UM GERADOR DE PROGRAMAS COBOL

Paulo Asterio de Castro Guerra

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.)

Aprovado por:

Prof. Estevam Gilberto de Simone
(Presidente)

Prof. José Lucas M. Rangel Netto

Prof. Célio Cardoso Guimarães

Rio de Janeiro, RJ - BRASIL

Outubro de 1979

GUERRA, PAULO ASTERIO DE CASTRO

Um Gerador de Programas COBOL

(Rio de Janeiro) 1979.

VII, 107 p. 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1979)

Tese - Univ.Fed.Rio de Janeiro Fac.Engenharia

1. Programação de Computadores I.COPPE/UFRJ II. Um Gerador de Programas COBOL.

Ā Aurora,
minha mãe e
primeira mestra.

AGRADECIMENTOS

Ao Professor Estevam Gilberto de Simone, meus agradecimentos pelo entusiasmo e dedicação com que me orientou no desenvolvimento dos trabalhos que resultaram nesta monografia.

À Beth, pelo estímulo recebido e cooperação em todas as horas dedicadas a esta tarefa, sem o que teria sido impossível realizá-la.

Aos Professores e Colegas, de escola e profissão, em cujo convívio surgiram e se destilaram as idéias básicas que motivaram este trabalho. Agradeço em particular ao amigo Antonio Wanderley pelos ensinamentos recebidos.

Aos Professores Célio Cardoso Guimarães e José Lucas Mourão Rangel Netto, meus agradecimentos, por terem acedido em participar da banca examinadora.

SINOPSE

Após uma breve revisão dos instrumentos de auxílio à programação já existentes, é apresentado um novo preprocessador para a linguagem COBOL denominado GPC.

O GPC possui algumas características de um processador de macros, orientado para a linguagem COBOL, oferecendo, porém, facilidades para a composição do texto gerado na ordem especificada pelo programador. Através desta facilidade de 'composição de texto', o programador pode definir e utilizar módulos (ou macros) responsáveis pela geração de todos os comandos necessários para a implementação de uma dada 'função' do programa, e que irão distribuir-se em diferentes pontos do programa gerado.

A apresentação do GPC é feita na forma de um manual de utilização, além de um conjunto de especificações para sua implementação onde são descritas as estruturas de dados e algoritmos utilizados pelos quatro programas que compõem os processos de 'geração de programas' e 'atualização da biblioteca de macros'.

ABSTRACT

After a brief review of the existing programming tools, a new COBOL preprocessor, named GPC, is presented.

The GPC has some characteristics of a COBOL oriented macroprocessor, but in addition to these there are some features for text composition that enable the programmer to specify the ordering of the various pieces of generated text. This 'text composition' facility allows the definition of modules (or macros) each of which is responsible for the full implementation of a given function in a COBOL program that may be scattered through the generated source code.

A User's Manual is included, describing the features and utilization of the GPC, as well as a set of specifications for its implementation, where the data structures and algorithms used by the four programs that constitute the 'program generator' and 'library maintenance' modules, are described.

ÍNDICE

	Páginas
CAPÍTULO I - Introdução	1
1.1. - Motivação	1
1.2. - Descrição do Tema	4
CAPÍTULO II - Revisão da Literatura	5
2.1. - Âmbito da Revisão da Literatura	5
2.2. - Programas Auxiliares Existentes	6
2.2.1. - Classificação	6
2.2.2. - Processadores de Macros de Uso Geral.	6
2.2.3. - Abreviadores	7
2.2.4. - Tradutores de Tabelas de Decisão . .	8
2.2.5. - Geradores de Programas - Tipo	8
2.2.6. - Extensores da Linguagem COBOL	9
2.3. - Técnicas de Programação	10
2.3.1. - Programação Modular	10
2.3.2. - Programação Estruturada	11
2.4. - Aspectos Psicológicos da Programação	12
CAPÍTULO III - Objetivos e Condicionantes do GPC . .	13
3.1. - Descrição dos Objetivos e Condicionantes . . .	13
CAPÍTULO IV - Apresentação do GPC	15
4.0. - Nota Explicativa	15
4.1. - Introdução	16
4.2. - A Utilização do GPC	22
4.2.1. - A Biblioteca de Macros	22
4.2.2. - A Geração de Programas	23
4.3. - A Linguagem do GPC	27

Páginas

CAPÍTULO V - Implementação do GPC 62

5.1. - Introdução 62

5.2. - Descrição dos Arquivos Utilizados pelo GPC . 64

5.2.1. - Arquivo CARD 64

5.2.2. - Arquivo CODE 65

5.2.3. - Arquivo MLIB 71

5.2.4. - Arquivo TEXT 72

5.2.5. - Arquivo PROG 74

5.3. - Descrição dos Programas 76

5.3.1. - Programa GPC-1 76

5.3.2. - Programa GPC-2 86

5.3.3. - Programa GPC-3 99

5.3.4. - Programa GPC-4 100

CAPÍTULO VI - Conclusões e Recomendações 101

6.1. - Resumo e Discussão 101

6.2. - Recomendações 104

BIBLIOGRAFIA 105

CAPÍTULO I

Introdução

"Hardware is Easy, it's
Software that's Hard"
BERNSTEIN [1]

1.1. Motivação

O processo de programação é atualmente a atividade crítica que restringe a plena utilização dos recursos de computação já disponíveis. McCRACKEN [2] sugere que somente com a adoção de métodos de programação radicalmente novos ("nonprocedural languages", por ex.) serão superadas as limitações dos métodos convencionais de programação.

O estabelecimento de um novo método de programação e sua aceitação por toda a comunidade de técnicos e usuários requer, porém, um tempo ainda indeterminado durante o qual os métodos convencionais continuarão sendo utilizados e dos quais cada vez mais será exigido.

Uma descrição geral dos métodos convencionais de programação é feita por YOHE [3], decompondo o processo de elaboração de um programa em 9 passos, conforme esquematizado na figura 1.1. Outros autores apresentam opiniões divergentes a respeito dessa decomposição (HOARE [4]), particularmente quanto à ordem em que os vários passos devem ser cumpridos.

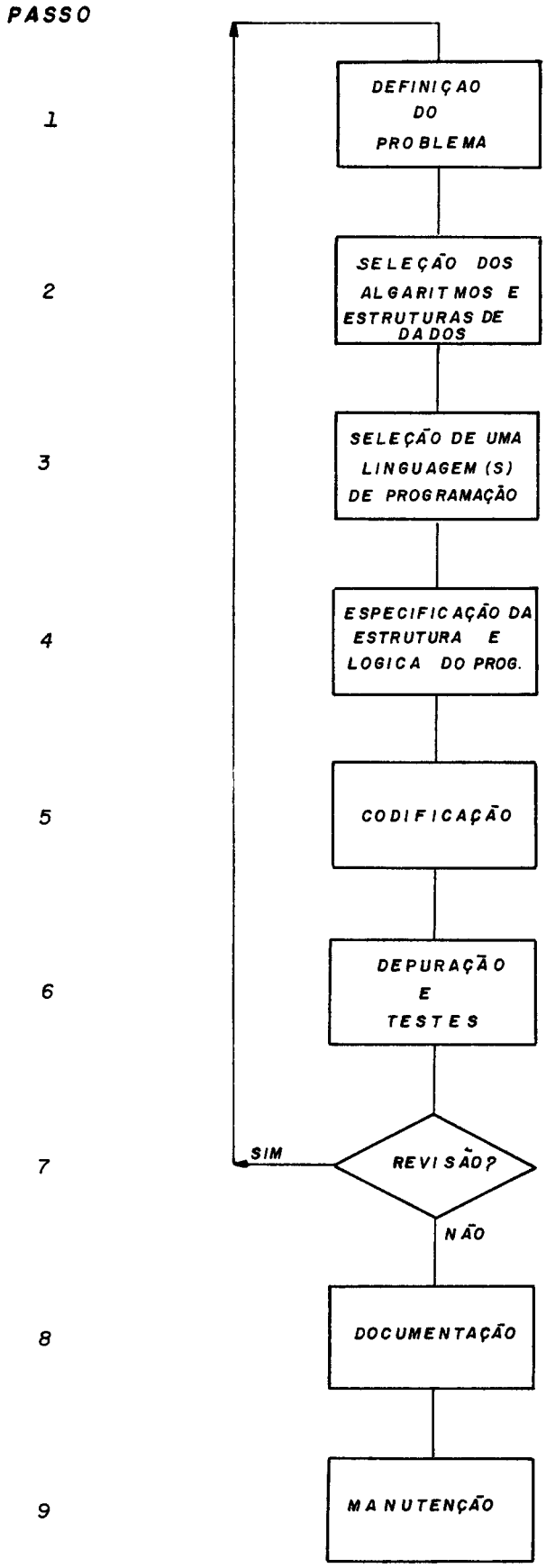


Fig. 1.1 - PROCESSO DE ELABORAÇÃO DE UM PROGRAMA

Visando a otimização dos métodos convencionais surgiram várias técnicas e programas auxiliares ('software tools') orientados para reduzir ou eliminar as dificuldades de cada um dos passos citados. REIFER e TRATTNER [5] relacionam 70 diferentes itens, entre técnicas e tipos de programas auxiliares existentes, classificando-os conforme a sua aplicação nas seguintes categorias: simulação, desenvolvimento, teste e avaliação, operação e manutenção, medida de performance e suporte de programação.

1.2. Descrição do Tema

O interesse do autor no presente trabalho está dirigido para a elaboração de um programa auxiliar para o desenvolvimento de programas COBOL, visando a simplificação das tarefas de especificação da estrutura e lógica do programa e de codificação. O programa auxiliar em questão foi denominado GPC, de Gerador de Programas COBOL, e assim será referenciado ao longo do texto.

A linguagem COBOL foi escolhida como alvo principal devido à importante posição que ocupa como linguagem mais difundida em todo o mundo, ao lado da relativa escassez de programas auxiliares congêneres. A experiência profissional do autor contribuiu decisivamente tanto para a formulação preliminar dos objetivos e condicionantes do GPC, baseados nas principais dificuldades encontradas durante vários anos de prática em contato com a linguagem COBOL, como também permitiu a adaptação desses objetivos e condicionantes em função das reações normalmente encontradas na comunidade de técnicos à adoção de programas auxiliares em geral.

O Capítulo II deste trabalho contém uma revisão da literatura pertinente a este tema. No Capítulo III são apresentados os objetivos e condicionantes adotados no projeto do GPC. O Capítulo IV descreve a utilização do GPC, na forma de um Manual do Usuário, estando as especificações para sua implementação no Capítulo V. Finalmente, no Capítulo VI, é feita uma avaliação crítica do trabalho, com as conclusões e recomendações para futuras pesquisas.

CAPÍTULO II

REVISÃO DA LITERATURA

2.1. Âmbito da Revisão de Literatura

A Revisão de Literatura foi direcionada para os setores ligados à atividade de programação, visando:

a) uma análise dos programas auxiliares existentes, a fim de se avaliar a oportunidade de se incluir um novo elemento nesse conjunto, assim como modelar este novo programa aproveitando-se dos resultados já alcançados até a presente data;

b) um estudo das técnicas de programação em evidência, visto que qualquer programa auxiliar a ser oferecido deve, necessariamente, integrar-se à utilização destas técnicas e não contrapor-se a elas;

c) investigação dos aspectos psicológicos da programação a serem, obrigatóriamente, considerados em vista da função de interface homem-máquina inerente aos programas auxiliares.

As seções seguintes deste capítulo apresentam um extrato da matéria publicada pesquisada, pertinente aos assuntos acima citados.

2.2. Programas Auxiliares Existentes

2.2.1. Classificação

Os programas auxiliares podem ser classificados, em função do âmbito da aplicação, em:

a) programas auxiliares de uso geral, que podem ser utilizados em conjunto com uma variedade de linguagens base. Nesta categoria enquadram-se os processadores de macros de uso geral.

b) programas auxiliares de uso restrito, que se aplicam a uma única linguagem base. NAFTALY [6] classifica os programas auxiliares orientados para o desenvolvimento de programas COBOL, de acordo com a sua única ou principal função, em: abreviadores, tradutores de tabelas de decisão e geradores de programas tipo. Uma quarta categoria que deve ser destacada engloba os extensores da linguagem COBOL para suportar programação estruturada.

2.2.2. Processadores de Macros de Uso Geral

BROWN [7] faz uma avaliação das facilidade oferecidas e limitações dos processadores de macros existentes, além de uma discussão das características básicas de projeto e implementações desses processadores.

Mais recentemente, COLE [8] apresenta os objetivos de vários processadores, descrevendo os algoritmos e formas de implementação utilizados.

As características básicas que diferenciam os vários processadores de macros existentes são:

a) sintaxe das macro chamadas, podendo ser mais ou menos rígidas quanto aos delimitadores utilizados e disposição dos argumentos na chamada;

b) forma de interpretação do texto, podendo ou não admitir macro chamadas como parte de outra macro chamada, por exemplo;

c) facilidades de manipulação de argumentos oferecidas, tais como manipulação de 'strings', manutenção de dicionários de símbolos, testes de condições, e outras.

A utilização de um processador de macros de uso geral é recomendável quando se exige a utilização de diferentes linguagens de programação concomitantemente. Neste caso, com a definição de macros apropriadas, pode-se definir um novo nível linguístico, comum às diferentes linguagens base utilizadas, obtendo-se assim benefícios dificilmente obtidos com programas auxiliares independentes para cada linguagem.

2.2.3. Abreviadores

Os abreviadores podem ser considerados como processadores de macros, em geral restritos a uma linguagem específica, e que executam frequentemente outras funções visando uma redução no esforço de codificação e depuração sintática dos programas produzidos.

O uso do termo "abreviadores" deve-se à reduzida capacidade de processamento de macros oferecida, quando comparados com os processadores de macros de uso geral, permitindo-se, por vezes, apenas o uso de macros sem parâmetros

que correspondem, em última análise, a uma simples abreviatura.

As outras funções normalmente executadas pelos abreviadores são: formatação do texto produzido, detecção e correção automática de erros de sintaxe, e auditoria de padrões de programação. Estas funções exigem que se analise sintaticamente todo o texto produzido, em diferentes níveis de complexidade (desde o simples reconhecimento da ocorrência de uma palavra chave, até a inclusão automática de símbolos ausentes no texto original), o que implica em maior complexidade do programa auxiliar e exige a sua adaptação a cada nova versão ou 'dialeto' da linguagem base.

NAFTALY |[6]| apresenta as características de vários abreviadores COBOL existentes.

2.2.4. Tradutores de Tabelas de Decisão

Os tradutores de tabelas de decisão aceitam como entrada a especificação de uma T.D. e produzem uma seção de procedimentos COBOL equivalente. Segundo POOCH |[9]| a sua utilização só é vantajosa a partir de um certo nível de complexidade da situação em que se deseja decidir entre vários procedimentos a seguir.

NAFTALY |[6]| apresenta as características de vários tradutores de T.D. existentes.

2.2.5. Geradores de Programas - Tipo

A característica básica dos geradores de pro-

gramas-tipo é a existência de uma fórmula padrão para a solução de uma classe de problemas, por ex.: impressão de relatórios analíticos, intrínseca ao gerador que é adaptada através de parâmetros para cada nova situação. Algum geradores permitem a inclusão de comandos COBOL em pontos específicos do programa gerado, por ex.: após a leitura do arquivo principal, dando maior flexibilidade à classe de programas gerados.

NAFTALY [6] apresenta as características de vários geradores de programas-tipo existentes.

BABENKO [10] descreve um programa auxiliar, denominado MAKROBOL, que permite a geração de uma ampla gama de programas a partir de 9 funções intrinsecas básicas combinadas com facilidades de processamento de macros.

2.2.6. Extensores da linguagem COBOL

Os extensores da linguagem COBOL visam fornecer ao programador as estruturas de controle do tipo CASE e blocos BEGIN-END, inexistentes na linguagem COBOL e cuja ausência dificulta a elaboração de programas estruturados.

WEINBERG [11] apresenta as características principais do programa METACOBOL que executa também funções de processamento de macros.

TAUSWORTHE [12] descreve o preprocessador CRISP que visa oferecer ao programador um conjunto de estruturas de controle aplicável à qualquer linguagem base de programação.

2.3. Técnicas de Programação

2.3.1. Programação Modular

O princípio básico da técnica de programação modular consiste na subdivisão de um programa em vários subprogramas mais simples, que podem ser escritos e testados separadamente, e posteriormente integrados em um único programa.

Apesar da universalidade do princípio de "subdivisão de problemas" em que se baseia, a técnica de programação modular não tem produzido, na prática, os resultados que se esperaria obter: rapidez de programação, confiabilidade, adaptabilidade, dentre outros.

