P) ,
       2 COP                        BIT(8) ,
       2 LIXO                       CHAR(3) ,
       2 OPND(3)                    BIN FIXED(31,0) ;
 DCL P                          POINTER ;
 DCL LC1                        BIN FIXED(15,0) INIT(1) ;
 DCL LC                         BIN FIXED(15,0) ;
 DCL FIM                        BIT(1) INIT('0'B) ;
 DCL (I , J , K , N)            BIN FIXED(15,0) ;
 DCL AUX                        BIN FIXED(31,0) ;
 DCL (ENDLC , ENDLI , TIPB)     BIN FIXED(15,0) ;
 DCL PT                         BIN FIXED(15,0) ;
 DCL COL                        BIN FIXED(15,0) STATIC INIT(1) ;
 DCL DIRETO                     BIT (1) STATIC ;
 DCL ESP                        BIN FIXED(15,0) STATIC ;
 DCL BJ                         BIN FIXED(31,0) STATIC ;
 DCL BJ1                        BIN FIXED(31,0) STATIC ;
 DCL BJ2                        BIN FIXED(31,0) STATIC ;
 DCL BJ3                        BIN FIXED(31,0) STATIC ;
 DCL CINT1                      BIN FIXED(15,0) ;
 DCL CREAL1                     BIN FIXED(15,0) ;
 DCL CBOOL1                     BIN FIXED(15,0) ;
 DCL CCHAR1                     BIN FIXED(15,0) ;
 DCL NCINT1                     BIN FIXED(15,0) ;
 DCL NCREAL1                    BIN FIXED(15,0) ;
 DCL NCBOOL1                    BIN FIXED(15,0) ;
 DCL NCCHAR1                    BIN FIXED(15,0) ;
 DCL NQUAD1                     BIN FIXED(15,0) ;
 DCL R(3)                       BIN FIXED(15,0) ;
 DCL CC                         BIT(4) ;
```

```
DCL CCODE(15)                  BIT(4) INIT
                                    ('0001' , '0010' , '0011' ,
                                     '0100' , '0101' , '0110' ,
                                     '0111' , '1000' , '1001' ,
                                     '1010' , '1011' , '1100' ,
                                     '1101' , '1110' , '1111');
DCL CBGT                       BIT(4) DEFINED CCODE POS(5) ;
DCL CBLT                       BIT(4) DEFINED CCODE POS(13) ;
DCL CBNEQ                      BIT(4) DEFINED CCODE POS(21) ;
DCL CBEQ                       BIT(4) DEFINED CCODE POS(29) ;
DCL CBGE                       BIT(4) DEFINED CCODE POS(37) ;
DCL CBLE                       BIT(4) DEFINED CCODE POS(45) ;
DCL (AUX2,AUX3,AUX4)           BIN FIXED(31,0) ;
DCL RE                         BIN FIXED(31,0) ;
DCL (FALSE,TRUE)               CHAR(1) ;
DCL BIT8                       BIT(8) BASED (PTCHAR) ;
DCL PTCHAR                     POINTER ;
DCL MAXMANT                    BIN FIXED(31,0) ;
DCL EXPMAX                     BIN FIXED(15,0) ;
DCL EXPMIN                     BIN FIXED(15,0) ;
DCL EXPNEG                     BIT(1) ;
DCL MAXINT                     BIN FIXED(31,0) ;
DCL LIMMANT                    BIN FIXED(31,0) ;
DCL EXPLIDO                    BIN FIXED(31,0) ;
DCL PTR                        BIN FIXED(15,0) STATIC INIT(81) ;
DCL CAR                        CHAR(1) ;
DCL CLASSE                     BIN FIXED(15,0) ;
DCL BOOL(4)                    CHAR(1) ;
DCL BOOL4                      CHAR(4) DEF BOOL POS (1) ;
DCL NINT                       BIN FIXED(31,0) ;
DCL NREAL                      BIN FLOAT(21) ;
DCL DIG                        PIC '9' ;
DCL MENOS                      BIT(1) ;
DCL EXP                        BIN FIXED(31,0) ;
DCL ADDR                       BUILTIN ;
DCL INDEX                      BUILTIN ;
DCL NULL                       BUILTIN ;
ON ENDFILE (OBJ2)
BEGIN ;
  CALL PLAPP33(10) ; /* PRGM OBJETO INCOMPLETO */
  GOTO ACABA ;
END ;
IF TRACE(1)
THEN PUT FILE(SPRINT) SKIP EDIT
        ('PLAPP32 (EXECUTA PRGM COMPILADO) INICIOU') (A) ;
OPEN FILE(OBJ2) INPUT TITLE ('2') ;
PTCHAR = ADDR(FALSE) ;
BIT8 = '00000000'B ;
PTCHAR = ADDR(TRUE) ;
BIT8 = '00000001'B ;
EXPMAX = 69 ;
EXPMIN = -75 ;
LIMMANT = 723701 ;
MAXINT = 2147483647 ;
MAXMANT = 7237008 ;
PUT FILE(SPRINT) PAGE ;
READ FILE(OBJ2) SET(P) ;
IF CO ¬= COAREA
THEN DO ;
        /* ERRO NO PROGRAMA OBJETO - FALTA INSTRUCAO PARA ALOCA- */
```

```
        /* CAO DE AREAS PARA EXECUCAO DO PROGRAMA.                    */
        CALL PLAPP33(1) ; /* IMPRIME MENSAGEM ERRO EXECUCAO */
        STOP ;
      END ;
1BEGIN ;
  DCL AREAINT(-TINT:CINT)                BIN FIXED(31,0) ;
  DCL AREAREAL(-TREAL:CREAL)             BIN FLOAT(21) ;
  DCL AREABOOL(-TBOOL:CBOOL)             CHAR(1) ;
  DCL AREACHAR(-TCHAR:CCHAR)             CHAR(1) ;
  DCL CHAR80                             CHAR(80) BASED (P) ;
  DCL OPCODE                             BIN FIXED(15,0) ;
  DCL AINST(5000,4)                      BIN FIXED(31,0) BASED(Q) ;
  DCL A(1000)                            CHAR(80) BASED(Q) ;
  DCL B(5000)                            BIN FIXED(31,0) BASED (Q) ;
  DCL 1 QUADRA1(NQUAD) UNAL ,
        2 COP1                               BIT(8) ,
        2 BASE1                              BIT(8) ,
        2 OPN1                               BIN FIXED(31,0) ,
        2 BASE2                              BIT(8) ,
        2 OPN2                               BIN FIXED(31,0) ,
        2 BASE3                              BIT(8) ,
        2 OPN3                               BIN FIXED(31,0) ;
  DCL NMAX                               BIN FIXED(15,0) ;
  DCL 1 BB BASED(BPOINT) ,
        2 BPTR                               POINTER ,
        2 NMAX1                              BIN FIXED(15,0) ,
        2 BASE(0:NMAX REFER (NMAX1) , 8)     BIN FIXED(15,0) ;
  DCL BPOINT                                 POINTER ;
  DCL PAUX                                   POINTER ;
  DCL Q                                  POINTER ;
  DCL MOD                                BUILTIN ;
  Q = ADDR(QUADRA1) ;
  DCL CODIGO(LINF:LSUP+1) AUTOMATIC LABEL INIT((LSUP-LINF+2) LERRO);
  DCL GETCAR                        ENTRY ;
  DCL INTEIRO                       ENTRY ;
  DCL REAL                          ENTRY ;
  DCL CEND                          ENTRY(BIN FIXED(15,0)) ;
  DCL CEND1                         ENTRY ;
  ON ERROR
  BEGIN ;
    PUT FILE(SPRINT) PAGE EDIT
       ('R1 = ' , R(1) , 'R2 = ' , R(2) , 'R3 = ' , R(3) , 'LC = ',
         LC , 'AREA ESTATICA' , (I , AREAINT(I) DO I=1 TO NCINT1))
       ((4)(A,F(11)),SKIP(2),A,(NCINT)(COL(1),(8)(F(11)))) ;
    GOTO ACABA ;
  END ;
  CINT1 = CINT ;
  CREAL1 = CREAL ;
  CBOOL1 = CBOOL ;
  CCHAR1 = CCHAR ;
  NCINT1 = NCINT ;
  NCREAL1 = NCREAL ;
  NCBOOL1 = NCBOOL ;
  NCCHAR1 = NCCHAR ;
  NQUAD1 = NQUAD ;
  NMAX = NBMAX ;
  IF NQUAD1 > 5000
  THEN DO ;
        /* PLAPP32 - RESTRICAO DE IMPLEMENTACAO - NUMERO MAXIMO  */
        /* PREVISTO DE QUADRUPLAS (5000) FOI ULTRAPASSADO.       */
```