PARNAS [13,14,15] sugere que a principal dificuldade consiste na especificação dos módulos, que tradicionalmente correspondem a vários passos do programa, e propõe um novo método para subdivisão em módulos, denominado "information hiding", segundo o qual cada módulo deve corresponder a uma decisão de projeto, que se torna transparente para os demais módulos. Por exemplo: um módulo corresponderia à estrutura de dados utilizada com as respectivas rotinas de acesso, ficando transparente para os demais módulos essa decisão de projeto.

HARDING [16] discute as causas do aparente insucesso da programação modular, e conclui que um esforço no sentido de se superar as dificuldades que se colocam ao seu uso de forma mais efetiva seria amplamente recompensado pelas vantagens decorrentes.

2.3.2. Programação Estruturada

Depois do aparecimento das linguagens de alto-nível, como FORTRAN e COBOL, o principal avanço no âmbito da programação foi, sem dúvida, a introdução dos conceitos de programação estruturada (DIJKSTRA [17]).

Através das técnicas de programação estruturada, torna-se explícita a estrutura de controle do programa, pela substituição do uso de rótulos e GO-TO's por blocos BEGIN-END, IF-THEN-ELSE, e DO-WHILE, obtendo-se assim uma maior clareza no programa e facilitando a sua depuração através de provas formais.

Embora a linguagem COBOL, ao contrário do que ocorre com ALGOL e PL/I, não se adapte plenamente ao uso das técnicas de programação estruturada, alguns autores advogam a aplicação dessas técnicas através de uma disciplinação da programação COBOL (CLURE [18] e GELDER [19]).

2.4. Aspectos Psicológicos da Programação

Quase sempre relegados a segundo plano, os aspectos psicológicos envolvidos no trabalho de programação são muitas vezes os principais responsáveis pelos grandes sucessos, ou insucessos, do trabalho realizado. O grau de motivação da equipe é, sabidamente,determinante da qualidade dos trabalhos produzidos por esta equipe, em qualquer ramo de atividade do homem.

Objetivando aumentar o grau de motivação dos programadores WEINBERG [20] propõe uma estratégia de programação que consiste em desenvolver o programa de forma que rapidamente se possa fazê-lo "funcionar", e só depois, através de refinamentos sucessivos, preocupar-se em atingir os requisitos secundários de eficiência (tempo de execução, memória requerida, e outros).

CAPÍTULO III

OBJETIVOS E CONDICIONANTES DO GPC

3.1. Descrição dos Objetivos e Condicionantes

Os objetivos básicos que se procurou atingir, com a elaboração do presente trabalho, foram:

a) imprimir maior rapidez ao processo de programação, permitindo que os programas desejados possam ser produzidos rapidamente, satisfazendo aos requisitos funcionais básicos, e

b) tornar evolutivo o trabalho de uma equipe de programadores, dirigindo os esforços da equipe para a solução de novos problemas e o aperfeiçoamento de soluções existentes, evitando-se esforços inúteis na investigação de problemas já solucionados.

Ao lado desses objetivos impôs-se as seguintes condicionantes ao trabalho:

a) minimizar a carga de aprendizado necessária para que um programador COBOL possa utilizar as facilidades oferecidas; e

b) limitar os recursos necessários para a implementação dessas facilidades ao mínimo indispensável.

A compatibilização desses objetivos e condicionantes no programa proposto - o GPC - se fez através das seguintes diretrizes básicas seguidas:

a) o GPC deverá oferecer as facilidades de processamento

de macros básicas, que permitem a redução do esforço de codificação;

b) o GPC deverá manter uma Biblioteca de Macros a ser criada e atualizada pelo programador;

c) a linguagem utilizada para comandar o GPC deve ser simples e possuir algum parentesco com o COBOL;

d) a definição de macros deve ser suficientemente flexível para permitir a inclusão numa mesma macro de comandos afins mas que não aparecem juntos no programa COBOL (exemplo: cláusulas SELECT e FD de um mesmo arquivo);

e) a implementação do GPC deve ser feita unicamente na linguagem COBOL, e utilizando o subconjunto mínimo de instruções do padrão internacional;

f) na implementação do GPC só devem ser utilizados arquivos organizados sequencialmente, admitindo-se o acesso direto se necessário.

CAPÍTULO IV

APRESENTAÇÃO DO GPC

4.0. Nota Explicativa

A apresentação das características do GPC será feita sob a forma de um Manual do Usuário, nas seções seguintes deste capítulo, no qual estão descritas as facilidades oferecidas, assim como a linguagem de comando utilizada.

4.1. Introdução

O GPC (Gerador de Programas COBOL) é um instrumento auxiliar para o desenvolvimento de programas COBOL, cujo objetivo básico é liberar o programador do trabalho árduo de codificação, permitindo que ele se concentre na solução de novos problemas e no aperfeiçoamento das soluções existentes.

O Processamento de Macros

O GPC oferece as facilidades de um processador de macros, com características especiais orientadas para o seu uso em conjunto com a linguagem COBOL.

A característica fundamental de um processador de macros, como o GPC, é oferecer ao programador facilidades de substituição de texto, isto é: o processador recebe um texto de entrada escrito pelo programador e fornece como saída um novo texto, usualmente mais longo que o texto de entrada. A transformação do texto de entrada no texto gerado é feita segundo regras estabelecidas nas definições das macros. A chamada de uma macro no texto de entrada é substituida, no texto gerado, pelo texto resultante da aplicação das regras definidas pela macro aos argumentos fornecidos na chamada da macro.

O exemplo 4.1(a) ilustra o processo de macro-expansão, podendo-se observar que:

- tanto na definição da macro FILE (linhas 1 a 6) como no texto de entrada (linhas 7 a 14) os registros assinalados com % contêm instruções ao GPC, e não são transcritas para o texto gerado (linhas 15 a 23);

Exemplo 4.1(a) - Processamento de Macros

Definição da Macro FILE

```
1    %    MACRO FILE, PARM=(NAME, BLOCK, LRECL).
2         FD  &NAME
3             BLOCK CONTAINS  &BLOCK RECORDS
4             RECORDS CONTAINS &LRECL CHARACTERS
5             RECORDING MODE IS F.
6    %    MACROEND.
```

Texto de Entrada

```
7    %    GENERATE PROG.
8         ...
9         DATA DIVISION.
10        FILE SECTION.
11   %    FILE WITH NAME=CARTAO, BLOCK=1, LRECL=80.
12        01 REG-CARTAO.
13        ...
14   %    ENDGEN.
```

Texto Gerado

```
15        ...
16        DATA DIVISION.
17        FILE SECTION.
18        FD   CARTAO
19             BLOCK CONTAINS 1 RECORDS
20             RECORD CONTAINS 80 CHARACTERES
21             RECORDING MODE IS F.
22        01 REG-CARTAO.
23        ...
```

- no texto gerado as linhas 15 a 17, 22 e 23 foram transcritas diretamente do texto de entrada, enquanto que as linhas 18 a 21 substituem a chamada da macro FILE no texto de entrada (linha 11);

- as linhas 18 a 21 foram obtidas a partir da definição da macro FILE, substituíndo-se os parametros formais (assinalados por &) pelos argumentos correspondentes fornecidos na chamada da macro.

A Composição do Texto

Examinando-se um programa escrito em COBOL verifica-se que vários trechos do programa que aparecem em diferentes pontos da listagem são responsáveis pela implementação de uma única função do programa. Por exemplo: as cláusulas SELECT, FD, áreas de trabalho e rotinas de acesso de um arquivo.

A facilidade de composição do texto, oferecida pelo GPC, permite reunir-se numa mesma macro todos os comandos relativos a uma função, bastando que se especifique onde, no programa COBOL, cada trecho do texto gerado deve ser inserido.

Esta facilidade é a principal característica que diferencia o GPC de outros processadores de macros, e que o faz um poderoso instrumento auxiliar à programação COBOL.

O exemplo 4.1(b) ilustra esse processo, podendo-se observar que:

- na definição da macro PROG são definidos os pontos de inserção SELECTS e FDS (linhas 6 e 9);

Exemplo 4.1(b) - Composição do Texto

Definição da macro PROG

```
1    %    MACRO PROG, PARM=NAME.
2    %    INS ROOT.
3         IDENTIFICATION DIVISION.
4         ...
5         INPUT-OUTPUT SECTION.
6    %    DEF SELECTS.
7         ...
8         FILE SECTION.
9    %    DEF FDS.
10        ...
11   %    MACROEND.
```

Definição da macro FILE

```
12   %    MACRO FILE, PARM=(NAME,...).
13   %    INS SELECTS.
14             SELECT &NAME ASSIGN TO ...
15   %    INS FDS.
16        FD &NAME
17             BLOCK CONTAINS ...
18        ...
19   %    MACROEND.
```

Exemplo 4.1(b) - Continuação

Texto de Entrada

```
20    %     GENERATE EXEMPLO.
21    %     PROG WITH NAME=TESTE.
22    %     FILE WITH NAME=ARQ-1, ... .
23    %     FILE WITH NAME=ARQ-2, ... .
24          ...
25    %     ENDGEN.
```

Texto Gerado

```
26          IDENTIFICATION DIVISION.
27          ...
28          INPUT-OUTPUT SECTION.
29                SELECT ARQ-1 ASSIGN TO....
30                SELECT ARQ-2 ASSIGN TO ...
31          ...
32          FILE SECTION.
33          FD ARQ-1
34             BLOCK CONTAINS ...
35          ...
36          FD ARQ-2
37             BLOCK CONTAINS ...
38          ...
39          ...
```

- na definição da macro FILE as linhas 13 e 15 especificam os pontos SELECTS e FDS, onde deverão ser inseridos, respectivamente, o texto gerado pela linha 14 e o texto gerado pelas linhas 16 em diante;

- no texto gerado (linhas 26 a 39) os vários segmentos de texto aparecem devidamente ordenados.

As Facilidades de Geração de Texto

Além das facilidades descritas anteriormente, o GPC oferece facilidades de parametrização, geração iterativa (blocos LOOP) e geração condicional (blocos IF). Todas essas facilidades serão descritas em detalhe nas seções seguintes.

4.2. A utilização do GPC

4.2.1. A Biblioteca de Macros

A criação e manutenção de uma Biblioteca de Macros de forma integrada é de fundamental importância para uma otimização dos benefícios oferecidos pelo GPC.

A Biblioteca de Macros deverá conter módulos de uso geral tais que interligando-se alguns desses módulos, modificados com base nos argumentos fornecidos, possa-se montar a estrutura básica de qualquer programa desejado, à qual deverão ser incorporadas apenas as instruções específicas ao programa, em pontos já definidos.

A formação da Biblioteca de Macros não é, óbvi amente, uma tarefa imediata e, deve-se frisar, nunca poderá ser considerada concluída, pois a própria dinâmica do trabalho de programaçao irá exigir uma constante revisão e ampliação dessa Biblioteca.

Como ponto de partida, porém, pode-se definir um conjunto de macros que inclua:

(a) definição da estrutura básica de um programa com os vários títulos de divisões e seções, cláusulas características da instalação tanto na IDENTIFICATION DIVISION como na ENVIRONMENT DIVISION, procedimentos padronizados de início e término, etc. Essa macro (ou macros) deverá definir os pontos de inserção mais gerais, tais como: ponto dos SELECT's, dos FD's, WORKING-STORAGE e PROCEDURE DIVISION.

(b) procedimentos padronizados de atualização, intercalações de registros, classificação, e outros, que correspondem,

normalmente a uma seção de procedimentos e que definem a estrutura de controle superior do programa. Essas macros irão comandar a inserção de texto em pontos definidos pelas macros básicas, definindo, por sua vez novos pontos de inserção, tais como: ponto para inserção dos procedimentos de inclusão, numa seção de atualização de arquivo.

(c) definição de arquivos, utilizando também os pontos de inserção definidos pelas macros básicas e, eventualmente,os pontos definidos pelas macros de seções.

(d) rotinas genéricas de crítica de dados, operações com datas, e outras rotinas de uso comum, com as suas respectivas áreas de trabalho.

Posteriormente, a medida que forem sendo identificados os problemas típicos da instalação, as soluções gerais desses problemas devem ser incorporados à Biblioteca, enriquecendo, assim, o seu acervo.

Na rotina de manutenção da Biblioteca de Macros, ilustrada na figura 4.2.1, as definições de macros que se deseja incluir ou substituir são analisadas sintáticamente, emitindo-se uma listagem onde são assinalados os erros detectados, e dando origens a um arquivo intermediário com as definições de forma codificada. Em seguida é gerada a nova Biblioteca de Macros, a partir da Biblioteca existente, das definições de macros préviamente codificadas e das especificações fornecidas.

4.2.2. A Geração de Programas

A geração de um programa se faz com base num

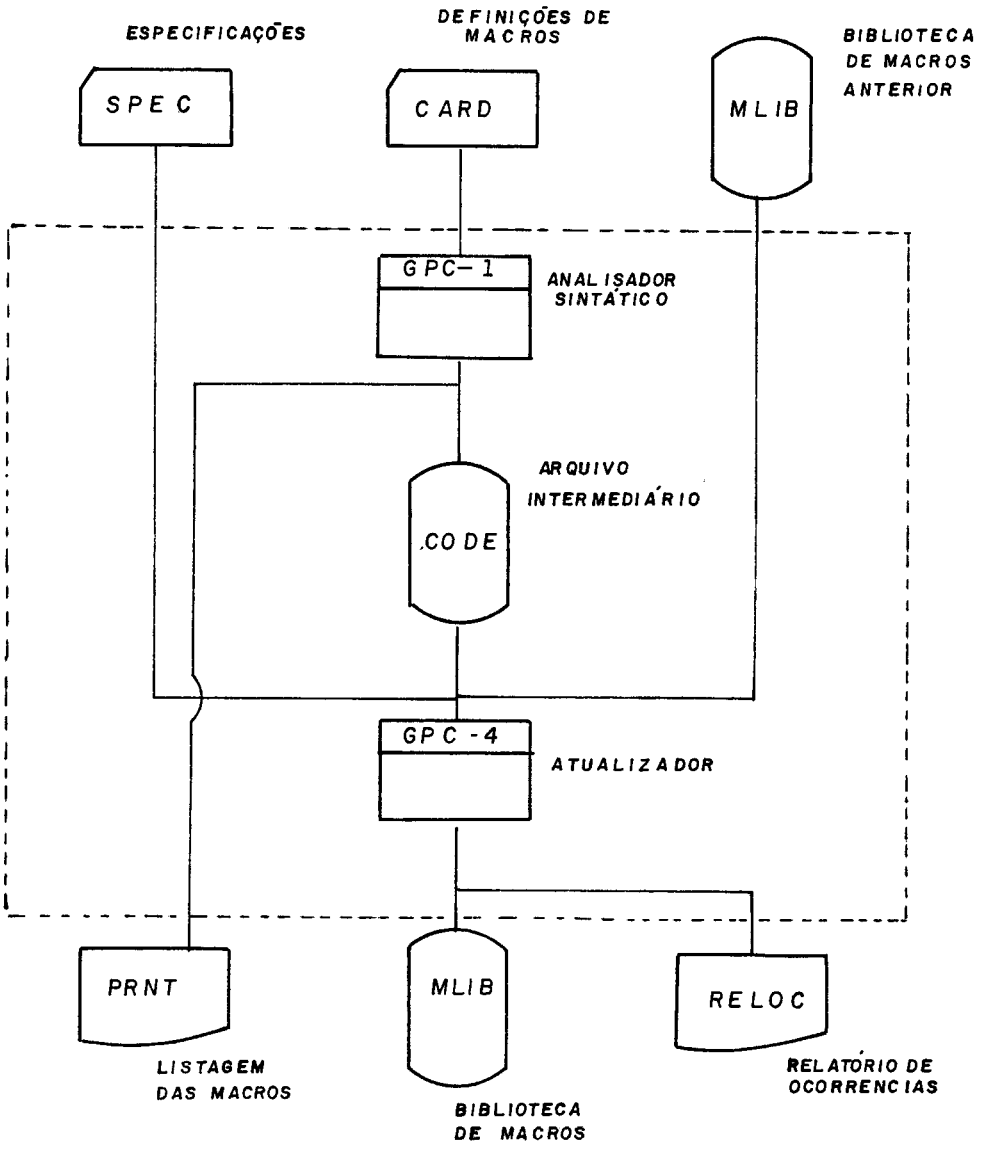


FIG. 4.2.1 - ROTINA DE ATUALIZAÇÃO DA BIBLIOTECA DE MACROS

texto de entrada fornecido pelo programador, e que especifica o programa a ser gerado. A especificação do programa consiste em um ou mais blocos de geração, que contêm, normalmente, as chamadas de algumas macros da Biblioteca, que irão definir a estrutura do programa desejado, e um conjunto de instruções para a geração do texto específico ao programa desejado, já endereçando as suas várias partes aos pontos da estrutura onde deverão ser inseridos. O algoritmo utilizado para a composição do texto permite que os comandos de inserção venham à frente da definição dos pontos correspondentes, desobrigando assim o programador de ordenar as chamadas de macro e comandos de inserção.

Opcionalmente o texto de entrada pode conter, além dos blocos de geração, definições de macros a serem utilizadas pelo programa.

Na rotina de Geração de Programas, ilustrada na figura 4.2.2, o texto de entrada é analisado sintáticamente, emitindo-se uma listagem com mensagens assinalando os erros detectados, e dando origem a um arquivo intermediário com as especificações de forma codificada. Em seguida essas especificações são interpretadas pelo gerador de texto produzindo os vários segmentos de texto que irão compor o programa COBOL. Finalmente os segmentos de texto produzidos são concatenados na ordem especificada, gerando-se o programa desejado.

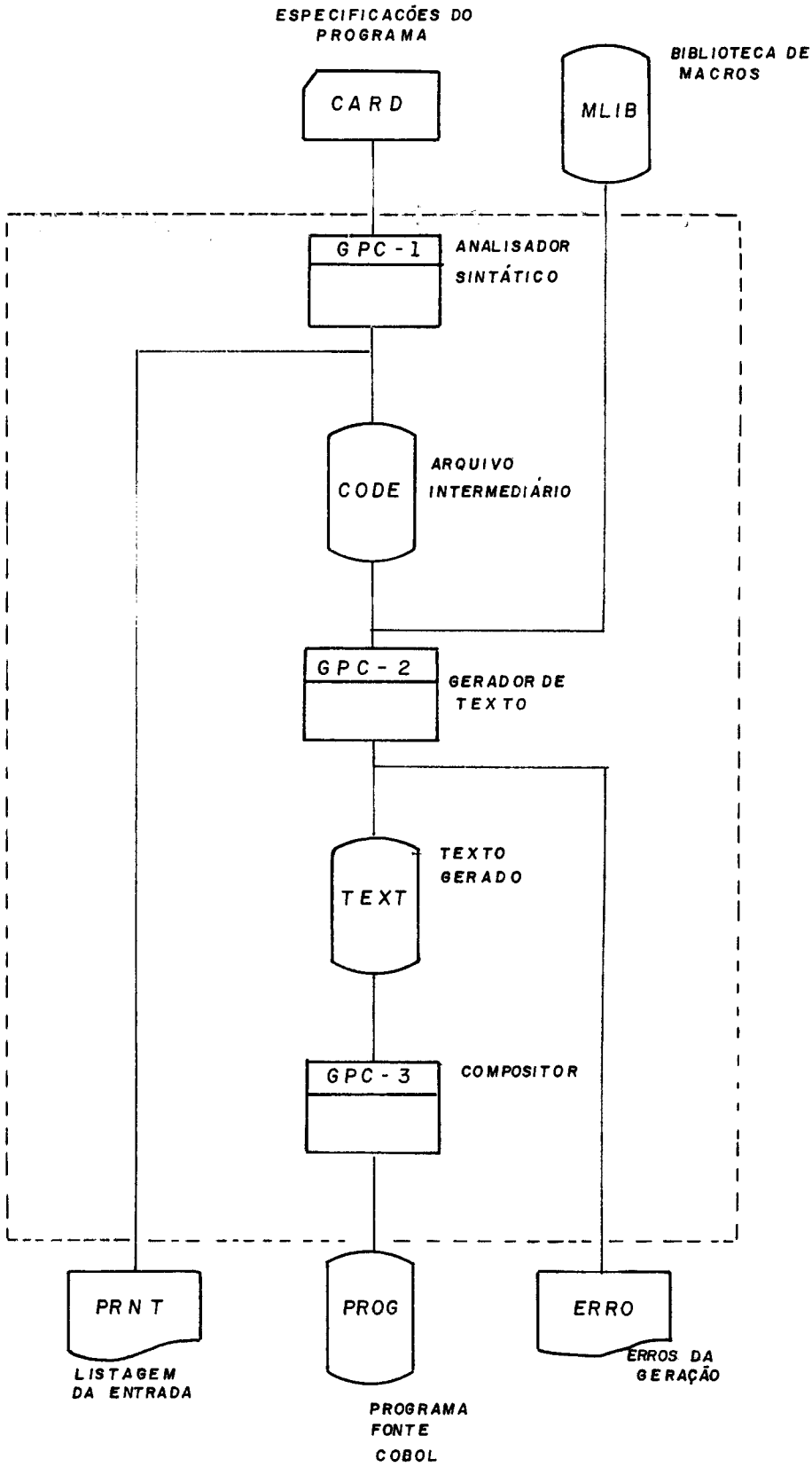


FIG. 4.2-2 ROTINA DE GERAÇÃO DE PROGRAMAS

4.3. A linguagem do GPC

A especificação de um programa a ser gerado pelo GPC, assim como as definições das macros a serem utilizadas na geração dos programas, é feita através de uma linguagem, especialmente projetada para este fim. Esta linguagem assemelha-se a uma linguagem de procedimentos, tal como o COBOL, contendo um número reduzido de elementos que serão apresentados a seguir.

Conjunto de Caracteres Utilizados

a) Caracteres utilizados para formação de palavras reservadas e identificadores:

   letras A a Z

   dígitos 0 a 9

   hífen -

b) Caracteres usados para pontuação das instruções do GPC

   b - espaço

   , - vírgula

   = - sinal de igual

   (- abre parêntesis

) - fecha parêntesis

   . - ponto

   / - barra oblíqua

   * - asterisco

c) Delimitadores especiais

   & - e comercial, que assinala a ocorrência de uma parâmetro formal

   " - aspas, utilizada para delimitar literais não numéricos

   ≠≠ - "jogo da velha", utilizada para delimitar sequencias de

caracteres tratadas como texto simplesmente

% - percentagem, utilizado para assinalar os registro que contêm instruções ao GPC

Regras Básicas de Codificação

O formulário utilizado deve ser o mesmo adotado pelo COBOL, que possui os seguintes campos:

de 1 a 6 - sequencial

7 - campo de continuação

de 8 a 11 - área A

de 12 a 72 - área B

de 73 a 80 - identificação do programa.

O campo de continuação é utilizado para distinguir os registros que contêm instruções ao GPC, que são assinalados como o símbolo % nessa posição, dos registros de texto matriz que podem conter qualquer outro símbolo nessa posição.

A codificação das instruções ao GPC pode ser feita livremente, utilizando-se as colunas 8 a 72 (áreas A e B), indistintamente.

A codificação do texto matriz deve obedecer às regras definidas pelo COBOL, respeitando-se a divisão de áreas A e B.

Nas instruções ao GPC os espaços podem ser omitidos, exceto entre palavras reservadas ou identificadores consecutivos. Por outro lado, os espaços podem ser usados livremente, entre os vários símbolos, melhorando-se a legibilidade do texto.

Continuação de Registros

Uma mesma instrução ao GPC pode ocupar 2 ou mais registros, bastando para isto que:

- todos os registros estejam assinalados com o símbolo % no campo de continuação;

- caso a continuação de registro provoque a divisão de um literal em duas ou mais partes, a continuação se fará segundo as regras do COBOL, à exceção do hífen no campo de continuação que será ocupado pelo % . Por exemplo:

```
COLUNA   7                                          73
     1   %    ...       ...      ,MENS = "DIGITO DE
     2   %             "CONTROLE ERRADO", ...
```

- a continuação de um texto, entre ≠≠'s, é feita de forma análoga;

- palavras chaves e identificadores devem estar, obrigatóriamente, inteiramente contidas num único registro.

Inclusão de Comentários

As instruções ao GPC podem receber comentários ao final de cada registro, desde que precedidos de um segundo símbolo % , que não esteja contido em literal ou texto.
Por exemplo:

```
     1   %   MACRO FILE      % DEFINIÇÃO DE ARQUIVO
     2   %       , PARM=(NAME  % FILE-NAME
     3   %   ...               %
```

Descrição da Sintaxe

A linguagem do GPC está definida através de 27 regras sintáticas, numeradas de 01 a 27. Da regra 01, que define a sintaxe do texto de entrada como um todo, derivam direta ou indiretamente, todas as demais regras sintáticas.

Cada regra sintática é composta por:

i) um número que a identifica;

ii) um símbolo não-terminal, delimitada por 'brackets' (< e >) que representa a entidade cuja sintaxe se está definindo;

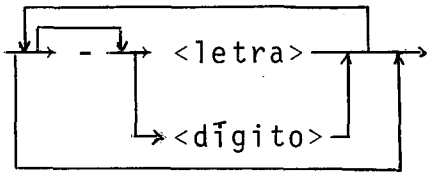
iii) o sinal ::=(lê-se "é definido como") que separa o símbolo não-terminal da sua definição sintática

iv) uma rede composta por símbolos terminais e não-terminais interligados por setas orientadas (→), que constitui a definição sintática. Os símbolos terminais aparecem na definição na forma como devem ser escritos no texto de entrada para o GPC. As setas orientadas indicam a ordem em que os diversos símbolos devem ser escritos. Bifurcações ($\downarrow$→) representam opções alternativas sintáticamente válidas.

Os símbolos terminais %(percentagem) e .(ponto) aparecem sublinhados para indicar o início e o fim, respectivamente, de uma instrução ao GPC, devendo o % ser codificado no campo de continuação dos registros em que estiver contida.

v) uma lista de números que indicam as regras sintáticas derivadas, que definem os símbolos não-terminais utilizados na definição.

Exemplo:

24 <identificador>::=→<letra> 

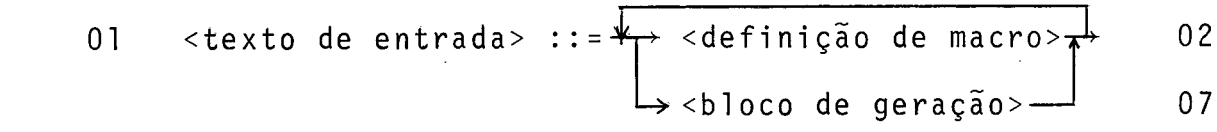
25
26

Significado:

Um <identificador> é formado por uma <letra> que pode ser seguida por uma sequência de <letra>'s, <digitos>'s e hífens (-), terminando em <letra > ou <dígito > . As regras 25 e 26 definem <letra> e <dígito> respectivamente.

O restante desta seção contém a especificação da sintaxe da linguagem do GPC, utilizando-se a notação acima descrita.

TEXTO DE ENTRADA



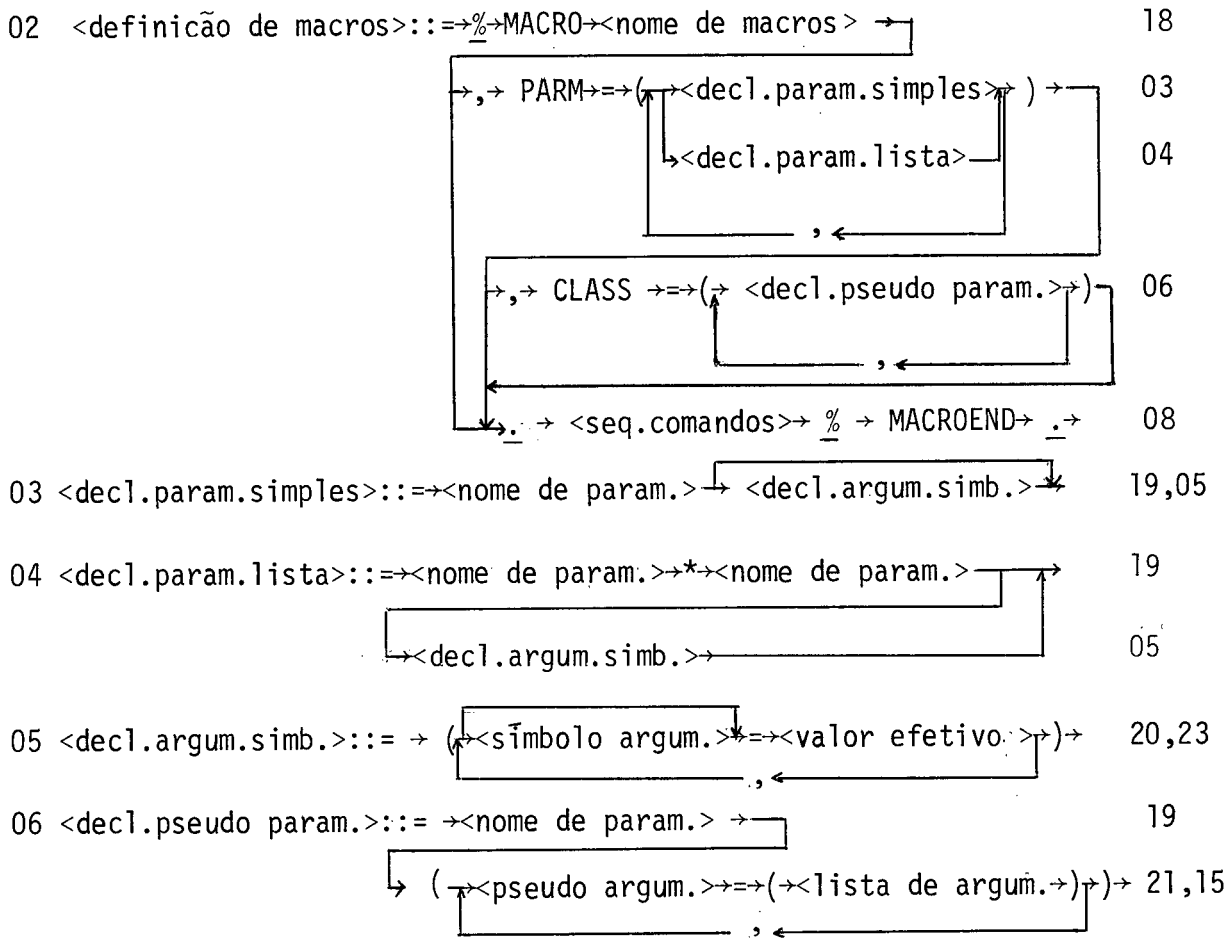
O texto de entrada para o GPC é composto por definições de macros e/ou blocos de geração.

O processo de geração de programas é comandado pelos blocos de geração, que especificam o programa a ser gerado. As definições de macros especificam ações do gerador a serem seguidas quando, no decorrer do processo de geração ocorrer uma chamada da macro.

As especificações de um programa podem estar contidas em mais de um bloco de geração, o que permite a utilização de parâmetros independentes em cada bloco. Por exemplo: um programa que atualize um arquivo e simultaneamente emite um relatório com dados do arquivo atualizado pode ser especificado em duas partes - a atualização do arquivo e a emissão do relatório.

As definições de macros, quando incluídas no texto de entrada para geração de programa, tem precedência sobre as definições homonimas existentes na biblioteca de macros.

DEFINIÇÃO DE MACRO



Exemplos:

i) Macro sem parâmetros.

```
1   %   MACRO M1.
2       ...
3   %   MACROEND.
```

ii) Macro com parâmetros simples:

```
4   %   MACRO M2, PARM=(PS1, PS2, PS3).
5       ...
6   %   MACROEND.
```

iii) Macro com parâmetro simples e argumentos simbólicos.

```
7   %   MACRO M3, PARM=(PS1(A1= ≠≠xxx≠≠,A2="yyy",A3=123)).
8       ...
9   %   MACROEND.
```

iv) Macro com parâmetros simples,lista e pseudo parâmetro.

```
10  %   MACRO M4, PARM=(PS1, PS2, PL1*N1, PL2*N2)
11  %             , CLASS=(PP(PA1=(PS1= ≠≠xxx≠≠, PS2="yyy")
12  %                        ,PA2=(PS2=123,PL2=(A,B,C)))).
13      ...
14  %   MACROEND.
```

Finalidade

Especificar as ações do gerador ao encontrar uma chamada da macro.

Semântica

O cabeçalho da macro contém, obrigatóriamente, o nome da macro e, opcionalmente, uma declaração de parâmetros da macro.

Os parâmetros efetivos da macro, que representam no corpo da macro os argumentos a serem fornecidos na chamada da macro,podem ser de dois tipos: simples, quando admite um único valor como argumento, ou lista, quando admite uma lista de valores. Os parâmetros do tipo lista são definidos em conjunto com um parâmetro simples que representa o número de valores na lista associada ao parâmetro,na chamada da macro.