```
        CALL PLAPP33(2) ; /* IMPRIME MENSAGEM DE ERRO */
        STOP ; /* CORRECAO - AUMENTAR NUMERO DE ELEMENTOS DOS VE-*/
      END ;    /* TORES "A" E "B" ACIMA.                          */
CODIGO(COWRITE) = LWRITE ;
CODIGO(COWRITELN) = LWRITE ;
CODIGO(COWRITEPG) = LWRITE ;
CODIGO(COWRITED) = LWRITE ;
CODIGO(COWRITELND) = LWRITE ;
CODIGO(COWRITEPGD) = LWRITE ;
CODIGO(COEXIT) = LEXIT ;
CODIGO(COMSG) = LMSG ;
CODIGO(COSET) = LSET ;
CODIGO(COSETB) = LSETB ;
CODIGO(COSTOB) = LSTOB ;
CODIGO(COEMPB) = LEMPB ;
CODIGO(CODESB) = LDESB ;
CODIGO(COATRIB) = LATRIB ;
CODIGO(COCVT) = LCVT ;
CODIGO(COADREG) = LADREG ;
CODIGO(COCREG) = LCREG ;
CODIGO(CODESV) = LDESV ;
CODIGO(COZEREG) = LZEREG ;
CODIGO(COADI) = LADI ;
CODIGO(COCDE) = LCDE ;
CODIGO(COSUBI) = LSUBI ;
CODIGO(COMLTI) = LMLTI ;
CODIGO(CODIVI) = LDIVI ;
CODIGO(COMOD) = LMOD ;
CODIGO(COSETM) = LSETM ;
CODIGO(COADR) = LADR ;
CODIGO(COSUBR) = LSUBR ;
CODIGO(CODIVR) = LDIVR ;
CODIGO(COMLTR) = LMLTR ;
CODIGO(CONOT) = LNOT ;
CODIGO(COCOMP) = LCOMP ;
CODIGO(COAND) = LAND ;
CODIGO(COOR) = LOR ;
CODIGO(COREAD) = LREAD ;
CODIGO(COREADP) = LREAD ;
CODIGO(COREADD) = LREAD ;
CODIGO(COREADPD) = LREAD ;
/* ALOCA MATRIZ DE BASES */
ALLOCATE BB ;
BPTR = NULL ;
DO I = 1 TO 8 ;
  BASE(0,I) = 0 ;
END ;
I = 5 ;
/* SAO SEMPRE GERADAS AS CONSTANTES INTEIRAS 0 E 1 */
IF COP(I) ¬= COCINT
THEN DO ;
        /* ERRO NO PROGRAMA OBJETO - FALTA INSTRUCAO PARA */
        /* GERACAO DE CTES INTEIRAS PARA EXECUCAO DO PGM. */
        CALL PLAPP33(3) ;
        STOP ;
      END ;
K = OPND(I,1) ; /* NUMERO DE CTES INTEIRAS */
J = I * 4 - 1 ; /* POS.PRIMEIRA CTE INTEIRA NA QUADRUPLA */
DO K = 1 TO K ;
  IF J > 20
```

```
   THEN DO ;
          READ FILE(OBJ2) SET(P) ;
          J = 1 ;
        END ;
   AREAINT(K) = QFIXED(J) ;
   J = J + 1 ;
END ;
I = (J + 2) / 4 + 1 ;
IF I > 5
THEN DO ;
       READ FILE(OBJ2) SET(P) ;
       I = 1 ;
     END ;
IF NCREAL1 > 0
THEN DO ; /* FORAM GERADAS CTES REAIS */
       IF COP(I) ¬= COCREAL
       THEN DO ;
              /* ERRO NO PROGRAMA OBJETO - FALTA INSTRUCAO PARA */
              /* GERACAO DE CTES REAIS PARA EXECUCAO DO PGM.    */
              CALL PLAPP33(4) ;
              STOP ;
            END ;
       K = OPND(I,1) ; /* NUMERO DE CTES REAIS */
       J = I * 4 - 1 ; /* POS.PRIMEIRA CTE REAL NA QUADRUPLA */
       DO K = 1 TO K ;
         IF J > 20
         THEN DO ;
                READ FILE(OBJ2) SET(P) ;
                J = 1 ;
              END ;
         AREAREAL(K) = QFLOAT(J) ;
         J = J + 1 ;
       END ;
       I = (J + 2) / 4 + 1 ;
       IF I > 5
       THEN DO ;
              READ FILE(OBJ2) SET(P) ;
              I = 1 ;
            END ;
     END ;
IF COP(I) ¬= COCBOOL
THEN DO ; /* SAO SEMPRE GERADAS 2 CTES BOOLEANAS (TRUE E FALSE) */
       /* ERRO NO PROGRAMA OBJETO - FALTA INSTRUCAO PARA GERA-  */
       /* CAO DE CTES BOOLEANAS PARA EXECUCAO DO PROGRAMA.      */
       CALL PLAPP33(5) ;
       STOP ;
     END ;
J = I * 16 - 4 ;
AREABOOL(1) = QCHAR(J) ;
AREABOOL(2) = QCHAR(J+4) ;
I = I + 1 ;
IF I > 5
THEN DO ;
       READ FILE(OBJ2) SET(P) ;
       I = 1 ;
     END ;
IF NCCHAR1 > 0
THEN DO ; /* FORAM GERADAS CTES ALFANUMERICAS */
       IF COP(I) ¬= COCCHAR
       THEN DO ;
```

```
            /* ERRO NO PROGRAMA OBJETO - FALTA INSTRUCAO PARA */
            /* GERACAO DE CTES ALFANUMERICAS PARA EXECUCAO DO */
            /* PROGRAMA.                                      */
            CALL PLAPP33(6) ;
            STOP ;
          END ;
        K = OPND(I,1) ; /* NUMERO DE CARACTERES DAS CTES ALFANUM.*/
        J = I * 16 - 7 ; /* POS.PRIMEIRA CTE ALFA.NA QUADRUPLA  */
        DO K = 1 TO K ;
          IF J > 80
          THEN DO ;
                 READ FILE(OBJ2) SET(P) ;
                 J = 1 ;
               END ;
          AREACHAR(K) = QCHAR(J) ;
          J = J + 1 ;
        END ;
        I = (J + 14) / 16 + 1 ;
        IF I > 5
        THEN DO ;
               READ FILE (OBJ2) SET (P) ;
               I = 1 ;
             END ;
      END ;
IF I ¬= 1
THEN READ FILE (OBJ2) SET (P) ;
IF NQUAD1 >= 5
THEN DO ;
       DO K = 1 TO NQUAD1 / 5 - 1 ;
         A(K) = CHAR80 ;
         READ FILE (OBJ2) SET (P) ;
       END ;
       A(K) = CHAR80 ;
       J = K * 5 + 1 ;
       IF J <= NQUAD1
       THEN READ FILE (OBJ2) SET (P) ;
     END ;
ELSE J = 1 ;
IF J <= NQUAD1
THEN DO ;
       I = 1 ;
       DO J = J TO NQUAD1 ;
         DO K = 1 TO 4 ;
           AINST(J,K) = QFIX(I,K) ;
         END ;
         I = I + 1 ;
       END ;
     END ;
/* NESTE PONTO FOI COMPLETADA A CARGA DO PROGRAMA OBJETO */
IF LIST
THEN DO ;
       IF NCINT1 > 0
       THEN PUT FILE(SPRINT) SKIP(2) EDIT
            ('CTES INTEIRAS',(I,AREAINT(I) DO I=1 TO NCINT1))
            (A,(NCINT1)(COL(1),(4)(F(11),F(11)))) ;
       IF NCREAL1 > 0
       THEN PUT FILE(SPRINT) SKIP(2) EDIT
            ('CTES REAIS',(I,AREAREAL(I) DO I=1 TO NCREAL1))
            (A,(NCREAL1)(COL(1),(4)(F(7),F(15,7)))) ;
       DCL 1 XXX UNAL ;
```