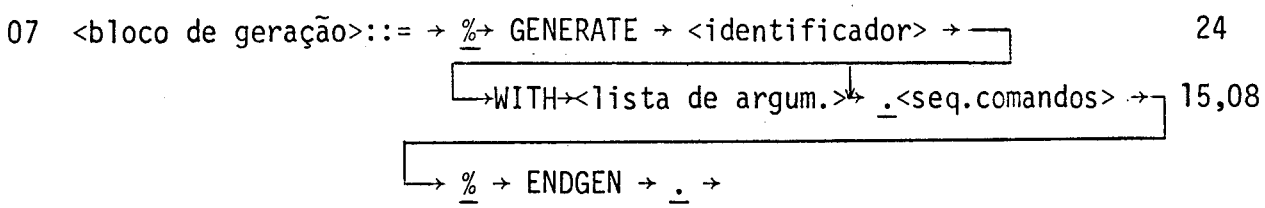
A cada parâmetro efetivo pode ser associada uma lista de argumentos simbólicos, cada qual consistindo na definição de um identificador, ou a omissão de qualquer valor, para simbolizar um valor pré-definido, nas chamadas da macro.

Os pseudo-parâmetros são utilizados para substituir, na chamada da macro, a atribuição de vários argumentos relacionados entre si por uma única atribuição de um pseudo-argumento que lhe é equivalente.

Uma explicação mais detalhada, com exemplo, da utilização de argumentos simbólicos e pseudo parâmetros encontra-se na descrição da macro chamada (regra nº 10).

O corpo da macro consiste numa sequência de comandos de geração a serem seguidos pelo gerador.

BLOCO DE GERAÇÃO



Exemplos:

i) Bloco de geração sem parâmetros

```
1    %   GENERATE G1.
2        ...
3    %   ENDGEN.
```

ii) Bloco de geração com parâmetros

```
4    %   GENERATE G2 WITH P1= ≠≠xxx≠≠, P2=123, P3=(A, B, C).
5        ...
6    %   ENDGEN.
```

Finalidade

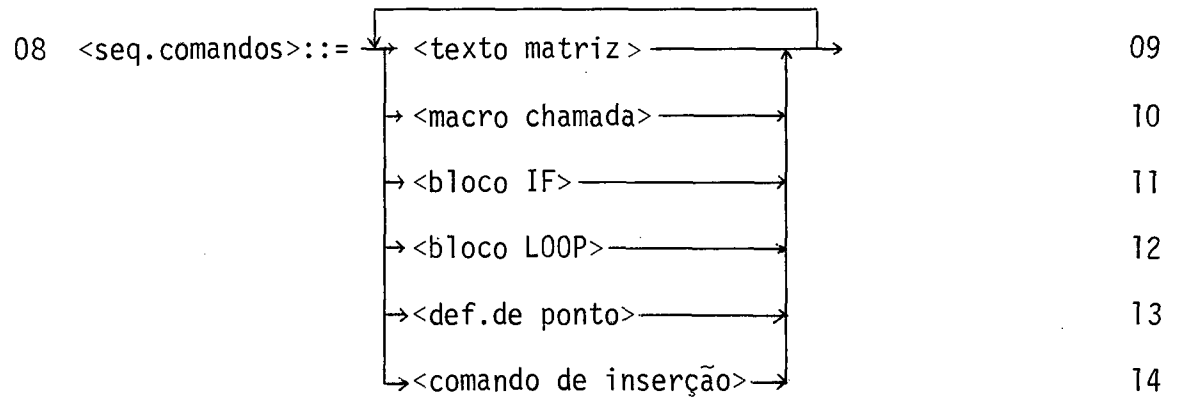
Especificar o programa a ser gerado.

Semântica

O cabeçalho do bloco de geração contém, obrigatoriamente, um identificador do bloco e, opcionalmente, uma lista de argumentos que define os parâmetros formais do bloco ao mesmo tempo que lhes atribui os valores a serem utilizados na geração.

O corpo do bloco de geração contém uma sequência de comandos a serem seguidos pelo gerador.

SEQUÊNCIA DE COMANDOS



Finalidade

Especificar as ações a serem seguidas pelo gerador.

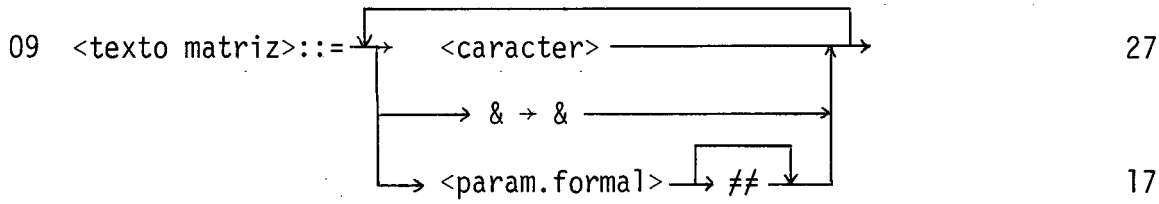
Semântica

O processo básico de geração de texto consiste na reprodução de um texto matriz na saída, substituindo-se os parâmetros formais aí contidos pelos valores a eles associados.

Os comandos de macro chamada, bloco IF e bloco LOOP alteram o curso normal do processo de geração (sequencialmente), seja desviando-o para outra sequência de comandos (macro chamada), seja selecionando os comandos a interpretar (bloco IF) ou repetindo uma sequência de comandos especificada(bloco LOOP).

Os comandos de definição de ponto e de inserção controlam a ordenação relativa dos vários segmentos de textos gerados.

TEXTO MATRIZ



Exemplo:

```
1       ROT-LE-&NAME.
2           READ &NAME INTO WS-REG-&NAME
3           AT END MOVE 1 TO IND-EOF-&NAME.
4       ROT-LE-&NAME≠≠-EXIT.
5           EXIT.
```

Finalidade

Especificar o conteúdo do texto a ser gerado.

Semântica

O texto matriz é composto por sequências de caracteres, que são transcritos diretamente no texto gerado, e parâmetros formais (iniciados por &) que são substituídos pelos valores a eles associados. O caracter ≠≠ é utilizado para se indicar o término de um parâmetro formal, quando este é seguido por uma sequência de caracteres iniciada por letra, dígito ou hífen.

Um duplo & é utilizado para indicar uma ocorrência desse caracter no texto matriz, a ser transcrito no texto gerado, não iniciando, portanto, um parâmetro formal.

Restrições

O campo de continuação (coluna 7) dos registros de texto matriz não pode conter o caracter % , que é reservado, nesta posição, para indicar registros de instruções ao GPC.

MACRO CHAMADA

10 <macro chamadas>::=→%→<nome de macro>→WITH→<lista de argum.>→.→ 18,15

Exemplos:

i) chamada de macro sem lista de argumentos.

% M1.

ii) chamada de macro com lista de argumentos:

% M2 WITH P1=123, P2="ABC", P3=(X, Y, Z).

Finalidade:

Comandar a expansão de uma macro.

Semântica:

Ao encontrar um comando de macro chamada o gerador suspende o processo de geração em curso, interpreta os comandos de geração armazenados na definição da macro e em seguida reinicia o processo interrompido, voltando ao comando seguinte à chamada da macro.

A identificação da macro a ser expandida é feita através do seu nome, e que inicia a macro chamada. A busca da definição da macro é feita inicialmente no conjunto de definições do texto de entrada e, não sendo aí encontrada, é consultada a biblioteca de macros.

Uma vez identificada a macro, e obtida a sua definição, o passo seguinte é estabelecer o valor efetivo dos

argumentos a serem utilizados na macro expansão. Isto é feito com base na lista de argumentos fornecidas na chamada da macro e nas declarações de parâmetros, argumentos simbólicos e pseudo parâmetros da definição da macro. O exemplo abaixo ilustra esse processo.

(a) Definição da macro FILE.

```
1    %   MACRO FILE , PARM= (NAME, LRECL , BLOCK(=1)
2    %                      ,LABEL(= STANDARD, 0=OMITTED))
3    %               , CLASS=(DEV(CARD=(LRECL=80,LABEL=0)
4    %                           ,PRNT=(LRECL=132,LABEL=0))).
5        ...
6    %   MACROEND.
```

(b) Chamada da macro FILE.

```
7    %   FILE WITH NAME=CARTÃO, DEV=CARD.
```

Transformação da lista de argumentos:

1) Substituição do pseudo-argumento DEV=CARD pela lista de argumentos correspondente (ver linha 3, na definição da macro), produzindo:

```
%   FILE WITH NAME=CARTAO, LRECL=80, LABEL=0.
```

2) Inclusão do parâmetro BLOCK, completando a lista de argumentos:

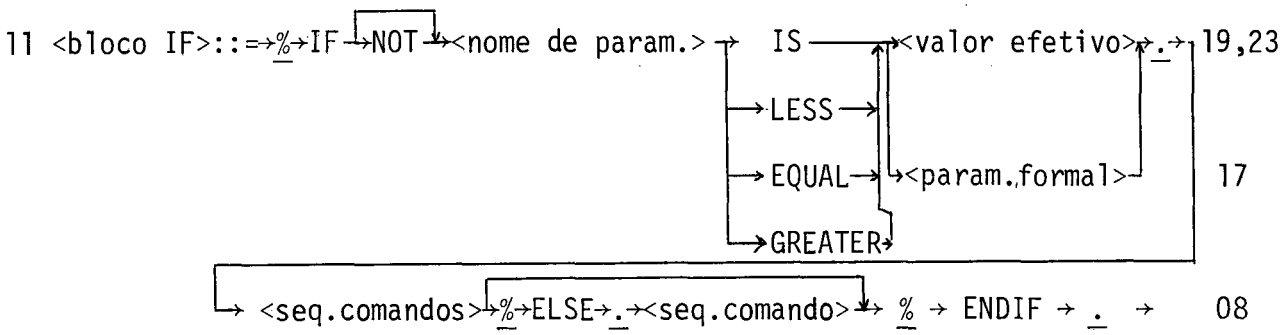
```
%   FILE WITH NAME=CARTAO,LRECL=80,LABEL=0,BLOCK=.
```

3) Substituição dos argumentos simbólicos LABEL=0 e BLOCK= pelos valores efetivos correspondentes (ver linha 1 e 2. na definição da macro), produzindo a lista de argumentos final:

```
%    FILE WITH NAME=CARTAO,LRECL=80,LABEL=OMITTED,BLOCK=1.
```

Após o estabelecimento dos valores efetivos de todos os argumentos da macro, o processo de geração passa a guiar-se pela sequencia de comandos da definição da macro. Ao final o controle do processo retorna à instrução seguinte à macro chamada, restabelecendo-se os argumentos anteriormente vigentes.

BLOCO IF



Exemplos:

i) IF-ENDIF

```
1   %   IF  NOT  P1  IS ≠≠SIM≠≠.
2       ...
3   %   ENDIF.
```

ii) IF-ELSE-ENDIF

```
4   %   IF  P1  LESS  &P2.
5       ...
6   %   ELSE.
7       ...
8   %   ENDIF.
```

Finalidade

Comandar a seleção, durante o processo de geração, de uma sequência de comandos a seguir, em função dos argumentos fornecidos.

Semântica:

a) 1a. Opção (instruções IF-ENDIF)

Ao encontrar a instrução IF o gerador desenvolve a expressão lógica a ela associada, utilizando os valores correntes dos parâmetros envolvidos. Caso o resultado obtido seja 'verdadeiro' o processo de geração prossegue normalmente, desprezando-se a instrução ENDIF correspondente. Caso contrário o processo é suspenso temporariamente, reiniciando após a instrução ENDIF correspondente.

b) 2a. Opção (instruções IF-ELSE-ENDIF).

Ao encontrar a instrução IF o gerador desenvolve a expressão lógica a ela associada, utilizando os valores correntes dos parâmetros envolvidos. Caso o resultado obtido seja 'verdadeiro' o processo de geração prossegue normalmente, até a instrução ELSE correspondente, sendo então suspenso temporariamente, reiniciando após a instrução ENDIF correspodente. Caso contrário o processo de geração é suspenso logo após a instrução IF, reiniciando após a instrução ELSE.

Operadores lógicos

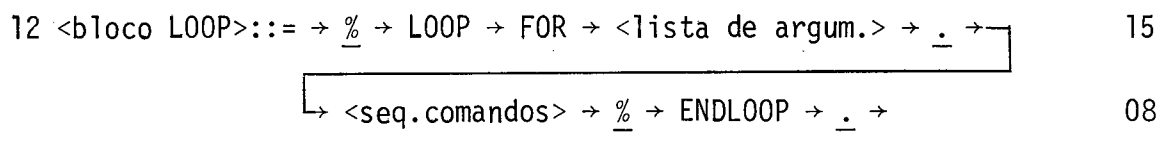
O operador IS destina-se a testar a identidade de duas sequências de caracteres quaisquer (identificadores, números inteiros, literais ou texto).

Os operadores LESS, EQUAL e GREATER destinam-se a comparações numéricas somente, não se aplicando a identificadores, literais ou textos.

Exemplos:

```
%   GENERATE EX WITH &P1= ≠≠SIM≠≠, &P2=123, &P3=0123
    ... IF P1 IS SIM ...              → Verdadeiro
    ... IF NOT P1 IS ≠≠SIMbbb≠≠       → Verdadeiro
    ... IF P2 EQUAL &P3 ...           → Verdadeiro
    ... IF P2 IS &P3 ...              → Falso
```

BLOCO LOOP



Exemplo:

```
1   %   LOOP  FOR  P1=(A, B, C), P2=(1, 2, 3).
2       ...
3   %   ENDLOOP.
```

Finalidade:

Comandar a repetição de uma mesma sequência de comandos, variando-se a cada vez os valores atribuidos a uma lista de parâmetros.

Semântica:

Ao encontrar uma instrução LOOP o gerador acrescenta aos parâmetros correntes aqueles definidos na instrução,que são os parâmetros do LOOP. Cada parâmetro do LOOP está associado a uma lista de valores, utilizando-se como argumento cada um dos valores da lista para cada ciclo em que são interpretados os comandos do LOOP. O número de ciclos a executar é determinado pelo tamanho da maior lista associado aos parâmetros do LOOP. Havendo listas de tamanho inferior, estas são percorridas circularmente enquanto houver ciclos a executar.

Ao final do último ciclo os parâmetros do LOOP

são desativados, permanecendo apenas aquelas anteriormente vigentes.

DEFINIÇÃO DE PONTO

13 <def.de ponto>::= → % → DEF → <ident.ponto inserção> → . → 22

Finalidade:

Definir, no texto que está sendo gerado, um novo ponto onde poderá ser comandada a inserção de outros segmentos de texto.

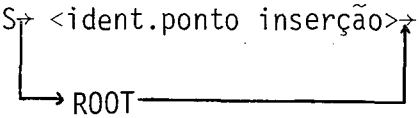
Semântica:

Ao encontrar um comando DEF o gerador associa à identificação do ponto a sua posição relativa no segmento de texto que está sendo gerado. Ao final do processo de geração de texto os segmentos de texto endereçados a esse ponto são então inseridos naquela posição relativa.

Uma explicação detalhada, com exemplo, do processo de composição de texto encontra-se na descrição do comando de inserção (regra nº 14).

COMANDO DE INSERÇÃO

14 <comando de inserção> ::= → % → INS → <ident.ponto inserção> → . → 22



Finalidade:

Especificar ao gerador a posição relativa onde será inserido o segmento de texto gerado pelos comandos subsequentes.

Semântica

Ao encontrar um comando INS o gerador inicia um novo segmento de texto, endereçando-o ao ponto de inserção identificado. Ao final do processo de geração, na fase de composição do texto gerado, os vários segmentos de texto são concatenados na ordem especificada.