```
          2 ZEROS                CHAR(1),
          2 BOOLX                CHAR(1) ;
      DCL BOOLY                BIN FIXED(15,0) BASED(PX) ;
      DCL PX                   POINTER ;
      PX = ADDR(XXX) ;
      BOOLY = 0 ;
      PUT FILE(SPRINT) SKIP(2) EDIT('CTES BOOLEANAS')(A) ;
      DO I = 1 TO NCBOOL1 ;
        BOOLX = AREABOOL(I) ;
        PUT FILE(SPRINT) SKIP EDIT(I , BOOLY)(F(3) , F(11)) ;
      END ;
      IF NCCHAR1 > 0
      THEN PUT FILE(SPRINT) SKIP(2) EDIT
              ('CTES CHAR','"',(AREACHAR(I) DO I=1 TO NCCHAR1),
               '"')(A,COL(1),A,(NCCHAR1)(A),A) ;
      PUT FILE(SPRINT) SKIP(2) EDIT
          ((I , VETCOD(COP1(I)) , COP1(I) , BASE1(I) , OPN1(I) ,
           BASE2(I) , OPN2(I) , BASE3(I) , OPN3(I) DO I=1 TO
          NQUAD1))((NQUAD1)(COL(1),F(3),COL(7),A,COL(12),F(3),
          (3)(F(4),F(13)))) ;
      PUT FILE(SPRINT) SKIP(2) ;
    END ;
FIM = ¬EXEC ;
IF EXEC
THEN DO ;
       PUT FILE(SPRINT) SKIP EDIT('INICIO DA EXECUCAO')(A) ;
       PUT FILE(SPRINT) PAGE ;
     END ;
DO WHILE(¬FIM) ;
  OPCODE = COP1(LC1) ; /* PEGA CODIGO DA INSTRUCAO */
  IF OPCODE < LINF | OPCODE > LSUP
  THEN OPCODE = LSUP + 1 ;
  LC = LC1 ;
  LC1 = LC1 + 1 ;
  GOTO CODIGO(OPCODE) ;
LERRO : /* CODIGO DE OPERACAO INVALIDO */
  CALL PLAPP33(7) ;
  FIM = '1'B ;
  GOTO FIMCASE ;
LEXIT : /* FIM NORMAL DO PROCESSAMENTO */
  CALL PLAPP33(8) ;
  FIM = '1'B ;
  GOTO FIMCASE ;
LMSG : /* INSTRUCAO P/IMPRIMIR MENSAGENS DURANTE EXECUCAO */
  CALL PLAPP33(OPN3(LC)) ;
  GOTO FIMCASE ;
LREAD : /* INSTRUCOES DE LEITURA */
  IF OPCODE = COREADP | OPCODE = COREADPD
  THEN IF PTR ¬= 1
       THEN PTR = 81 ;
  DIRETO = OPCODE=COREADD | OPCODE=COREADPD ;
  J = OPN2(LC) ; /* NUMERO DE OPERANDOS */
  PT = OPN3(LC) ; /* PONTEIRO P/ LISTA DE OPERANDOS */
  DO J = 1 TO J ;
    BJ = AREAINT(PT) ; /* TIPO DO ENDERECO DO OPERANDO */
    BJ2 = AREAINT(PT+1) ; /* ENDERECO DO OPERANDO */
    BJ3 = AREAINT(PT+2) ; /* NUM.DO REG.BASE */
    IF BJ = IVI | BJ = RVI | BJ = BVI | BJ = CVI
    THEN DO ; /* INDIRETO */
           IF BJ2 < 0
```

```
        THEN TIPB = 5 ; /* TEMPORARIO */
        ELSE TIPB = 1 ; /* VARIAVEL */
        BJ2 = AREAINT(BASE(BJ3,TIPB) + BJ2) ;
      END ;
ELSE DO ;
        IF BJ = IVD
        THEN TIPB = 1 ;
        ELSE IF BJ = RVD
             THEN TIPB = 2 ;
             ELSE IF BJ = BVD
                  THEN TIPB = 3 ;
                  ELSE TIPB = 4 ;
        BJ2 = BASE(BJ3,TIPB) + BJ2 ;
      END ;
IF ¬DIRETO
THEN DO ;
        PTR = (PTR-2) / 10 ;
        PTR = PTR * 10 + 11 ;
      END ;
IF BJ = CVD | BJ = CVI
THEN DO ; /* CHAR */
        CALL GETCAR ;
        AREACHAR(BJ2) = CAR ;
      END ;
ELSE DO ;
        IF DIRETO
        THEN DO ;
               CAR = ' ' ;
               DO WHILE (CAR = ' ') ;
                 CALL GETCAR ;
               END ;
             END ;
        IF BJ = BVD | BJ = BVI
        THEN DO ;
               BOOL(1) = CAR ;
               CALL GETCAR ;
               IF CAR ¬= ' ' & CAR ¬= ','
               THEN DO ;
                      BOOL(2) = CAR ;
                      CALL GETCAR ;
                      BOOL(3) = CAR ;
                      CALL GETCAR ;
                      BOOL(4) = CAR ;
                      IF BOOL4 = 'FALS'
                      THEN DO ;
                             CALL GETCAR ;
                             IF CAR = 'E'
                             THEN AREABOOL(BJ2) = FALSE ;
                             ELSE DO ;
                                    /* DADO BOOLEANO INVALIDO*/
                                    /* EM COMANDO DE LEITURA */
                                    CALL PLAPP33(11) ;
                                    GOTO ACABA ;
                                  END ;
                           END ;
                      ELSE IF BOOL4 = 'TRUE'
                           THEN AREABOOL(BJ2) = TRUE ;
                           ELSE DO ;
                                  /* DADO BOOLEANO INVALIDO */
                                  /* EM COMANDO DE LEITURA. */
```

```
                                        CALL PLAPP33(11) ;
                                        GOTO ACABA ;
                                      END ;
                            END ;
                        ELSE DO ;
                               IF BOOL(1) = 'F'
                               THEN AREABOOL(BJ2) = FALSE ;
                               ELSE IF BOOL(1) = 'T'
                                    THEN AREABOOL(BJ2) = TRUE ;
                                    ELSE DO ;
                                           /* DADO BOOLEANO INVALIDO */
                                           /* EM COMANDO DE LEITURA  */
                                           CALL PLAPP33(11) ;
                                           GOTO ACABA ;
                                         END ;
                               PTR = PTR + 1 ;
                             END ;
                      END ;
                ELSE DO ; /* OPERANDO INVALIDO */
                       IF BJ = IVD | BJ = IVI
                       THEN CALL INTEIRO ;
                       ELSE CALL REAL ;
                       IF FIM
                       THEN GOTO ACABA ;
                       PTR = PTR + 1 ; /* PULA SEPARADOR */
                     END ;
            END ;
      PT = PT + 3 ;
    END ;
    GOTO FIMCASE ;
LWRITE : /* INSTRUCOES DE IMPRESSAO */
    IF OPCODE = COWRITEPG | OPCODE = COWRITEPGD
    THEN DO ;
           PUT FILE(SPRINT) PAGE ;
           COL = 1 ;
         END ;
    ELSE IF OPCODE = COWRITELN | OPCODE = COWRITELND
         THEN DO ;
                PUT FILE(SPRINT) SKIP ;
                COL = 1 ;
              END ;
    DIRETO= OPCODE=COWRITED | OPCODE=COWRITELND | OPCODE=COWRITEPGD;
    IF DIRETO
    THEN ESP = 0 ;
    J = OPN2(LC) ; /* NUMERO DE OPERANDOS */
    PT = OPN3(LC) ; /* PONTEIRO P/ LISTA DOS OPERANDOS */
    DO J = 1 TO J ;
      IF ¬DIRETO
      THEN IF COL > 90
           THEN DO ;
                  PUT FILE(SPRINT) SKIP ;
                  COL = 1 ;
                  ESP = 0 ;
                END ;
           ELSE IF COL > 60
                THEN DO ;
                       ESP = 90 - COL ;
                       COL = 90 ;
                     END ;
                ELSE IF COL > 30
```

```
                    THEN DO ;
                           ESP = 60 - COL ;
                           COL = 60 ;
                         END ;
                    ELSE IF COL > 1
                         THEN DO ;
                                ESP = 30 - COL ;
                                COL = 30 ;
                              END ;
                         ELSE ESP = 0 ;
BJ = AREAINT(PT) ; /* TIPO DO ENDERECO DO OPERANDO */
BJ2 = AREAINT(PT+2) ; /* ENDERECO DO OPERANDO */
BJ1 = AREAINT(PT+1) ; /* TAMANHO DO OPERANDO */
BJ3 = AREAINT(PT+3) ; /* NUM.DO REG.BASE */
IF BJ = IVI | BJ = RVI | BJ = BVI | BJ = CVI
THEN DO ; /* ENDERECO INDIRETO */
       IF BJ2 < 0
       THEN TIPB = 5 ; /* TEMPORARIO */
       ELSE TIPB = 1 ; /* VARIAVEL */
       BJ2 = AREAINT(BASE(BJ3,TIPB) + BJ2) ;
     END ;
ELSE IF BJ ¬= ICT & BJ ¬= RCT & BJ ¬= BCT & BJ ¬= CCT
     THEN DO ;
            IF BJ = IVD
            THEN TIPB = 1 ;
            ELSE IF BJ = RVD
                 THEN TIPB = 2 ;
                 ELSE IF BJ = BVD
                      THEN TIPB = 3 ;
                      ELSE IF BJ = CVD
                           THEN TIPB = 4 ;
                           ELSE IF BJ = ITE
                                THEN TIPB = 5 ;
                                ELSE IF BJ = RTE
                                     THEN TIPB = 6 ;
                                     ELSE TIPB = 7 ;
            BJ2 = BASE(BJ3,TIPB) + BJ2 ;
          END ;
IF BJ = ITE | BJ = ICT | BJ = IVD | BJ = IVI
THEN DO ; /* INTEIRO */
      PUT FILE(SPRINT) EDIT(AREAINT(BJ2)) (X(ESP) , F(10)) ;
      COL = COL + 10 ;
     END ;
ELSE IF BJ = RTE | BJ = RCT | BJ = RVD | BJ = RVI
     THEN DO ; /* REAL */
            PUT FILE(SPRINT) EDIT
                (AREAREAL(BJ2)) (X(ESP) , E(15,7)) ;
            COL = COL + 15 ;
          END ;
     ELSE IF BJ = CCT | BJ = CVD | BJ = CVI
          THEN DO ; /* CHAR */
                 PUT FILE(SPRINT) EDIT
                     ((AREACHAR(I) DO I=BJ2 TO BJ2+BJ1-1))
                     (X(ESP) , (BJ1) A) ;
                 COL = COL + BJ1 ;
               END ;
          ELSE       /* BOOLEANOS */
               IF AREABOOL(BJ2) = TRUE
               THEN DO ; /* TRUE */
                      PUT FILE(SPRINT) EDIT('TRUE')(X(ESP),A);
```

```
                        COL = COL + 4 ;
                      END ;
                    ELSE DO ; /* FALSE */
                        PUT FILE(SPRINT) EDIT('FALSE')(X(ESP),A);
                        COL = COL + 5 ;
                      END ;
      IF COL > 120
      THEN COL = COL - 120 ;
      PT = PT + 4 ;
    END ; /* DO J = 1 TO J */
    GOTO FIMCASE ;
LSET : /* INSTRUCAO PARA ATRIBUIR UM VALOR A UM INDEXADOR */
    IF OPN3(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    R(OPN2(LC)) = AREAINT(BASE(BASE3(LC),TIPB) + OPN3(LC)) ;
    GOTO FIMCASE ;
LSETB : /* INSTRUCAO P/ATRIBUIR UM VALOR A UM REGISTRADOR BASE */
    IF OPN3(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    BASE(OPN2(LC),OPN1(LC))=AREAINT(BASE(BASE3(LC),TIPB)+OPN3(LC));
    GOTO FIMCASE ;
LSTOB : /* INSTRUCAO P/COPIAR P/UMA POSICAO DE MEMORIA (INTEIRA) */
        /* O CONTEUDO DE UM REGISTRADOR BASE                    */
    IF OPN3(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    AREAINT(BASE(BASE3(LC),TIPB)+OPN3(LC))=BASE(OPN2(LC),OPN1(LC));
    GOTO FIMCASE ;
LEMPB : /* INSTRUCAO P/EMPILHAR UMA NOVA MATRIZ DE BASES,        */
        /* COPIANDO PARTE DOS REGISTRADORES                     */
    PAUX = BPOINT ;
    IF OPN3(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    TIPB = AREAINT(BASE(BASE3(LC),TIPB) + OPN3(LC)) ;
    ALLOCATE BB ;
    DO I = 0 TO TIPB ; /* COPIA BASES */
      DO J = 1 TO 8 ;
        BASE(I,J) = PAUX -> BASE(I,J) ;
      END ;
    END ;
    BPTR = PAUX ;
    GOTO FIMCASE ;
LDESB : /* INSTRUCAO P/DESEMPILHAR MATRIZ DE BASES */
    PAUX = BPTR ;
    FREE BB ;
    BPOINT = PAUX ;
    GOTO FIMCASE ;
LSETM : /* INSTRUCAO P/ATRIBUIR VALORES AOS 3 REGISTRADORES */
    IF OPN1(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    R(1) = AREAINT(BASE(BASE1(LC),TIPB) + OPN1(LC)) ;
    IF OPN2(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    R(2) = AREAINT(BASE(BASE2(LC),TIPB) + OPN2(LC)) ;
    IF OPN3(LC) < 0
```

```
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    R(3) = AREAINT(BASE(BASE3(LC),TIPB) + OPN3(LC)) ;
    GOTO FIMCASE ;
LCVT : /* INSTRUCAO DE ATRIBUICAO COM CONVERSAO DE TIPO */
    IF OPN1(LC) = 1 /* CONVERSAO P/INTEIRO ? */
    THEN DO ;
           AUX3 = 2 ; /* OPERANDO REAL */
           AUX4 = 1 ; /* OPERANDO INTEIRO */
         END ;
    ELSE DO ; /* CONVERSAO PARA REAL */
           AUX3 = 1 ; /* OPERANDO INTEIRO */
           AUX4 = 2 ; /* OPERANDO REAL */
         END ;
    GOTO SEGUE ;
LATRIB : /* INSTRUCAO PARA ATRIBUICAO DIRETA */
    AUX3 , AUX4 = OPN1(LC) ;
SEGUE :
    AUX2 = OPN1(LC) ;
    IF OPN2(LC) < 0
    THEN AUX3 = AUX3 + 4 ;
    IF OPN3(LC) < 0
    THEN AUX4 = AUX4 + 4 ;
    AUX3 = BASE(BASE2(LC),AUX3) + OPN2(LC) + R(2) ;
    AUX4 = BASE(BASE3(LC),AUX4) + OPN3(LC) + R(3) ;
    IF AUX2 = 1  /* ATRIBUICAO OU CONVERSAO INTEIRA */
    THEN IF OPCODE = COCVT /* SE E' CONVERSAO */
         THEN AREAINT(AUX4) = AREAREAL(AUX3) ;
         ELSE AREAINT(AUX4) = AREAINT(AUX3) ;
    ELSE IF AUX2 = 2 /* ATRIBUICAO OU CONVERSAO REAL */
         THEN IF OPCODE = COCVT /* SE E' CONVERSAO */
              THEN AREAREAL(AUX4) = AREAINT(AUX3) ;
              ELSE AREAREAL(AUX4) = AREAREAL(AUX3) ;
         ELSE IF AUX2 = 3 /* OPERANDOS BOOLEANOS */
              THEN AREABOOL(AUX4) = AREABOOL(AUX3) ;
              ELSE AREACHAR(AUX4) = AREACHAR(AUX3) ; /* OP. CHAR */
    GOTO FIMCASE ;
LADREG : /* INSTRUCAO PARA INCREMENTAR UM REG.INDEXADOR */
    IF OPN3(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    R(OPN2(LC)) = R(OPN2(LC)) + AREAINT(BASE(BASE3(LC),TIPB) +
                  OPN3(LC)) ;
    GOTO FIMCASE ;
LCREG : /* INSTRUCAO PARA COMPARACAO DO VALOR DE UM REGISTRADOR */
    IF OPN3(LC) < 0
    THEN TIPB = 5 ;
    ELSE TIPB = 1 ;
    AUX4 = AREAINT(BASE(BASE3(LC),TIPB) + OPN3(LC)) ;
    RE = R(OPN2(LC)) ;
    IF RE > AUX4
    THEN CC = CBGT ;
    ELSE IF RE < AUX4
         THEN CC = CBLT ;
         ELSE CC = CBEQ ;
    GOTO FIMCASE ;
LCOMP : /* INSTRUCAO P/ COMPARACAO */
    AUX2 , AUX3 , AUX4 = OPN1(LC) ;
    IF OPN2(LC) < 0
    THEN AUX3 = AUX3 + 4 ;
```