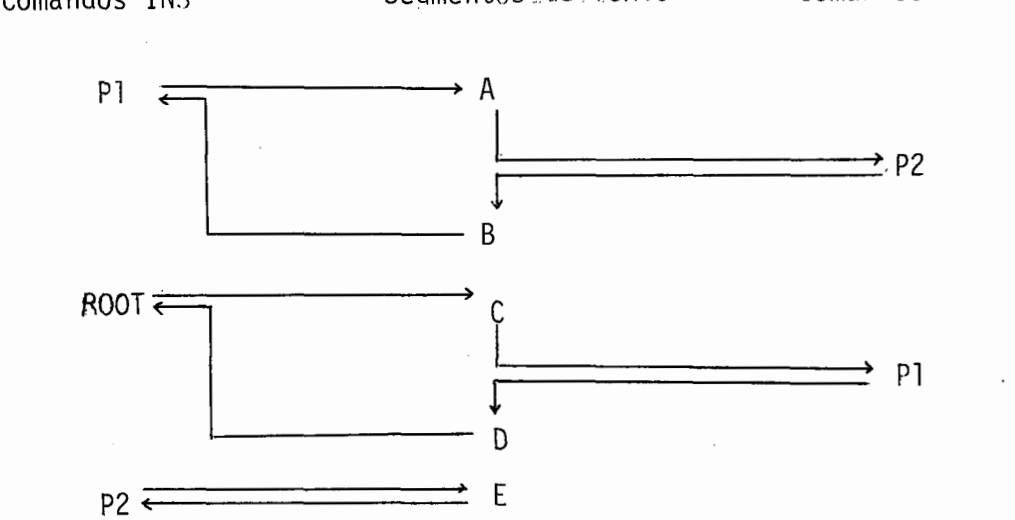
Exemplo:

(a) Texto de Entrada para o Gerador

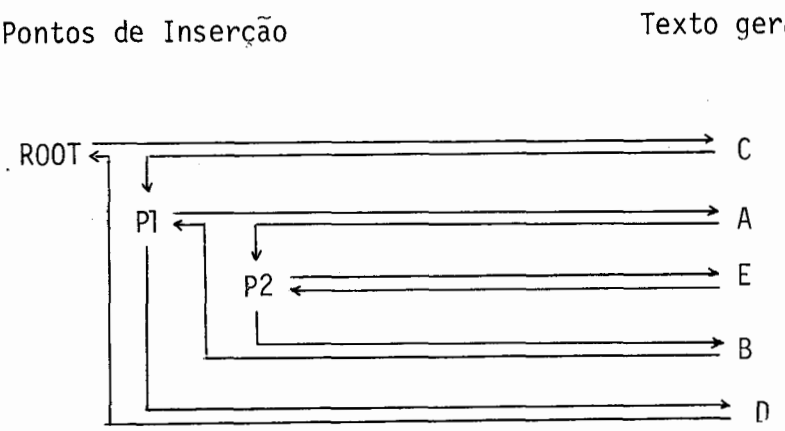
```
1    %    GENERATE EXEMPLO.
2    %    INS  P1.
3           =TEXTO COBOL 'A'=
4    %    DEF P2.
5           =TEXTO COBOL 'B'=
6    %    INS ROOT.
7           = TEXTO COBOL 'C'=
```

```
 8      %      DEF P1.
 9               = TEXTO COBOL 'D'=
10      %      INS P2.
11               = TEXTO COBOL 'E' =
12      %      ENDGEN.
```

(b) representação gráfica da estrutura gerada:



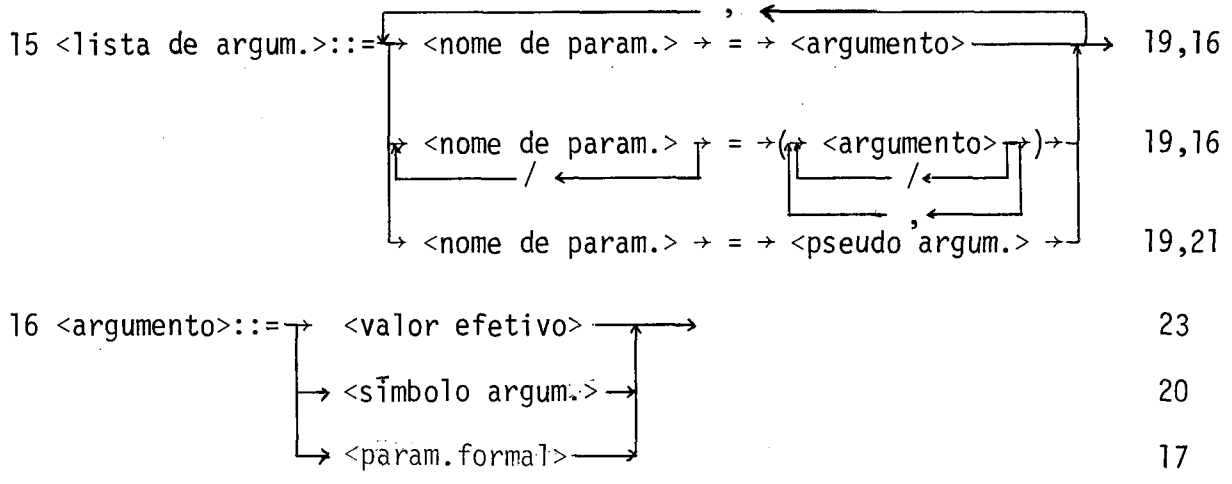
(c) ordenação dos segmentos de texto, iniciando-se pelos segmentos inseridos em 'ROOT':



Deve-se observar, através deste exemplo, que a definição de um ponto de inserção não deve, necessàriamente, preceder os comandos de inserção no ponto.

Em um mesmo ponto de inserção pode ser comandada a inserção de vários segmentos de texto, em diferentes pontos do texto de entrada, que são concatenados, no programa COBOL, na mesma ordem em que são gerados.

LISTA DE ARGUMENTOS



Exemplos:

i) lista de argumentos simples:

```
1   %   ... P1=123, P2=S, P3=&X ...
```

ii) lista de argumentos com listas de valores:

```
2   %   P1=(A, B, C), P2=(0, 2, 4) ...
```

iii) lista de argumentos com listas de valores combinadas:

```
3   %   ... P1/P2 = (A/0, B/2, C/4) ...
```

Finalidade:

Designar os argumentos a serem utilizados na geração.

Semântica:

A cada parâmetro identificado à esquerda do sinal = é atribuido um valor ou lista de valores, que aparecem à direita do sinal = correspondente. Duas ou mais listas

de valores de mesmo tamanho podem ser combinadas, reunindo-se à esquerda de um único sinal = os nomes dos parâmetros separados por / , e à direita uma lista única com valores também separados por / e dispostos na mesma ordem.

PARÂMETRO FORMAL

17 <param.formal>::= → & → <nome de param.> → 19

Exemplos:

&NAME

&BLOCK

Finalidade:

Representar o valor corrente do parâmetro identificado.

NOME DE MACRO

18 <nome de macro>::= → <identificador> → 24

Exemplos:

PROG

FILE

Finalidade:

Identificar a macro que se está definindo ou chamando.

NOME DE PARÂMETRO

19 <nome de param.>::= → <identificador> → 24

Exemplos:

NAME

BLOCK

Finalidade:

Identificar os parâmetros de um bloco de geração, macro definição, ou bloco LOOP.

Restrições:

Não pode haver duplicidade de nomes de parâmetros no mesmo bloco de geração ou definição de macro, incluíndo-se os parâmetros dos blocos LOOP neles contidos.

SÍMBOLO ARGUMENTO

20 <símbolo argum.> ::= → <identificador> → 24

Exemplos:

S

0

Finalidade:

Representar, quando atribuído ao parâmetro da macro ao qual está associado, um valor efetivo a ser utilizado como argumento.

Restrições:

Os símbolos argumento de um mesmo parâmetro de macro devem estar univocamente identificados.

PSEUDO ARGUMENTO

21 <pseudo argum.> ::= → <identificador> → 24

Exemplo:

CARD

PRNT

Finalidade:

Representar, quando atribuido ao pseudo parâmetro da macro ao qual está associado, uma lista de argumentos a ser utilizada na geração.

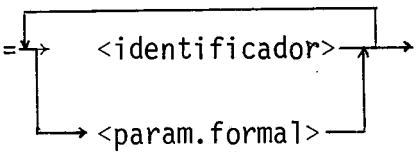
Restrições:

Os pseudo argumentos de um mesmo pseudo parâmetro devem estar univocamente identificados.

IDENTIFICAÇÃO DE PONTO DE INSERÇÃO

22 <ident.ponto inserção> ::= → <identificador> → 24

→ <param.formal> → 17



Exemplo:

```
SELECT
WORKING &SECTION
PROC  &SECTION INICIO
```

Finalidade:

Identificar os vários pontos de inserção de um programa.

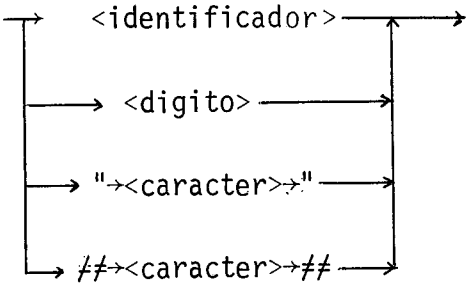
Restrições:

O comprimento final da identificação do ponto de inserção, após a substituição dos parâmetros formais pelos valores correspondentes, não poderá exceder 60 caracteres.

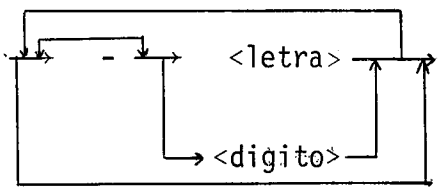
Na formação da identificação do ponto de inserção não são permitidos parâmetros formais que representam listas de valores.

VALOR EFETIVO

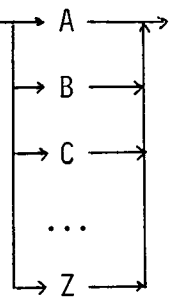
23 <valor efetivo> ::=



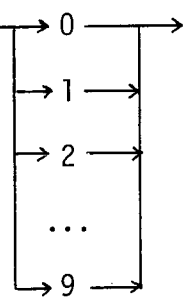
24 <identificador> ::=



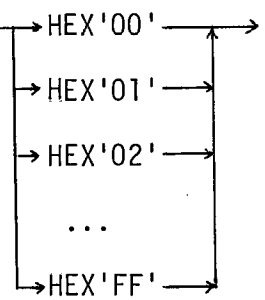
25 <letra> ::=



26 <digito> ::=



27 <caracter> ::=



Exemplos:

ARQ-1	(identificador)
132	(número inteiro)
" *** TOTAIS - "	(literal)
≠≠ zz.zz9,99≠≠	(texto)

Finalidade:

É a unidade básica que pode ser associada a um parâmetro formal.

Restrições:

Os limites máximos, em número de caracteres, para cada uma das classes de valores são:

identificador	→	30
nº inteiro	→	18
literal	→	120
texto	→	200

Os identificadores não podem pertencer à lista de palavras reservadas do GPC, apresentada a seguir:

CLASS	EQUAL	INS	MACROEND
DEF	FOR	IS	NOT
ELSE	GENERATE	LESS	PARM
ENDGEN	GREATER	LOOP	ROOT
ENDIF	IF	MACRO	WITH
ENDLOOP			

CAPÍTULO V

IMPLEMENTAÇÃO DO GPC

5. Implementação do GPC

5.1. Introdução

A Figura 5.1 apresenta o interrelacionamento dos vários componentes do GPC, agrupados em:

a) Módulo de Atualização da Biblioteca de Macros, constituído pelos programas GPC-1 (Analisador Sintático) e GPC-4 (Atualizador); e

b) Módulo de Geração de Programas, constituído pelos programas GPC-1 (Analisador Sintático), GPC-2 (Gerador de Texto) e GPC-3 (Compositor).

As seções seguintes deste capítulo descrevem detalhadamente os vários arquivos e programas envolvidos.

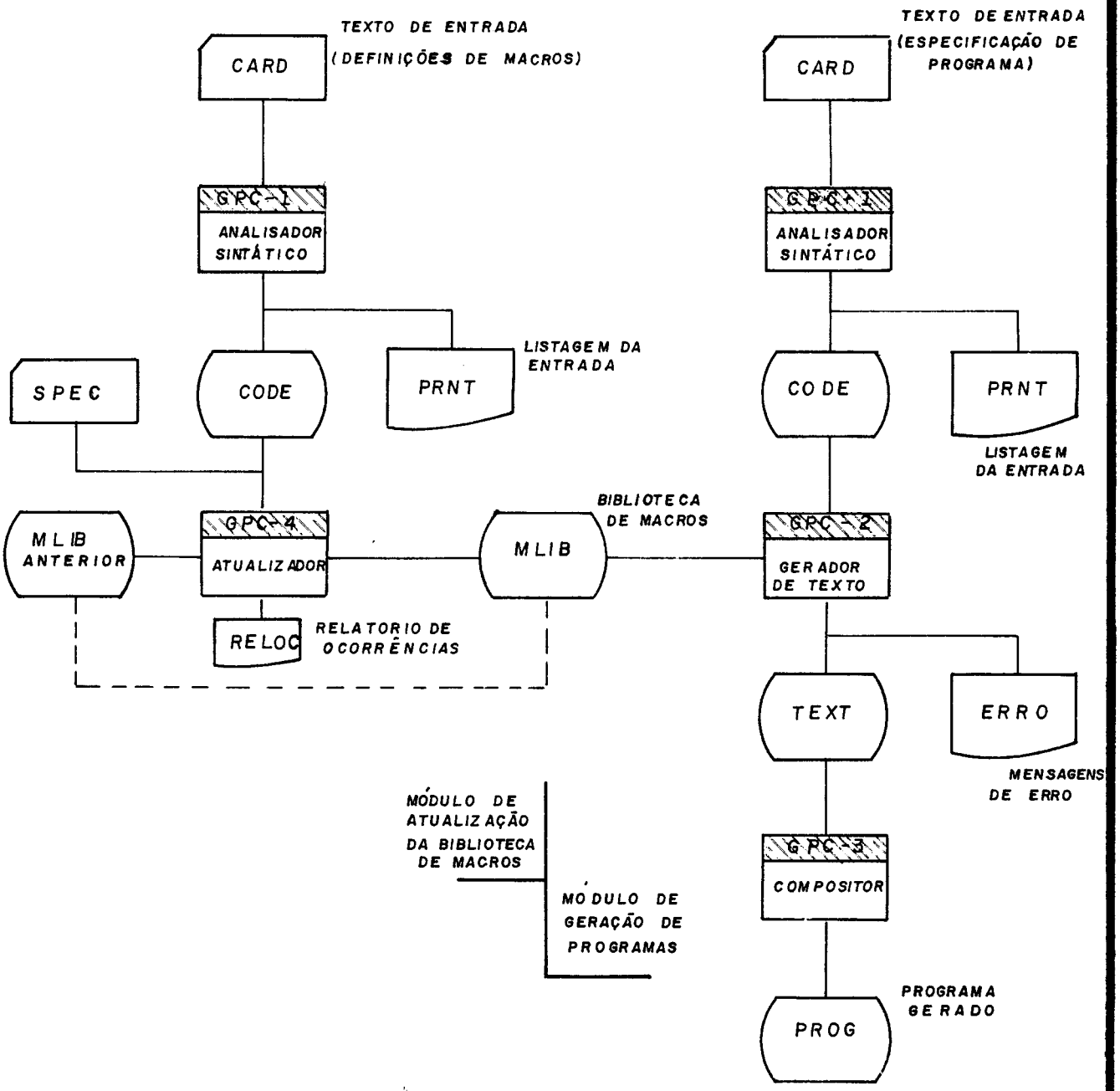


FIG. 5.1 - FLUXO LÓGICO DO GPC

5.2. Descrição dos Arquivos Utilizados pelo GPC

5.2.1. Arquivo CARD

a) Conteúdo - Contém o texto de entrada fornecido pelo programador, podendo ser:

i) Definições das Macros a serem incluídas ou substituidas na Biblioteca de Macros pela Rotina de Atualização da Biblioteca de Macros; ou

ii) Especificação do Programa a ser gerado pela Rotina de Geração de Programas.

b) Meio - cartões perfurados

c) Organização - sequencial

d) Modo de Gravação - registros de comprimento fixo

e) Comprimento dos Registros - 80 bytes

f) Fator de Bloco - 1

g) Tipos de Registros - possui 1 único tipo de registro - Registro Texto de Entrada, descrito a seguir:

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 6	sequencial do registro	SEQIN	x(6)
7 a 72	texto de entrada	TEXT	x(66)
73 a 80	espaço livre para identificação	IDENT	x(8)

Uma explicação detalhada do conteúdo desses registros encontra-se no Capítulo 4 deste trabalho.

5.2.2. Arquivo CODE

a) Conteúdo - Contém as instruções ao Gerador de Texto, de forma codificada, equivalentes ao texto de entrada fornecido pelo programa.

b) Meio - Disco Magnético

c) Organização - Direta

d) Modo de Gravação - registros de comprimento fixo

e) Comprimento dos Registros - 2048 bytes

f) Fator de Bloco - 1

g) Tipos de Registros - possui três diferentes tipos de registros que são:

i) Rótulo do Arquivo - é o 1º registro do arquivo, e contém os parâmetros básicos do arquivo.