```
    IF OPN3(LC) < 0
    THEN AUX4 = AUX4 + 4 ;
    AUX3 = BASE(BASE2(LC),AUX3) + OPN2(LC) + R(2) ;
    AUX4 = BASE(BASE3(LC),AUX4) + OPN3(LC) + R(3) ;
    IF AUX2 = 1 /* COMPARACAO INTEIRA? */
    THEN IF AREAINT(AUX3) > AREAINT(AUX4)
         THEN CC = CBGT ;
         ELSE IF AREAINT(AUX3) < AREAINT(AUX4)
              THEN CC = CBLT ;
              ELSE CC = CBEQ ;
    ELSE IF AUX2 = 2 /* COMPARACAO REAL? */
         THEN IF AREAREAL(AUX3) > AREAREAL(AUX4)
              THEN CC = CBGT ;
              ELSE IF AREAREAL(AUX3) < AREAREAL(AUX4)
                   THEN CC = CBLT ;
                   ELSE CC = CBEQ ;
         ELSE IF AUX2 = 3 /* COMPARACAO BOOLEANA? */
              THEN IF AREABOOL(AUX3) > AREABOOL(AUX4)
                   THEN CC = CBGT ;
                   ELSE IF AREABOOL(AUX3) < AREABOOL(AUX4)
                        THEN CC = CBLT ;
                        ELSE CC = CBEQ ;
              ELSE IF AREACHAR(AUX3) > AREACHAR(AUX4)
                   THEN CC = CBGT ;
                   ELSE IF AREACHAR(AUX3) < AREACHAR(AUX4)
                        THEN CC = CBLT ;
                        ELSE CC = CBEQ ;
    GOTO FIMCASE ;
LDESV : /* INSTRUCAO DE DESVIO CONDICIONAL */
    IF (CCODE(OPN2(LC)) & CC) = CC
    THEN DO ;
           LC1 = OPN3(LC) ;
           IF OPN1(LC) ¬= 0 /* INDEXADO? */
           THEN LC1 = LC1 + R(3) ;
         END ;
    GOTO FIMCASE ;
LZEREG : /* INSTRUCAO PARA ZERAR REGISTRADORES */
    IF OPN1(LC) = 1
    THEN R(1) = 0 ;
    IF OPN2(LC) = 1
    THEN R(2) = 0 ;
    IF OPN3(LC) = 1
    THEN R(3) = 0 ;
    GOTO FIMCASE ;
LADI : /* INSTRUCAO PARA ADICAO INTEIRA */
    CALL CEND(1) ;
    AREAINT(AUX4) = AREAINT(AUX2) + AREAINT(AUX3) ;
    GOTO FIMCASE ;
LADR : /* INSTRUCAO PARA ADICAO REAL */
    CALL CEND(2) ;
    AREAREAL(AUX4) = AREAREAL(AUX2) + AREAREAL(AUX3) ;
    GOTO FIMCASE ;
LSUBI : /* INSTRUCAO P/SUBTRACAO INTEIRA */
    CALL CEND(1) ;
    AREAINT(AUX4) = AREAINT(AUX2) - AREAINT(AUX3) ;
    GOTO FIMCASE ;
LSUBR : /* INSTRUCAO P/SUBTRACAO REAL */
    CALL CEND(2) ;
    AREAREAL(AUX4) = AREAREAL(AUX2) - AREAREAL(AUX3) ;
    GOTO FIMCASE ;
```

```
LDIVI : /* INSTRUCAO P/DIVISAO INTEIRA */
    CALL CEND(1) ;
    AREAINT(AUX4) = AREAINT(AUX2) / AREAINT(AUX3) ;
    GOTO FIMCASE ;
LDIVR : /* INSTRUCAO P/DIVISAO REAL */
    CALL CEND(2) ;
    AREAREAL(AUX4) = AREAREAL(AUX2) / AREAREAL(AUX3) ;
    GOTO FIMCASE ;
LMLTI : /* INSTRUCAO P/MULTIPLICACAO INTEIRA */
    CALL CEND(1) ;
    AREAINT(AUX4) = AREAINT(AUX2) * AREAINT(AUX3) ;
    GOTO FIMCASE ;
LMLTR : /* INSTRUCAO P/MULTIPLICACAO REAL */
    CALL CEND(2) ;
    AREAREAL(AUX4) = AREAREAL(AUX2) * AREAREAL(AUX3) ;
    GOTO FIMCASE ;
LMOD : /* INSTRUCAO MODULO */
    CALL CEND(1) ;
    AREAINT(AUX4) = MOD(AREAINT(AUX2) , AREAINT(AUX3)) ;
    GOTO FIMCASE ;
LCDE : /*INSTR. P/CALCULO DO DESLOCAMENTO DE ELEMENTO DE ARRAY */
    ENDLC = OPN1(LC) ; /* END. LISTA CTES DO ARRAY */
    K = AREAINT(ENDLC) ; /* NUM. DIMENSOES DO ARRAY  */
    I = OPN2(LC) ; /* END. LISTA DE END. DOS INDICES */
    J = I + K * 3 - 5 ; /* J PERCORRERA' LISTA END DOS INDICES */
                        /* DE TRAS PARA DIANTE                 */
    ENDLC = ENDLC + K * 2 ; /* ENDLC PERCORRERA' LISTA DE CTES */
                            /* DO ARRAY DE TRAS PARA DIANTE    */
    BJ = AREAINT(J+2) ; /* TIPO DO ENDERECO */
    BJ1 = AREAINT(J+3) ; /* DESLOCAMENTO */
    BJ2 = AREAINT(J+4) ; /* NUM.DO REG.BASE */
    /* CALCULA END.ABSOLUTO DO INDICE E COLOCA EM BJ1 */
    CALL CEND1 ;
    AUX = AREAINT(ENDLC) + AREAINT(BJ1) - AREAINT(ENDLC-1) ;
    DO J = J TO I BY -3 ;
      ENDLC = ENDLC - 2 ;
      BJ = AREAINT(J-1) ; /* TIPO DO ENDERECO */
      BJ1 = AREAINT(J) ; /* DESLOCAMENTO */
      BJ2 = AREAINT(J+1) ; /* NUM.DO REG.BASE */
      /* CALCULA END.ABSOLUTO DO INDICE E COLOCA EM BJ1 */
      CALL CEND1 ;
      AUX = AUX * (AREAINT(BJ1)-AREAINT(ENDLC-1)) * AREAINT(ENDLC);
    END ;
    /* COLOCA DESLOCAMENTO CALCULADO NO TEMPORARIO */
    AREAINT(BASE(BASE3(LC),5)+OPN3(LC)) = AUX - 1 ;
    GOTO FIMCASE ;
LNOT : /* INSTRUCAO NOT */
    AUX3 , AUX4 = 3 ;
    IF OPN2(LC) < 0
    THEN AUX3 = AUX3 + 4 ;
    IF OPN3(LC) < 0
    THEN AUX4 = AUX4 + 4 ;
    AUX3 = BASE(BASE2(LC),AUX3) + OPN2(LC) * R(2) ;
    AUX4 = BASE(BASE3(LC),AUX4) + OPN3(LC) ;
    IF AREABOOL(AUX3) = FALSE
    THEN AREABOOL(AUX4) = TRUE ;
    ELSE AREABOOL(AUX4) = FALSE ;
    GOTO FIMCASE ;
LAND : /* INSTRUCAO AND */
    CALL CEND(3) ;
```

```
    IF AREABOOL(AUX2) = TRUE & AREABOOL(AUX3) = TRUE
    THEN AREABOOL(AUX4) = TRUE ;
    ELSE AREABOOL(AUX4) = FALSE ;
    GOTO FIMCASE ;
-LOR : /* INSTRUCAO OR */
    CALL CEND(3) ;
    IF AREABOOL(AUX2) = TRUE | AREABOOL(AUX3) = TRUE
    THEN AREABOOL(AUX4) = TRUE ;
    ELSE AREABOOL(AUX4) = FALSE ;
    GOTO FIMCASE ;
-FIMCASE :
 END ; /* DO WHILE(¬FIM) */
1CEND : PROC(TIPB) ;
   DCL TIPB          BIN FIXED(15,0) ;
   AUX2 , AUX3 , AUX4 = TIPB ;
   IF OPN1(LC) < 0
   THEN AUX2 = AUX2 + 4 ;
   IF OPN2(LC) < 0
   THEN AUX3 = AUX3 + 4 ;
   IF OPN3(LC) < 0
   THEN AUX4 = AUX4 + 4 ;
   AUX2 = BASE(BASE1(LC),AUX2) + OPN1(LC) + R(1) ;
   AUX3 = BASE(BASE2(LC),AUX3) + OPN2(LC) + R(2) ;
   AUX4 = BASE(BASE3(LC),AUX4) + OPN3(LC) + R(3) ;
 END CEND ;
1CEND1 : PROC ;
   /* CALCULA END.ABSOLUTO DO INDICE E COLOCA EM BJ1 */
   IF BJ = IVI
   THEN DO ; /* ENDERECO INDIRETO */
          IF BJ1 < 0
          THEN TIPB = 5 ; /* TEMPORARIO */
          ELSE TIPB = 1 ; /* VARIAVEL */
          BJ1 = AREAINT(BASE(BJ2,TIPB) + BJ1) ;
        END ;
   ELSE IF BJ ¬= ICT
        THEN DO ;
               IF BJ = IVD
               THEN TIPB = 1 ;
               ELSE TIPB = 5 ;
               BJ1 = BASE(BJ2,TIPB) + BJ1 ;
             END ;
END CEND1 ;
1GETCAR : PROC ;
 IF PTR > 80
THEN DO ;
        READ FILE (SCARDS) INTO (CARD) ;
        PTR = 1 ;
      END ;
 CAR = CARD(PTR) ;
 IF CAR >= '0' & CAR <= '9'
THEN CLASSE = 6 ; /* DIGITO */
 ELSE CLASSE = INDEX(' ,+-.' , CAR) ;
 PTR = PTR + 1 ;
END GETCAR ;
1INTEIRO : PROC ;
 ON FIXEDOVERFLOW
BEGIN ;
   /* DADO INTEIRO MAIOR QUE O PERMITIDO , EM COMANDO DE LEITURA */
   CALL PLAPP33(12) ;
   FIM = '1'B ;
```

```
   CLASSE = 0 ;
 END ;
 NINT = 0 ;
 MENOS = '0'B ;
 IF CLASSE = 3 | CLASSE = 4
 THEN DO ; /* "+" OU "-" */
         MENOS = CLASSE = 4 ;
         CALL GETCAR ;
       END ;
 DO WHILE(CLASSE = 6) ; /* ENQUANTO FOR DIGITO */
   DIG = CAR ;
   CALL GETCAR ;
   NINT = NINT * 10 + DIG ;
 END ;
 IF ¬FIM
 THEN IF CLASSE ¬= 1 & CLASSE ¬= 2
      THEN DO ; /* NAO TERMINOU EM SEPARADOR */
              /* DADO INTEIRO INVALIDO EM COMANDO DE LEITURA */
              CALL PLAPP33(13) ;
              FIM = '1'B ;
            END ;
      ELSE IF MENOS
           THEN AREAINT(BJ2) = - NINT ;
           ELSE AREAINT(BJ2) = NINT ;
 END INTEIRO ;
REAL : PROC ;
 MENOS = '0'B ;
 IF CLASSE = 3 | CLASSE = 4
 THEN DO ; /* "+" OU "-" */
         MENOS = CLASSE = 4 ;
         CALL GETCAR ;
       END ;
 NINT = 0 ;
 EXP = 0 ;
 DO WHILE(CLASSE = 6) ; /* ENQUANTO FOR DIGITO */
   IF NINT < LIMMANT
   THEN DO ;
          DIG = CAR ;
          NINT = NINT * 10 + DIG ;
        END ;
   ELSE EXP = EXP + 1 ;
   CALL GETCAR ;
 END ;
 IF CLASSE ¬= 5 & CAR ¬= 'E'
 THEN DO ; /* NAO E' "." NEM "E" */
         /* DADO REAL INVALIDO EM COMANDO DE LEITURA */
         CALL PLAPP33(14) ;
         FIM = '1'B ;
       END ;
 ELSE DO ;
         IF CLASSE = 5 /* "." ? */
         THEN DO ; /* PEGA PARTE FRACIONARIA */
                CALL GETCAR ;
                IF CLASSE ¬= 6 /* SE NAO E' DIGITO */
                THEN DO ;
                       /* DADO REAL INVALIDO EM COMANDO DE LEITURA */
                       CALL PLAPP33(14) ;
                       FIM = '1'B ;
                     END ;
              ELSE DO WHILE(CLASSE = 6) ; /* ENQUANTO FOR DIGITO */
```

```
                    IF NINT < LIMMANT
                    THEN DO ;
                           DIG = CAR ;
                           NINT = NINT * 10 + DIG ;
                           EXP = EXP - 1 ;
                         END ;
                    CALL GETCAR ;
                  END ;
          END ;
      IF ¬FIM & CAR = 'E'
      THEN DO ; /* PEGA EXPOENTE */
             EXPLIDO = 0 ;
             CALL GETCAR ;
             IF CLASSE = 3 | CLASSE = 4
             THEN DO ; /* "+" OU "-" */
                    EXPNEG = CLASSE = 4 ;
                    CALL GETCAR ;
                  END ;
             IF CLASSE ¬= 6 /* SE NAO FOR DIGITO */
             THEN DO ;
                    /* DADO REAL INVALIDO EM COMANDO DE LEITURA */
                    CALL PLAPP33(14) ;
                    FIM = '1'B ;
                  END ;
             ELSE DO ;
                    DO WHILE(CLASSE = 6) ;
                      /* ENQUANTO FOR DIGITO */
                      DIG = CAR ;
                      EXPLIDO = EXPLIDO * 10 + DIG ;
                      IF EXPLIDO >= LIMMANT
                      THEN DO ;
                             /* EXPOENTE DE DADO REAL FORA DOS */
                             /* LIMITES EM COMANDO DE LEITURA  */
                             CALL PLAPP33(15) ;
                             FIM = '1'B ;
                             CLASSE = 0 ;
                           END ;
                      ELSE CALL GETCAR ;
                    END ;
                    IF ¬FIM
                    THEN IF EXPNEG
                         THEN EXP = EXP - EXPLIDO ;
                         ELSE EXP = EXP + EXPLIDO ;
                  END ;
           END ;
    END ;
IF ¬FIM
THEN IF CLASSE ¬= 1 & CLASSE ¬= 2
     THEN DO ; /* NAO TERMINOU EM SEPARADOR */
            /* DADO REAL INVALIDO EM COMANDO DE LEITURA */
            CALL PLAPP33(14) ;
            FIM = '1'B ;
          END ;
     /* AJUSTE DE EXPOENTE */
     ELSE IF EXP < EXPMIN
          THEN DO ; /* DADO REAL MENOR QUE O PERMITIDO EM */
                    /* COMANDO DE LEITURA                 */
                 CALL PLAPP33(16) ;
                 FIM = '1'B ;
               END ;
```

```
          ELSE IF EXP > EXPMAX
               THEN DO ; /* DADO REAL MAIOR QUE O PERMITIDO */
                         /* EM COMANDO DE LEITURA           */
                    CALL PLAPP33(17) ;
                    FIM = '1'B ;
                  END ;
               ELSE DO ;
                    IF NINT > MAXMANT
                    THEN DO ;
                           NINT = (NINT + 1) / 2 ;
                           NREAL = NINT * 2 ;
                         END ;
                    ELSE NREAL = NINT ;
                    IF MENOS
                    THEN NREAL = -NREAL ;
                    IF EXP ¬= 0
                    THEN NREAL = NREAL * 10 ** EXP ;
                    AREAREAL(BJ2) = NREAL ;
                  END ;
 END REAL ;
 END ; /* BEGIN */
 ACABA :
 IF TRACE(1)
 THEN PUT FILE(SPRINT) SKIP EDIT
           ('PLAPP32 (EXECUTA PRGM COMPILADO) RETORNOU')(A) ;
 END PLAPP32 ;
```

```
 /* PLAPP33 - IMPRIME MENSAGENS DE EXECUCAO.                          */
 PLAPP33 : PROC (NUM) ;
 /*********************************************************************/
 /* ESTA PROCEDURE IMPRIME MENSAGENS DE EXECUCAO.                     */
 /* RECEBE COMO PARAMETRO O CODIGO DA MENSAGEM A SER IMPRESSA.        */
 /* SAO UTILIZADAS 2 ESTRUTURAS, 'MENS' E 'COMP', PARA ARMAZENAR      */
 /* AS MENSAGENS.                                                     */
 /* SE A MENSAGEM TIVER APENAS 53 CARACTERES, O CAMPO 'SEG' DA        */
 /* ESTRUTURA 'MENS' CONTERA' 0 E SERA' IMPRESSO APENAS O CAMPO       */
 /* 'INICIO' DA MESMA ESTRUTURA.                                      */
 /* CASO CONTRARIO, O FINAL DA MENSAGEM ESTARA' NO CAMPO              */
 /* 'COMPLEM' DA ESTRUTURA 'COMP'; O CAMPO 'SEG' CORRESPONDENTE       */
 /* A PRIMEIRA PARTE DA MENSAGEM APONTARA' PARA A SEGUNDA PARTE       */
 /* DA MENSAGEM; O CAMPO 'TER' DA SEGUNDA PARTE APONTARA' PARA        */
 /* A TERCEIRA PARTE (NA ESTRUTURA 'COMP') E ASSIM POR DIANTE,        */
 /* ATE' QUE NA ULTIMA PARTE O CAMPO 'TER' CONTEM 0.                  */
 /* SAO IMPRESSAS 2 PARTES DA MENSAGEM (SE HOUVER) EM CADA LINHA.     */
 /*********************************************************************/
1 /* MACRO SPRINT */
$CONTINUE WITH PLAMV19 RETURN
1DCL NUM                    BIN FIXED(15,0) ;
 DCL  I                     BIN FIXED (15,0) STATIC ;
 DCL 1 MENS(17)             STATIC ,
       2 INICIO             CHAR (53) ,
       2 SEG                BIN FIXED (15,0) ;
 DCL 1 COMP(9)              STATIC ,
       2 COMPLEM            CHAR(53) ,
       2 TER                BIN FIXED (15,0) ;
 DCL N                      BIN FIXED(15,0) STATIC ;
 DCL FLIPFLOP               BIT(1) STATIC ;
 DCL PRIMEIRA               BIT(1) STATIC INIT ('1'B) ;
 IF PRIMEIRA
 THEN DO ; /* INICIALIZA TABELA DE MENSAGENS */
    PRIMEIRA = '0'B ; /* INDICA TABELA JA' INICIALIZADA */
    INICIO (1) =
    'ERRO NO PROGRAMA OBJETO. FALTA INSTRUCAO PARA ALOCACA' ; /*01*/
    SEG (1) = 1 ;
    INICIO (2) =
    'PLAPP32 - RESTRICAO DE IMPLEMENTACAO. NUMERO MAXIMO P' ; /*02*/
    SEG (2) = 2 ;
    INICIO (3) =
    'ERRO NO PROGRAMA OBJETO. FALTA INSTRUCAO PARA ALOCACA' ; /*03*/
    SEG (3) = 3 ;
    INICIO (4) =
    'ERRO NO PROGRAMA OBJETO. FALTA INSTRUCAO PARA ALOCACA' ; /*04*/
    SEG (4) = 4 ;
    INICIO (5) =
    'ERRO NO PROGRAMA OBJETO. FALTA INSTRUCAO PARA ALOCACA' ; /*05*/
    SEG (5) = 5 ;
    INICIO (6) =
    'ERRO NO PROGRAMA OBJETO. FALTA INSTRUCAO PARA ALOCACA' ; /*06*/
    SEG (6) = 6 ;
    INICIO (7) =
    'CODIGO DE OPERACAO INVALIDO.                           ' ; /*  */
    SEG (7) = 0 ;
    INICIO (8) =
    'FIM NORMAL DE PROCESSAMENTO.                           ' ; /*  */
    SEG (8) = 0 ;
    INICIO (9) =
    'CHAMADA INVALIDA DE PROCEDURE OU FUNCAO.               ' ; /*  */
```