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 5	Chave do Registro (00001)	RKEY	9(5)
6 a 6	Tipo de Registro (1)	TPREG	9
7 a 11	Chave do 1º Registro Índice	PRIMRINDEX	9(5)
12 a 16	Chave da Última Reg.Índice	ULTRINDEX	9(5)
13 a 2048	Espaço não utilizado	FILLER	x(2032)

ii) Registros Índice - contém um índice das definições de macros e dos blocos de geração existentes no arquivo.

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 5	Chave do Registro	RKEY	9(5)
6 a 6	Tipo de Registro (2)	TPREG	9
7 a 8	Número de Entradas na Tabela do Registro (máx 53)	IXMAX	99
9 a 2024	Tabela contendo:		
	-Tipo,podendo ser: 1→Bloco de Geração 2→Definição de Macro	TP	9
	-Nome da macro(se tipo=1)	NOME	x(32)
	-Chave inicial dos Registros de Instruções de Geração	PRIMRINS	9(5)
2025 a 2048	Espaço não utilizado	FILLER	x(26)

As entradas correspondentes aos Blocos de Geração precedem às de Definição de Macros,as quais são classificadas em ordem crescente de nome.

iii) Registro de Instruções de Geração - contém as instruções codificadas, no seguinte formato geral:

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 5	Chave do Registro	RKEY	9(5)
6 a 6	Tipo de Registro (3)	TPREG	9
7 a 10	Comprimento total das instruções no registro	TOTL	9(4)

11 a(10 + 'TOTL')	Sequencia de instruções codificadas,contendo		
	-Comprimento da Instrução	INSTRL	999
	-Sequencial do Registro Correspondente no Texto de Entrada	SEQIN	x(6)
	-Código da Instrução	OP	99
	-1º operando	N	99
	-2º operando	I	9
	-3º operando	J	9
	-Comprimento do 'String'	L	999
	-'String'	S	X x 'L'
(11+)a 2048 'TOTL'	- Espaço não utilizado	FILLER	

Os códigos de operação (OP) válidos estão listados a seguir, juntamente com o seu significado e dos operandos que utiliza.

OP	Instrução	Operandos	
01	Início de Bloco de Geração	-	não tem
02	Parâmetro de Bloco de Geração	N	nº atribuído ao parâmetro
		I	tipo de parâmetro simples → 1 lista → 2
		J	tipo de valor atribuído ao parâmetro identificador → 1 nº inteiro → 2 literal → 3 texto → 4
		L	comprimento do 'string'
		S	'string'

03	Término dos parâmetros de Bloco de Geração	-	não tem
04	Texto Matriz	J	tipo: texto COBOL → 1 parâmetro formal → 2
		L	se texto COBOL:comprimento do 'string' se parâmetro formal:nº do parâmetro
		s	'string' (se texto COBOL)
05	Término de registro texto matriz	-	não tem
06	Definição de ponto de inserção(identificação)	J	tipo: identificador → 1 parâmetro formal → 2 root → 3
		L	se idenficador:comprimento do identificador se param.formal: nº do parâmetro
		s	identificador(se identificador)
07	Término de definição de ponto de inserção	-	não tem
08	Comando de Inserção (identificação)		idem OP=06
09	Término de comando de inserção	-	não tem
10	Chamada de macro(nome da macro)	L	comprimento do nome
		s	nome
11	Chamada de macro(nome de parâmetro)	N	nº do argumento na lista
		L	comprimento do nome
		s	nome
12	Chamada de macro(argumento)	N	nº do argumento na lista
		I	tipo de argumento: simples → 1 lista → 2

		J	tipo de valor: identificador → 1 nº inteiro → 2 literal → 3 texto → 4 parâmetro formal → 5
		L	se identificador,nº inteiro,literal contexto:comprimento do 'string' se parâmetro: nº do parâmetro
		s	'string'
13	Término da chamada de macro	-	não tem
14	Início de Bloco IF (condição lógica)	N	nº do parâmetro
		I	operador lógico: IS → 1 NOT IS → 2 LESS → 3 NOT LESS → 4 EQUAL → 5 NOT EQUAL → 6 GREATER → 7 NOT GREATER → 8
		J	
		L	idem OP = 12
		s	
15	Bloco IF(início de bloco ELSE)	-	não tem
16	Fim de Bloco IF	-	não tem
17	Início de Bloco LOOP	-	não tem
18	Parâmetro de Bloco LOOP		idem OP=02
19	Término dos parâmetros de Bloco LOOP	-	não tem
20	Fim de Bloco LOOP	-	não tem
21	Fim de Bloco Generate	-	não tem
22	Início de Definição de Macro (nome de macro)		idem OP=10
23	Declaração de Parâmetro Efetivo	N	nº atribuido ao parâmetro

		I	tipo de parâmetro: simples → 1 lista → 2 tam.lista → 3
		L	se param.simples ou lista:comprimento do nome se param.tam.lista:nº do parâmetro lista
		s	nome
24	Declaração de Argumento Simbólico(simbolo argumento)		idem OP=10
25	Declaração de Argumento Simbólico (argumento efetivo)	J	
		L	idem OP=12
		s	
26	Término dos Argumentos Simbólicos	-	não tem
27	Término dos Parâmetros Efetivos	-	não tem
28	Declaração de Pseudo-Parâmetro		idem OP=10
29	Declaração de Pseudo Argumento		idem OP=10
30	Argumento Equivalente		idem OP=02
31	Término da lista de argumentos	-	não tem
32	Término dos pseudo argumentos	-	não tem
33	Término dos pseudo parâmetros	-	não tem
34	Fim de Macro	-	não tem
35	Fim de Texto	-	não tem

5.2.3. Arquivo MLIB

a) Conteúdo - Contém as Definições de Macros de uso comum a todos os programas, na forma codificada.

b) Meio
c) Organização
d) Modo de Gravação
e) Comprimento dos Registros
f) Fator de Bloco
g) Tipos de Registros

} Especificações idênticas às do arquivo CODE (vide seção 5.2.2)

5.2.4. Arquivo TEXT

a) Conteúdo - Contém os vários segmentos de texto produzidos pelo gerador de texto, e uma tabela que especifica a ordenação lógica desses segmentos de texto

b) Meio - Disco Magnético

c) Organização - Sequencial

d) Modo de Gravação - registros de comprimento fixo

e) Comprimento dos Registros - 2048 bytes

f) Fator de Bloco - 1

g) Tipos de Registros - possui 3 diferentes tipos de registro, que são:

i) Rótulo do Arquivo - é o 1º registro do arquivo, e contém os parâmetros básicos do arquivo.

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 5	Chave do Registro (00001)	RKEY	9(5)
6 a 6	Tipo do Registro (1)	TPREG	9
7 a 11	Chave do 1º registro Tabela	PRIMTAB	9(5)
12 a 16	Chave do últ.reg.Tabela	ULTTAB	9(5)
13 a 2048	Espaço não utilizado	FILLER	X(2032)

ii) Registro Tabela - especifica a ordenação lógica dos vários segmentos de texto.

POSIÇÕES (BYTES)	CAMPO	MNEMONIO	PICTURE COBOL
1 a 5	Chave do Registro	RKEY	9(5)
6 a 6	Tipo do Registro (2)	TPREG	9
7 a 8	Número de Entradas na Tabela do Registro (máx.127)	IXMAX	999
9 a 2041	Tabela contendo:		
	-Ponteiro p/1º comando COBOL do segmento de texto	PRIMT	9(8)
	-Ponteiro p/último comando COBOL do segmento de texto	ULTT	9(8)
2042 a 2048	Espaço não utilizado	FILLER	x(7)

iii) Registro Segmentos de Texto - contém os vários segmentos de texto gerador

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 5	Chave do Registro	RKEY	9(5)
6 a 6	Tipo do Registro (3)	TPREG	9
7 a 10	Comprimento total dos comandos COBOL no registro	TOTL	9(4)
11 a (10+ 'TOTL')	Sequência de comandos COBOL, contendo:		
	-Comprimento do comando	L	999
	-Comando COBOL	SEQIN S	x(6) S x 'L'
(11+ 'TOTL') a 2048	Espaço não utilizado	-	-

5.2.5. Arquivo PROG

a) Conteúdo - Contém o programa COBOL gerado

b) Meio - Disco magnético

c) Organização - sequencial

d) Modo de gravação - registros de comprimento fixo

e) Comprimento dos Registros - 80 bytes

f) Fator de Bloco - 5

g) Tipos de Registros - possui um único tipo de registro - Registro Texto COBOL, descrito a seguir:

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 6	Sequencial do Registro	SEQOUT	9(6)
7 a 72	Comando COBOL	STMT	x(66)
73 a 78	Sequencial do Registro de Entrada Corresp.	SEQIN	x(6)
79 a 80	Espaço não utilizado	FILLER	xx

5.2.6. Arquivo SPEC

a) Conteúdo - parâmetros para atualização da Biblioteca de Macros.

b) Meio - cartões perfurados

c) Organização - sequencial

d) Modo de Gravação - registros de comprimento fixo

e) Comprimento dos Registros - 80 bytes

f) Fator de Bloco - 1

g) Tipos de Registros - possui um único tipo de registro - Parâmetros para Atualização, descrito a seguir:

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 3	Código de ação: Inclusão → INS Substituição → SUB Exclusão → EXC	CDACAO	xxx
4 a 80	Lista de nomes das macros a incluir, substituir ou excluir	NOMES	x(77)

5.3. Descrição dos Programas

5.3.1. Programa GPC-1

a) Objetivo - transformar o texto de entrada fornecido pelo programador numa sequência de instruções de geração codificadas.

b) Arquivos Utilizados

Entrada - Texto de Entrada (CARD)

Saídas - Arquivo Intermediário (CODE), e
Listagem do Texto de Entrada (PRNT).

c) Lógica - As instruções de geração são produzidas a partir de uma análise sintática do texto de entrada, que reconhece as diferentes estruturas gramaticais da linguagem do GPC. O analisador sintático tem como subrotina um analisador léxico ('scanner') que reconhece os símbolos básicos da gramática, fornecendo a cada chamada um novo símbolo retirado do texto de entrada e codificado no seguinte formato:

POSIÇÕES (BYTES)	CAMPO	MNEMONICO	PICTURE COBOL
1 a 6	Sequencial do registro no Texto de Entrada	SEQIN	x(6)
7 a 8	Código do símbolo	TPTOKEN	99
9 a 11	Comprimento do 'string'	L	999
12 a (11+) 'L'	'String'	S	X x 'L'

O Analisador Léxico

A análise léxica é feita através do método de

análise léxica direta (AHO |[21]|).

Os símbolos reconhecidos pelo analisador léxico são:

i) símbolos utilizados na especificação dos comandos ao GPC, extraídos dos registros assinalados com o carater especial % na 7a. posição:

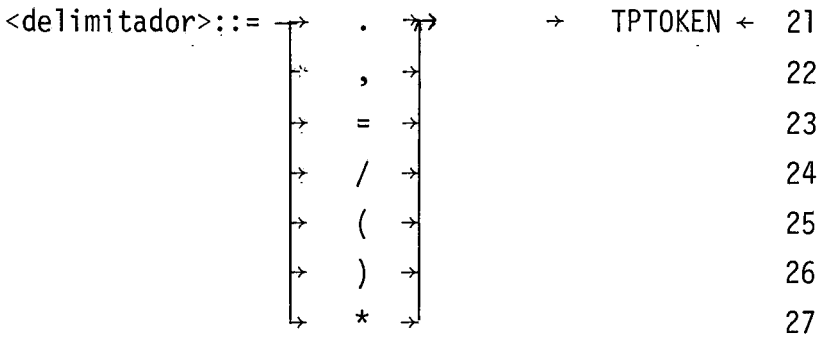
<identificador>::= → <letra> → - → <letra> → → TPTOKEN ← 28
<dígito>
L ← comprimento do identificador
S ← identificador

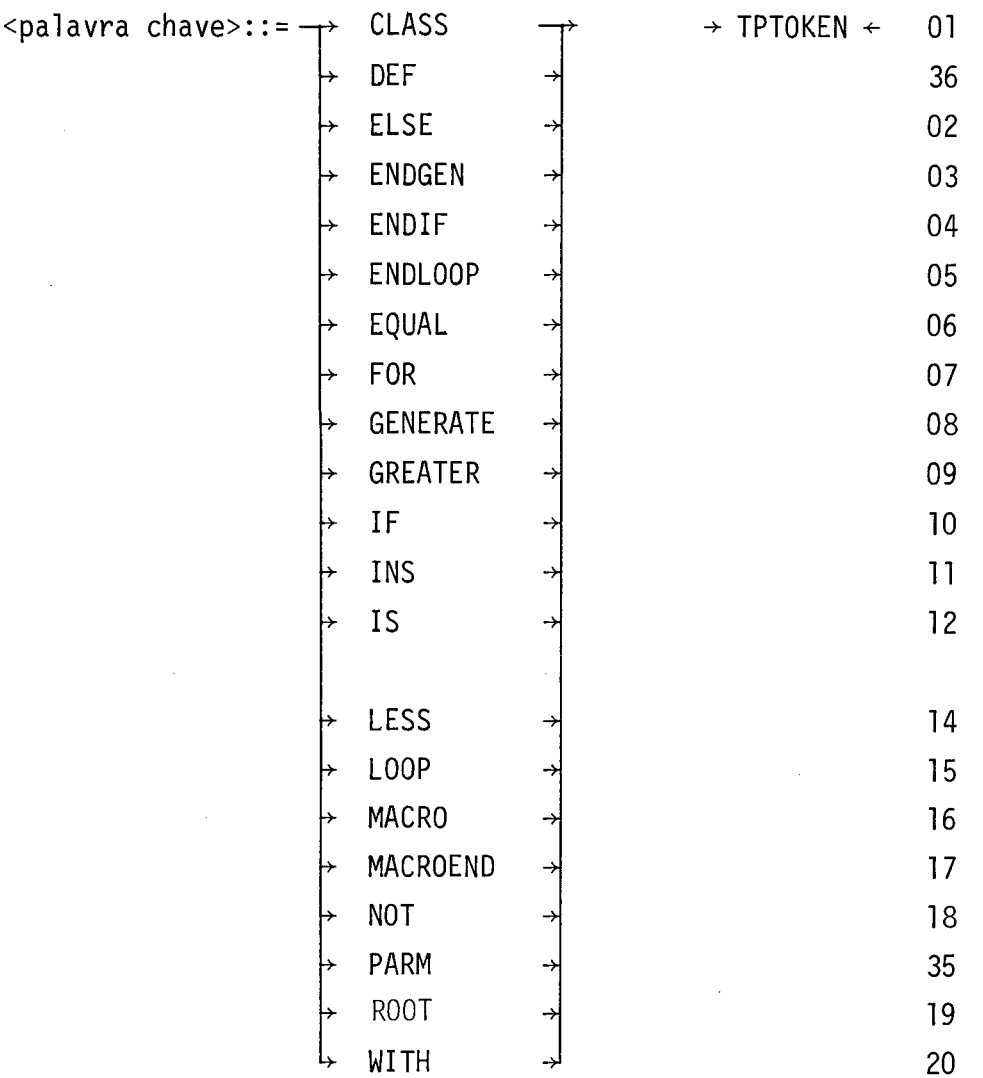
<número inteiro>::= → <dígito> → → TPTOKEN ← 29
L ← nº de dígitos
S ← nº inteiro

<literal não numérico>::= → " → <caracter> → " → TPTOKEN ← 30
EOL
L ← comprimento to literal
S ← literal(incluindo as aspas inicial e final)

<texto>::= → ## → <caracter> → ## → → TPTOKEN ← 31
EOL
L ← comprimento do texto
S ← texto (não inclui os delimitadores)

<parâmetro formal>::= → & → <identificador> → TPTOKEN ← 32
L ← comprimento do identificador
S ← identificador





Observação: as palavras chaves são, inicialmente, reconhecidas como identificadores.

ii) Símbolos utilizados na especificação do texto matriz, extraído dos demais registros.

<texto COBOL>::= → <caracter> → (& → &) → TPTOKEN ← 33

L ← comprimento do texto

S ← texto COBOL, excluindo-se 1 de cada duplo &

<parâmetro formal>::= → & → <identificador> → ≠≠ → TPTOKEN ← 32

L ← comprimento do identificador

S ← identificador

Fim de Registro → TPTOKEN ← 37

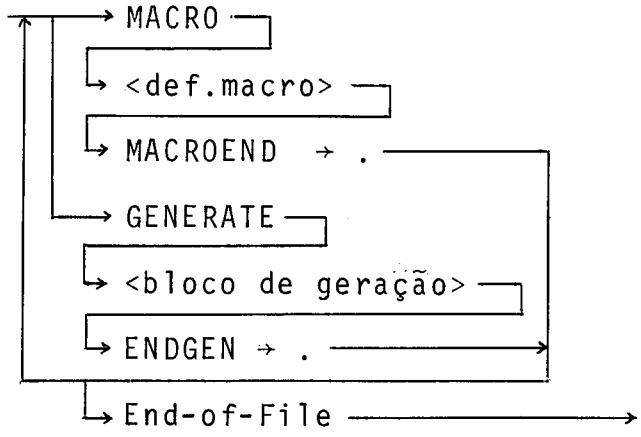
iii) Símbolo designativo de fim de texto → TPTOKEN ← 34

O Analisador Sintático

A análise sintática é feita através do método de matriz de transição (GRIES [22]). As instruções de geração produzidas pelo analisador sintático encontram-se descritas na seção 5.2.2 (Descrição do Arquivo CODE). As regras para transformação do texto de entrada nessas instruções são descritas a seguir:

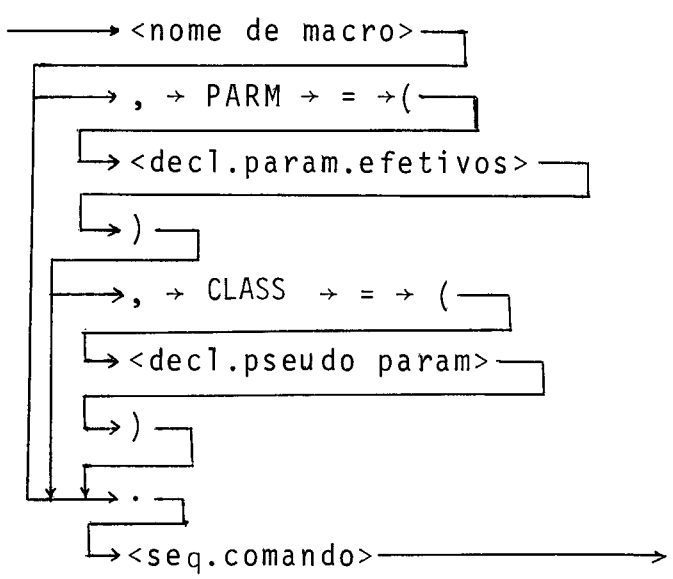
Nº	REGRA SINTÁTICA	INSTRUÇÕES GERADAS

01 <texto de entrada>::=



Regra 02
OP = 34
Regra 17
OP = 21
OP = 35

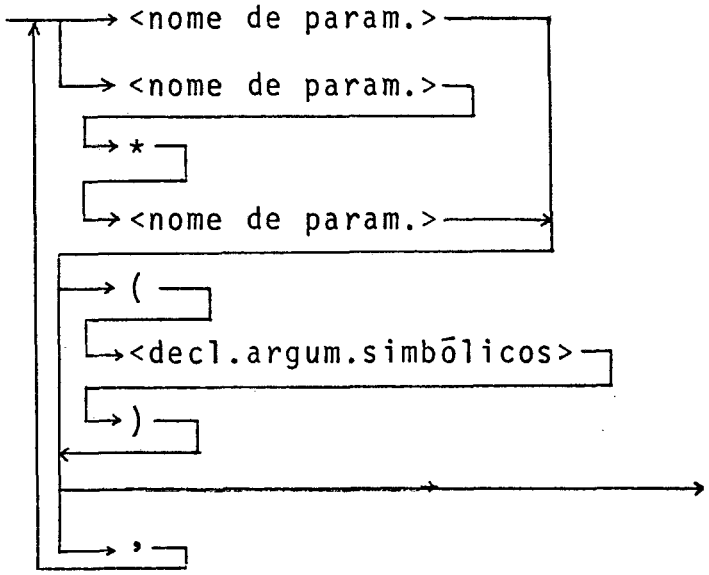
02 <def. marco>::=



OP = 22
Regra 03
OP = 27
Regra 09
OP = 33
Regra 10

Nº	REGRA SINTÁTICA	INSTRUÇÕES GERADAS

03 <decl.param.efetivo>::=



OP = 23 (I = 1)

OP = 23 (I = 2)

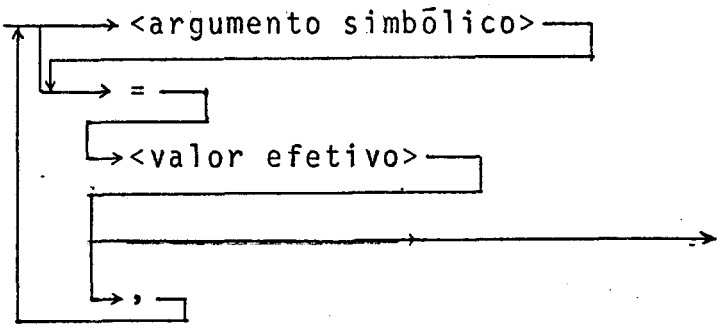
OP = 23 (I = 3)

Regra 04

OP = 26

---

04 <decl.argum.simbólicos>::=

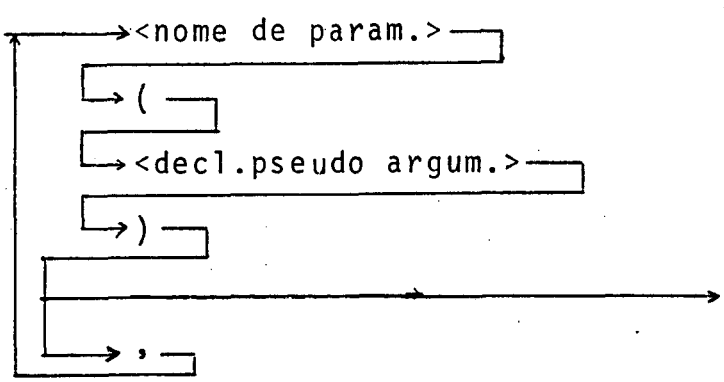


OP = 24

OP = 25

---

05 <decl.pseudo param.>::=



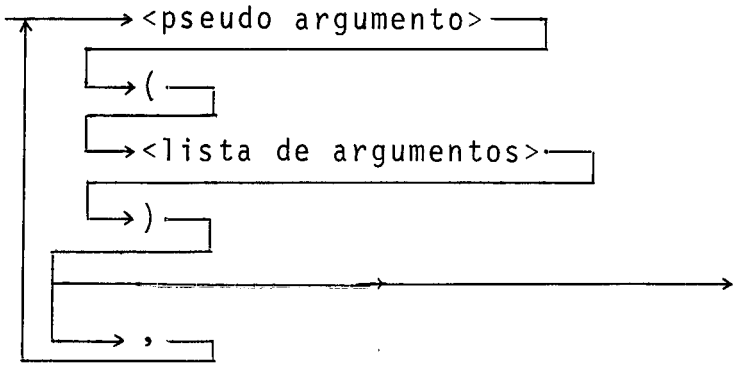
OP = 28

Regra 06

OP = 32

Nº	REGRA SINTÁTICA	INSTRUÇÕES GERADAS

06 <decl.pseudo argum.>::=



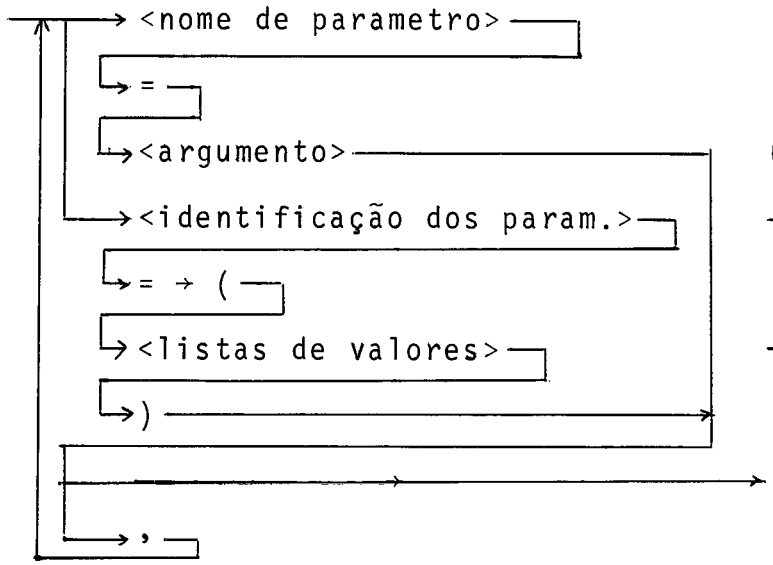
OP = 29

Regra 07

OP = 31

07 <lista de argumentos>::=

(1) (2) (3) (4)



OP=11

OP=12

OP=30 OP=18 OP=02

Regra 08

Regra 09

(1) - na declaração de pseudo argumentos

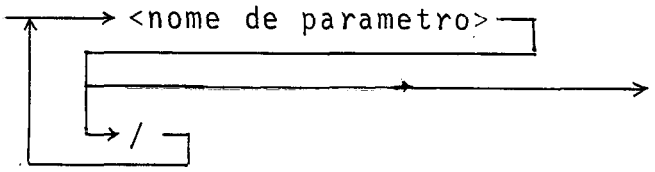
(2) - na chamada de macro

(3) - no bloco LOOP

(4) - no bloco de geração

Nº	REGRA SINTÁTICA	INSTRUÇÕES GERADAS

08 <identificação dos param.>

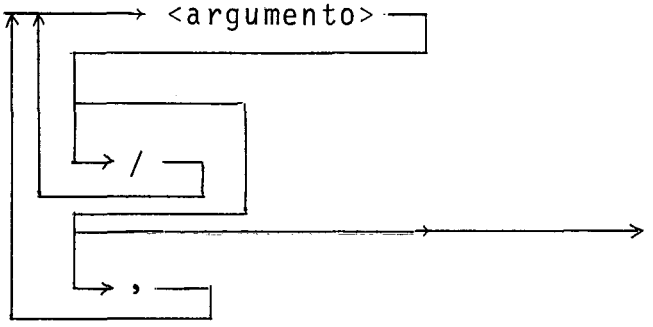


(1)	(2)	(3)	(4)
	OP=11		

(1), (2), (3), (4) - vide regra 07

---

09 <listas de valores>::=

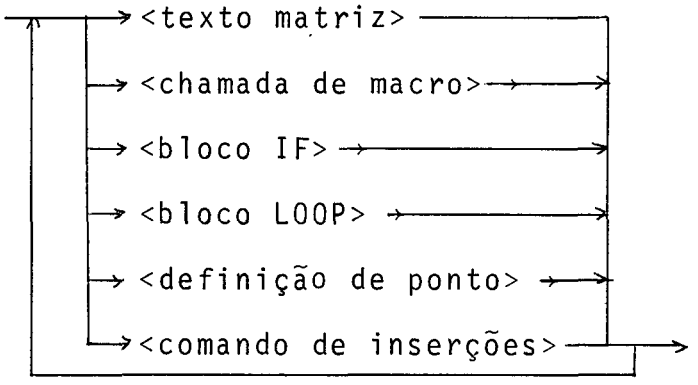


(1)	(2)	(3)	(4)
OP=30	OP=12	OP=08	OP=02

(1), (2), (3), (4) - vide regra 07

---

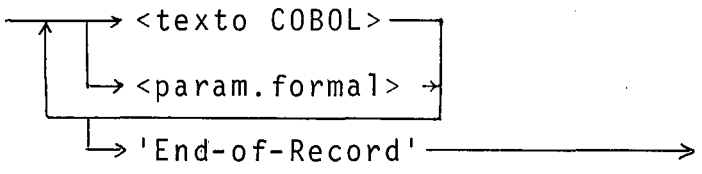
10 <seq.comandos>::=



Regra 11
Regra 12
Regra 13
Regra 14
Regra 15
Regra 16

---

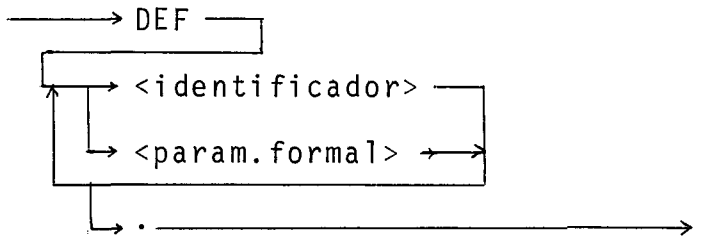
11 <texto matriz>::=



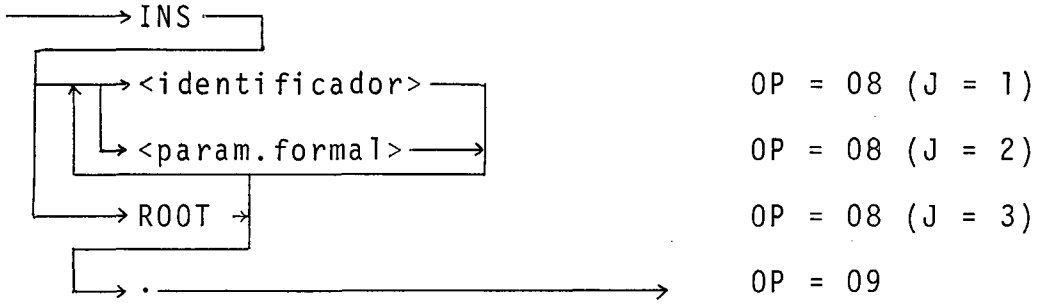
OP = 04 (J = 1)
OP = 04 (J = 2)
OP = 05

Nº	REGRA SINTÁTICA	INSTRUÇÕES GERADAS
12	<chamada de macro>::=	
	→ <nome de macro> →	OP = 10
	→ WITH →	
	→ <lista de argumentos> →	Regra 07
	→ . →	OP = 13
13	<Bloco IF>::=	
	→ IF →	
	→ NOT →	
	→ <param.formal> →	
	→ IS →	
	→ LESS →	
	→ EQUAL →	
	→ GREATER →	
	→ <argumento> →	
	→ . →	OP = 14
	→ <seq.comandos> →	Regra 10
	→ ELSE → . →	OP = 15
	→ <seq. comandos> →	Regra 10
	→ ENDIF → . →	OP = 16
14	<bloco LOOP>::=	
	→ LOOP → FOR →	OP = 17
	→ <lista de argumentos> →	Regra 07
	→ . →	OP = 19
	→ <seq. comandos> →	Regra 10
	→ ENDLOOP → . →	OP = 20

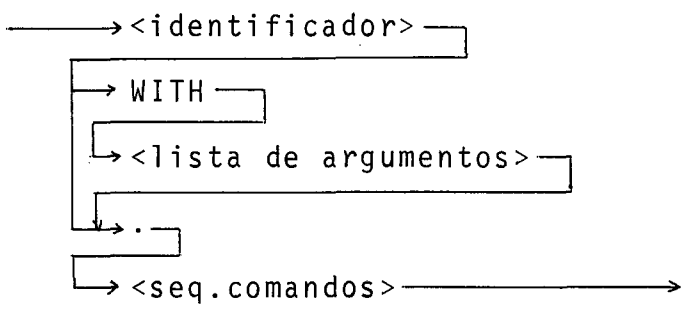
Nº	REGRA SINTÁTICA	INSTRUÇÕES GERADAS
15	<definição de ponto>	



16	<comando de inserção>::=	



17	<bloco de geração>::=	



5.3.2. Programa GPC-2

a) Objetivo - Gerar os vários segmentos do texto que irão compor o programa COBOL, e uma árvore representativa do programa.

b) Arquivos Utilizados

Entrada - Arquivo Intermediário (CODE), e
Biblioteca de Macros (MLIB).

Saídas - Texto Gerado (TEXT); e
Mensagens de Erro (ERRO).

c) Lógica - A Figura 5.3.2(a) apresenta o fluxo geral do programa, que opera de forma semelhante a um p.d.a. ('push-down-automata') com 2 estados apenas. No estado 1 as instruções de geração obtidas pela rotina FETCH são interpretadas produzindo os vários segmentos de texto COBOL e as diferentes subárvores que compõem a árvore representativa do programa. A transição para o estado 2 se faz quando é detectado, pela rotina OPERA, o início de uma sequência de instruções de geração pertencentes ao ramo falso de um Bloco IF. A rotina FALSO-IF provoca o retorno ao estado 1 ao encontrar o término da sequência de instruções a ignorar. Ao final do processo de geração é gravado no arquivo TEXT a tabela que especifica a ordenação final dos vários segmentos de texto produzidos, obtida a partir da árvore do programa.