```
    SEG (9) = 0 ;
    INICIO (10) =
    'PROGRAMA OBJETO INCOMPLETO.                                    ' ; /*  */
    SEG (10) = 0 ;
    INICIO(11) =
    'DADO BOOLEANO INVALIDO EM COMANDO DE LEITURA                   ' ; /*  */
    SEG(11) = 0 ;
    INICIO(12) =
    'DADO INTEIRO MAIOR QUE O PERMITIDO, EM COMANDO DE LEI          ' ; /*08*/
    SEG(12) = 8 ;
    INICIO(13) =
    'DADO INTEIRO INVALIDO EM COMANDO DE LEITURA                    ' ; /*  */
    SEG(13) = 0 ;
    INICIO(14) =
    'DADO REAL INVALIDO EM COMANDO DE LEITURA                       ' ; /*  */
    SEG(14) = 0 ;
    INICIO(15) =
    'EXPOENTE DE DADO REAL FORA DOS LIMITES, EM COMANDO DE          ' ; /*09*/
    SEG(15) = 9 ;
    INICIO(16) =
    'DADO REAL MENOR QUE O PERMITIDO EM COMANDO DE LEITURA          ' ; /*  */
    SEG(16) = 0 ;
    INICIO(17) =
    'DADO REAL MAIOR QUE O PERMITIDO EM COMANDO DE LEITURA          ' ; /*  */
    SEG(17) = 0 ;
    /*****************************          ***************************/
    COMPLEM (1) =
    'O DE AREAS PARA EXECUCAO DO PROGRAMA.                          ' ; /*  */
    TER (1) = 0 ;
    COMPLEM (2) =
    'REVISTO DE QUADRUPLAS (5000) FOI ULTRAPASSADO.                 ' ; /*  */
    TER (2) = 0 ;
    COMPLEM (3) =
    'O DE CONSTANTES INTEIRAS PARA EXECUCAO DO PROGRAMA.            ' ; /*  */
    TER (4) = 0 ;
    COMPLEM (4) =
    'O DE CONSTANTES REAIS PARA EXECUCAO DO PROGRAMA.               ' ; /*  */
    TER (4) = 0 ;
    COMPLEM (5) =
    'O DE CONSTANTES BOOLEANAS PARA EXECUCAO DO PROGRAMA.           ' ; /*  */
    TER (5) = 0 ;
    COMPLEM (6) =
    'O DE CONSTANTES ALFANUMERICAS PARA EXECUCAO DO PRO-            ' ; /*07*/
    TER (6) = 7 ;
    COMPLEM (7) =
    'GRAMA.                                                         ' ; /*  */
    TER (7) = 0 ;
    COMPLEM(8) =
    'TURA                                                           ' ; /*  */
    TER(8) = 0 ;
    COMPLEM(9) =
    ' LEITURA                                                       ' ; /*  */
    TER(9) = 0 ;
/*****************************          ***************************/
      END ;
ELSE DO ;
       PUT FILE (SPRINT) EDIT(INICIO(NUM)) (COL(1) , A) ;
       N = SEG(NUM) ;
       FLIPFLOP = '1'B ;
       DO WHILE(N ¬= 0) ;
```

```
          IF FLIPFLOP
          THEN PUT FILE (SPRINT) EDIT(COMPLEM(N))(A) ;
          ELSE PUT FILE (SPRINT) EDIT(COMPLEM(N))(COL(1) , A) ;
          N = TER(N) ;
          FLIPFLOP = ¬FLIPFLOP ;
        END ;
      END ;
END PLAPP33 ;
```

```
/* PLAPP34 - GRAVA REGISTRO DE INSTRUCAO EM OBJ1 */
PLAPP34 : PROC(COP,B1,OP1,B2,OP2,B3,OP3) ;
/*****************************************************************/
/*   ESTA PROCEDURE VERIFICA SE O REGISTRO DE INSTRUCOES         */
/* (QUADRUPLAS) GERADAS JA' ESTA' COMPLETO, GRAVANDO-O EM CASO   */
/* POSITIVO. ALEM DISSO, ATUALIZA O CONTADOR DE INSTRUCOES       */
/* ('CINST') E O PONTEIRO PARA AS QUADRUPLAS NO REGISTRO COR-    */
/* RENTE ('LC1').                                                */
/*****************************************************************/
1 /* IMPRIME MENSAGENS DE ERRO */
$CONTINUE WITH PLAMP01 RETURN
1 /* MACRO AUXSCAN */
$CONTINUE WITH PLAMV01 RETURN
1 /* MACRO ERRO E COLUNA */
$CONTINUE WITH PLAMV06 RETURN
1 /* MACRO GERA E AUXGERA */
$CONTINUE WITH PLAMV11 RETURN
1 /* MACRO OBJ1 */
$CONTINUE WITH PLAMV13 RETURN
1DCL (COP,B1,B2,B3)    BIN FIXED(15,0) ;
 DCL (OP1,OP2,OP3)     BIN FIXED(31,0) ;
 DCL 1 QUADRA2(5)      UNAL BASED(PTOBJ1) ,
       2 BB1           BIN FIXED(15,0) ,
       2 LIXO1         CHAR(1) ,
       2 AUX           BIN FIXED(15,0) ,
       2 BB2           BIN FIXED(15,0) ,
       2 LIXO2         CHAR(3) ,
       2 BB3           BIN FIXED(15,0) ,
       2 LIXO3         CHAR(4) ;
 DCL 1 QUADRA3(5)      UNAL BASED(PTOBJ1) ,
       2 LIXO1         CHAR(4) ,
       2 COD           BIT(8) ,
       2 RESTO         CHAR(11) ;
 BB1(LC1) = B1 ;
 BB2(LC1) = B2 ;
 BB3(LC1) = B3 ;
 AUX(LC1) = COP ;
 COP1(LC1) = COD(LC1) ;
 OPND1(LC1,1) = OP1 ;
 OPND1(LC1,2) = OP2 ;
 OPND1(LC1,3) = OP3 ;
 IF LC1 = 5
 THEN DO ;
        WRITE FILE(OBJ1) FROM(REG1) ;
        CINST = CINST + 5 ;
        LC1 = 1 ;
      END ;
 ELSE LC1 = LC1 + 1 ;
 END PLAPP34 ;
```

```
         TITLE 'PLAPA03 - ROTINA PARA CALCULO DE FUNCAO HASH'
PLAPA03  CSECT
         DS    0H
         STM   14,12,12(13)        SAVE REGISTERS
         LR    12,15
         USING PLAPA03,12
         ST    13,#@1+4
         LR    15,13
         LA    13,#@1
         ST    13,8(0,15)
         LR    15,12
         CNOP  0,4
         B     *+76
#@1      DS    18F
         LM    2,3,0(1)            R1 -> LISTA DE ENDERECOS
*                                  R2 -> PARAMETRO ENTRADA
*                                  R3 -> PARAMETRO RETORNO
         MVC   AUX,4(2)
         XC    AUX,0(2)
         XC    AUX(2),AUX+2
         XC    AUX(1),AUX+1
         MVC   1(1,3),AUX
         NC    0(2,3),=X'003F'
         L     13,4(13)
         LM    14,12,12(13)
         LA    15,0(0,0)
         BR    14
AUX      DS    F
         END
```

H - Modo de Uso do CDP

Na descrição abaixo são mostrados os passos necessários para a geração e execução, via "batch", do Compilador Didático Pascal sob o sistema operacional MTS - Michigan Terminal System. Para o desenvolvimento deste trabalho foi utilizado o MTS implantado no Laboratório de Computação Científica (LCC) do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

1. Criação do arquivo que conterá o compilador em formato executável.

```
$SIGNON sigla
senha
$CREATE PASCAL
```

Observações:
- sigla é a identificação de um usuário autorizado a utilizar o sistema MTS;
- senha é a senha deste usuário;
- estas observações são válidas para os demais passos.

2. Criação de arquivos com as macros utilizadas pelos módulos em PL1.

```
$SIGNON sigla
senha
$CREATE PLAMTXX
$COPY *SOURCE* PLAMTXX
  texto da macro PLAMTXX
$ENDFILE
```

Observações:
- PLAMTXX é o nome da macro, conforme referenciada nos módulos em PL1;
- cada macro deve ser gravada num arquivo tipo "line".

3. Compilação de módulos em PL1, incluindo o Módulo Principal.

```
$SIGNON sigla
senha
$RUN  *PL1 SCARDS=*SOURCE* SPUNCH=-OBJ SPRINT=*PRINT*-
      PAR=SORMGIN=(2,72,1)
  programa fonte
$ENDFILE
$RUN *OBJUTIL
  EDIT PASCAL
  INCLUDE -OBJ
$ENDFILE
```

4. Montagem de módulos em Assembler.

```
$SIGNON sigla
senha
$RUN *ASMG SCARDS=*SOURCE* SPUNCH=-OBJ SPRINT=*PRINT*
  programa fonte
$ENDFILE
$RUN *OBJUTIL
  EDIT PASCAL
  INCLUDE -OBJ
$ENDFILE
```

5. Compilação de programas no CDP.

```
$SIGNON sigla
senha
$RUN PASCAL+*PL1LIB SCARDS=*SOURCE* SPRINT=*PRINT* 1=-OBJ1-
     2=-OBJ2 PAR=opções de compilação
  programa fonte em PASCAL
$
  dados para o programa compilado
$ENDFILE
```

Para utilização do Compilador Didático Pascal sob o sistema operacional OS/370 (Operating System/370) algumas pequenas modificações nos textos dos programas fonte, conforme apresentados no apêndice G, fazem-se necessárias.

Os passos relacionados abaixo indicam como proceder para a geração e execução do Compilador Didático Pascal sob o OS/370:

1. Modificação nos módulos em PL1.

Nos módulos em PL1, o comando MTS usado para efetuar a inclusão das macros, e que tem a forma

```
$CONTINUE WITH PLAMTXX RETURN
```

deve ser substituido pelo comando do pré-processador

```
%INCLUDE PLAMTXX ;
```

Além disso, devem ser retiradas as opções TITLE dos comandos OPEN existentes nos módulos PLAPP00, PLAPP31 e PLAPP32.

2. Criação da biblioteca de macros e da biblioteca de módulos em formato "load".

```
//jobname  JOB
//         PGM=IEFBR14
//MACROS   DD    DSNAME=PASCAL.MACROLIB.PLI,DISP=(NEW,CATLG),
//           UNIT=unitname,DCB=(RECFM=FB,BLKSIZE=3120,
//           LRECL=80),SPACE=(3120,(37,1,4))
//LOAD     DD    DSNAME=PASCAL.LIBLOAD,DISP=(NEW,CATLG),
//           UNIT=unitname,SPACE=(TRK,(5,1,3))
```

3. Gravação da biblioteca de macros.

```
//jobname  JOB
//         PGM=IEBGENER
//SYSIN    DD    DUMMY
//SYSPRINT DD    SYSOUT=A
//SYSUT2   DD    DSNAME=PASCAL.MACROLIB.PLI(PLAMTXX),
//           DISP=SHR
```

```
//SYSUT1   DD    *
    texto da macro PLAMTXX
//
```

4. Montagem de módulos em Assembler.

```
//jobname  JOB
//ASM      PGM=IFOXOO,PARM='OBJ,NODECK'
//SYSLIB   DD    DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD    DSNAME=&&SYSUT1,UNIT=unitname,
//           SPACE=(CYL,(1,1))
//SYSUT2   DD    DSNAME=&&SYSUT2,UNIT=unitname,
//           SPACE=(CYL,(1,1))
//SYSUT3   DD    DSNAME=&&SYSUT3,UNIT=unitname,
//           SPACE=(CYL,(1,1))
//SYSPRINT DD    SYSOUT=A
//SYSPUNCH DD    SYSOUT=B
//SYSGO    DD    DSNAME=&&OBJSET,UNIT=unitname,
//           SPACE=(CYL,(1,1)),DISP=(NEW,PASS),
//           DCB=(RECFM=FBS,BLKSIZE=3040,LRECL=80)
//SYSIN    DD    *
    texto do módulo em assembler
//         EXEC  PGM=IEWL,PARM=(XREF,LET,LIST,NCAL),
//           COND=(8,LT,ASM)
//SYSLIN   DD    DSNAME=&&OBJSET,DISP=(OLD,DELETE)
//SYSLMOD  DD    DSNAME=PASCAL.LIBLOAD(PLAPAXX),DISP=SHR
//SYSUT1   DD    DSNAME=&&SYSUT1,UNIT=unitname,
//           SPACE=(CYL,(1,1))
//SYSPRINT DD    SYSOUT=A
```

5. Compilação de módulos em PL1 (exceto o Módulo Principal - PLAPPOO).

```
//jobname  JOB
//PLI      PGM=IEMAA,PARM='OBJECT,NODECK,M'
//SYSPRINT DD    SYSOUT=A
//SYSLIN   DD    DSNAME=&&LOADSET,DISP=(NEW,PASS),
//           UNIT=unitname,DCB=(RECFM=FBS,BLKSIZE=3040,
//           LRECL=80),SPACE=(CYL,(1,1))
//SYSUT1   DD    DSNAME=&&SYSUT1,UNIT=unitname,
```

```
//            SPACE=(CYL,(1,1)),DCB=BLKSIZE=1024
//SYSUT3   DD    DSNAME=&&SYSUT3,UNIT=unitname,
//            SPACE=(CYL,(1,1)),DCB=BLKSIZE=80
//SYSIN    DD    *
    texto do módulo PLAPPXX
//         EXEC  PGM=IEWL,PARM='XREF,LIST,NCAL',
//            COND=(9,LT,PLI)
//SYSLIB   DD    DSNAME=SYS1.PLIBASE,DISP=SHR
//SYSLMOD  DD    DSNAME=PASCAL.LIBLOAD(PLAPPXX),DISP=SHR
//SYSUT1   DD    DSNAME=&&SYSUT1,UNIT=unitname,
//            SPACE=(CYL,(1,1)),DCB=BLKSIZE=1024
//SYSPRINT DD    SYSOUT=A
//SYSLIN   DD    DSNAME=&&LOADSET,DISP=(OLD,DELETE)
```

6. Compilação do Módulo Principal - PLAPP00.

```
//jobname  JOB
//PLI      PGM=IEMAA,PARM='OBJECT,NODECK,M'
//SYSPRINT DD    SYSOUT=A
//SYSLIN   DD    DSNAME=&&LOADSET,DISP=(NEW,PASS),
//            UNIT=unitname,DCB=(RECFM=FBS,BLKSIZE=3040,
//            LRECL=80),SPACE=(CYL,(1,1))
//SYSUT1   DD    DSNAME=&&SYSUT1,UNIT=unitname,
//            SPACE=(CYL,(1,1)),DCB=BLKSIZE=1024
//SYSUT3   DD    DSNAME=&&SYSUT3,UNIT=unitname,
//            SPACE=(CYL,(1,1)),DCB=BLKSIZE=80
//SYSIN    DD    *
    texto do Módulo Principal - PLAPP00
//         EXEC  PGM=IEWL,PARM='XREF,LIST,NONCAL',
//            COND=(9,LT,PLI)
//SYSLIB   DD    DSNAME=SYS1.PLIBASE,DISP=SHR
//SYSLMOD  DD    DSNAME=PASCAL.LIBLOAD(PASCAL),DISP=SHR
//SYSUT1   DD    DSNAME=&&SYSUT1,UNIT=unitname,
//            SPACE=(CYL,(1,1)),DCB=BLKSIZE=1024
//SYSPRINT DD    SYSOUT=A
//SYSLIN   DD    DSNAME=&&LOADSET,DISP=(OLD,DELETE)
```

7. Compilação de programas no CDP.

```
//jobname  JOB
//         EXEC  PGM=PASCAL
//STEPLIB  DD    DSNAME=PASCAL.LIBLOAD,DISP=SHR
//OBJ1     DD    DSNAME=&&OBJ1,UNIT=unitname,
//            SPACE=(TRK,(1,1))
//OBJ2     DD    DSNAME=&&OBJ2,UNIT=unitname,
//            SPACE=(TRK,(1,1))
//SPRINT   DD    SYSOUT=A
//SCARDS   DD    *
    programa fonte em PASCAL
$
    dados para o programa compilado
/*
//
```