A Rotina FETCH

Esta rotina torna disponível, a cada chamada, uma nova instrução de geração, endereçada por:

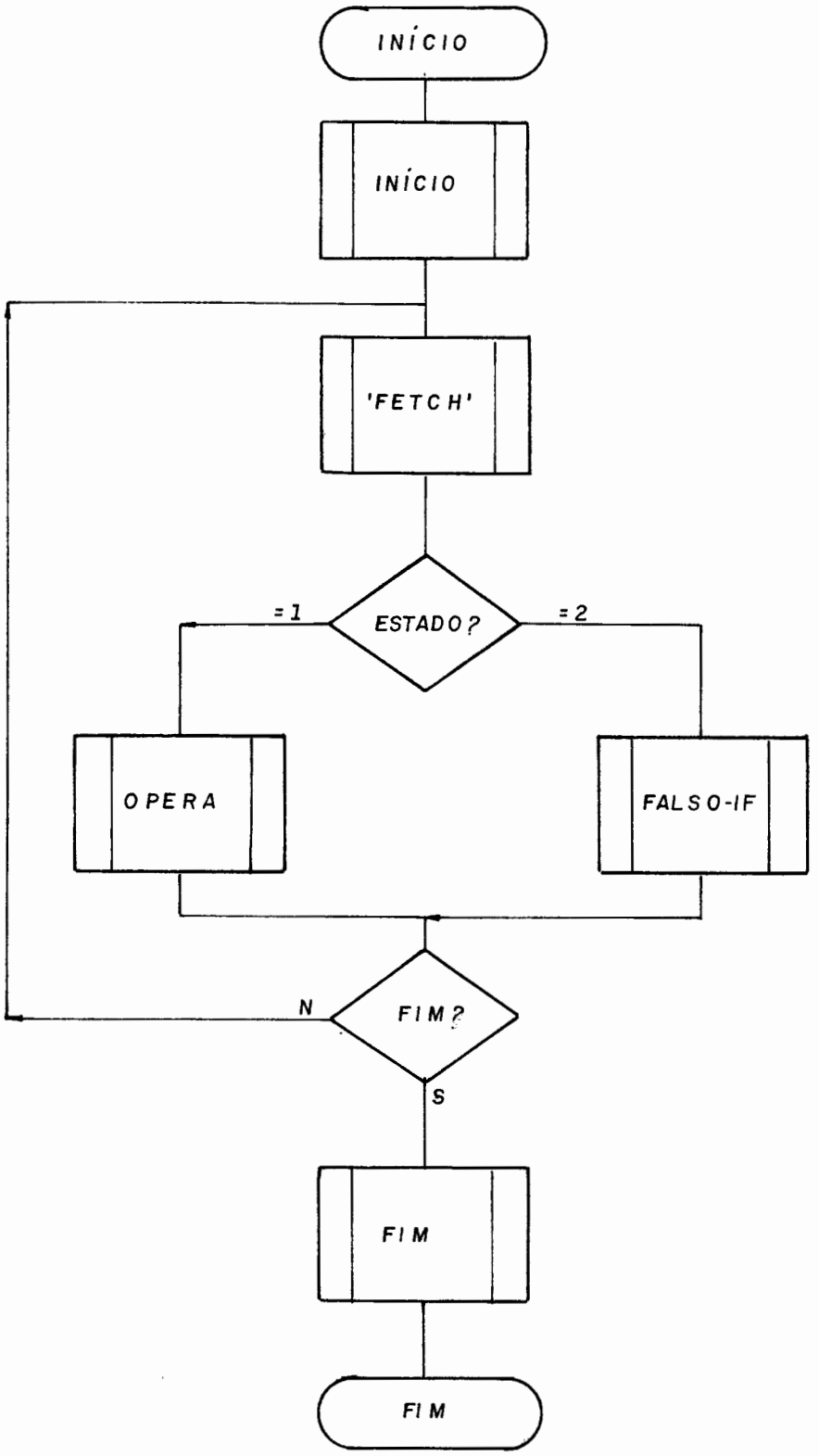


FIG. 5.3.2 (a) FLUXO GERAL DO PROGRAMA GPC-2

- indicador do arquivo onde se encontra a instrução (FILE), podendo ser 1(arquivo CODE) ou 2(arquivo MLIB);
- chave do registro correspondente no arquivo (KEY), e
- posição relativa do 1º byte da instrução no registro (IX).

A rotina mantém atualizado um campo com o endereço da próxima instrução (LOCO), que é inicializado com o endereço da 1a. instrução do 1º Bloco de Geração, do arquivo CODE.

A Rotina OPERA

Com base no código de operação (OP) da instrução de geração fornecida pela rotina 'FETCH' é selecionado um dos vários conjuntos de procedimentos a seguir, conforme apresentado na Tabela 5.3.2(b).

A estrutura de dados básica utilizada, esquematizada na Figura 5.3.2(c), compõe-se de:

i) uma pilha de controle em cujo topo são mantidas as informações básicas sobre o processo em curso, contendo:
- tipo do processo (TP), podendo ser: 1(Bloco de Geração), 2 (Macro Expansão), ou 3(Bloco LOOP),
- ponteiro para a base da pilha de parâmetros do processo (BPAR);
- ponteiro para o topo da pilha de parâmetros do processo (TPAR);
- endereço da instrução de retorno (LCRTN),

se Bloco LOOP:

- nº de ciclos a executar (NLOOP);
- endereço da instrução inicial do Bloco (LCLOOP);
- ponteiro para 1º parâmetro interno do LOOP (PARLOOP).

ii) uma pilha de parâmetros que especifica os valores atribuídos aos parâmetros dos vários processos ativos no momento, contendo:

- Tipo de Valor (TP), podendo ser:

  identificador → 1,
  nº inteiro → 2,
  literal → 3,
  texto → 4, ou
  lista de valores → 7;

  se identificador:
  - comprimento do identificador (L),
  - ponteiro para identificador na Tabela de Identificadores (P).

  se nº inteiro, literal ou texto:
  - comprimento do 'string' (L),
  - ponteiro para início do 'string' no Poço de Literais (P),
  - ponteiro para fim do 'string' no Poço do Literais (Q);

  se lista de valores:
  - nº de elementos na lista (L),
  - ponteiro para início da lista na lista de valo-

res (P),

- ponteiro para final da lista na Lista de Valores (Q);
- ponteiro auxiliar (NEXT) utilizado para indicar o próximo valor, na lista de valores a ser utilizada como parâmetro interno do Bloco LOOP.

iii) uma Lista de Valores onde são especificados os valores constituintes das listas especificadas na pilha de parâmetros (TP = 7), com formatação idêntica à Pilha de Parâmetros, utilizando-se o ponteiro NEXT para a ligação dos valores da mesma lista;

iv) uma Tabela de Identificadores onde são armazenados os identificadores referenciados pela Lista de Parâmetros; e

v) um Poço de Literais onde são armazenados os 'strings' correspondentes aos números inteiros, literais e textos referenciados pela lista de Parâmetros.

OP	SUBROTINA	AÇÃO
01	Início de Bloco de Geração	. Inicializa Pilha de Controle
02	Parâmetro de Bloco de Geração	. Inclui valor na Pilha de Parâmetros . Atualiza TPAR na Pilha de Contrôle
03	-	
04	Comando COBOL	. Concatena texto(ou valor de parâmetro)ao Comando COBOL
05	Término do Comando COBOL	. Grava comando COBOL . Inclui comando COBOL na árvore do programa
06	Definição de Ponto de Ponto de Inserção	. Concatena identificador(ou valor de parâmetro)à Identificação do Ponto de Inserção

07	Término de Def.de Ponto	.Inclui nó de definição de ponto na árvore do programa
08	Comando de Inserção	.idem OP=06
09	Término de Com.Inserção	.Posiciona ponteiro para inserção na sub-árvore do programa
10	Nome de Macro	.Guarda Nome da Macro .Inicializa lista de argumento da Macro chamada
11	Nome de Argumento	.Inclui nó na lista de argumentos da Macro chamada
12	Valor de Argumento	.Preenche nó da lista de argumentos correspondente
13	Término da chamada	.Busca Definição da Macro .Acrescenta à pilha de controle entrada para macroexpansão .Faz LOCO=primeira instrução da Definição da Macro
14	Condição Lógica	.Compara operandos .Faz ESTADO igual a 1, ou 2, depedendo do resultado da comparação e do operador lógico
15	Fim de Bloco THEN	.Faz ESTADO ← 2
16	-	
17	Início de LOOP	.Acrescenta à pilha de controle entrada para Bloco LOOP
18	Parâmetro de LOOP	idem OP = 02
19	Término dos parâmetros do LOOP	.Faz LCLOOP ← LOCO na pilha de controle .Faz NLOOP ← nº de elementos nas listas dos parâmetros do LOOP .Faz parâmetros do LOOP na pilha de parâmetros iguais aos primeiros valores das listas
20	Fim de Bloco LOOP	.Faz NLOOP ← NLOOP - 1 .Se NLOOP > 0: -faz parâmetros do LOOP na pilha de parâmetros iguais aos próximos valores das listas -faz LOCO ← LCLOOP na pilha de controle .Se NLOOP=0. -desempilha entrada na pilha de controle

21	Fim de Bloco de Geração	.Faz LOCO ← 1a.restrição do próximo Bloco de Geração (9's se final)
22	Início de Definição de	.Inicializa listas de parâmetros e pseudo parâmetros da macro
23	Parâmetro efetivo	.Inclui parâmetro na lista de parâmetros da macro .Inicializa lista de argumentos simbólicos do parâmetro
24	Símbolo argumento	.Inclui argumento simbólico na lista de argumentos simbólicos do parâmetro
25	Argumento efetivo	.Preenche nó da lista de argumento simbólico correspondente
26	-	
27	-	
28	Pseudo Parâmetro	.Inclui pseudo parâmetro na lista de pseudo parâmetros da macro .Inicializa lista de pseudo argumentos do pseudo parâmetro
29	Pseudo Argumento	.Inclui pseudo argumento na lista de pseudo argumento do pseudo parâmetro .Inicializa lista de parametros equivalente do pseudo argumento
30	Parâmetro Equivalente	.Inclui parâmetro equivalente na lista de parâmetros equivalente do pseudo argumento
31	-	
32	-	
33	Término dos pseudo argumentos	.Transforma lista de argumentos da macro,com base nas listas de parâmetros da macro; .Preenche Pilha de Parâmetros
34	Fim de Macro	.Desempilha entrada da Pilha de Controle
35	-	

Tabela 5.3.2(b) - Procedimentos da Rotina OPERA

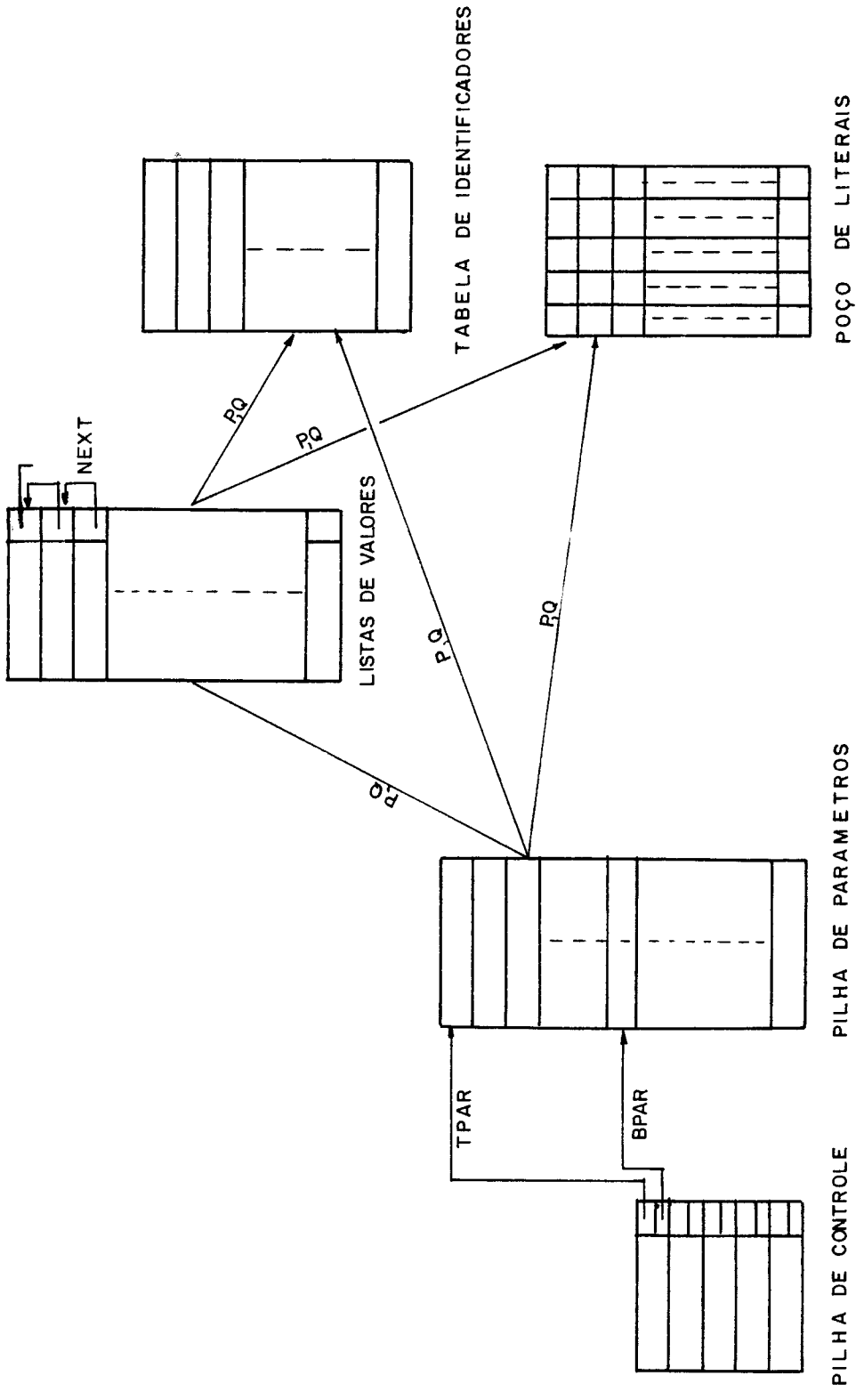


FIG.- 5.3.2 (c) ESTRUTURA DE DADOS BÁSICA

Construção da Árvore do Programa

Os algoritmos utilizados para a construção da árvore do programa e gravação da tabela resultante no arquivo TEXT estão descritos a seguir, juntamente com a estrutura de dados utilizada, representada na Fig.5.3.2(d).

ITEM	DESCRIÇÃO	MNEMONICO	PICTURE COBOL
1	Tabela de Pontos de Inserção,com:		
1.1	-identificação do Ponto de inserção	IDPONTO	x(60)
1.2	-Ponteiro para 1º nó inserido no Ponto	PRIMNO	9(4)
1.3	-Ponteiro para último nó inserido no Ponto	ULTINO	9(4)
2	Tabela de Nós, com:		
2.1	-Tipo do Nó,podendo ser: 1 → nó de definição de ponto,ou 2 → nó de segmento de texto	TP	9
2.2	-Ponteiro para próximo nó inserido no Ponto se Nó de definição de ponto	PROXNO	9(4)
2.3	-Ponteiro para entrada correspondente na Tabela de Pontos se Nó de segmento de texto:	P	9(3)
2.4	-Ponteiro para 1º comando COBOL de segmento de texto, no arquivo TEXT	PRIMT	9(8)
2.5.	-Ponteiro para último comando COBOL do segmento de texto,no arquivo TEXT	ULTT	9(8)
3	Ponteiros		
3.1	-Ponteiro para última entrada ocupada na Tabela de Pontos	TOPP	9(3)

3.2	-Ponteiro para último nó ocupado na Tabela de Nós	TOPNO	9(4)
3.3	-Ponteiro para entrada da Tabela de Pontos,correspodente ao ponto de inserção ativo	PAT	9(3)
3.4	-Ponteiro para nó de segmento de texto ativo	NOAT	9(4)
3.5	-Ponteiro para último comando COBOL gravado	T	9(8)
4	Áreas auxiliares		
4.1	-Identificação de Ponto	IDAUX	x(60)
4.2	-Ponteiro para tabela de Pontos	PAUX	9(3)
4.3	-Ponteiro para tabela de Nós	NDAUX	9(6)

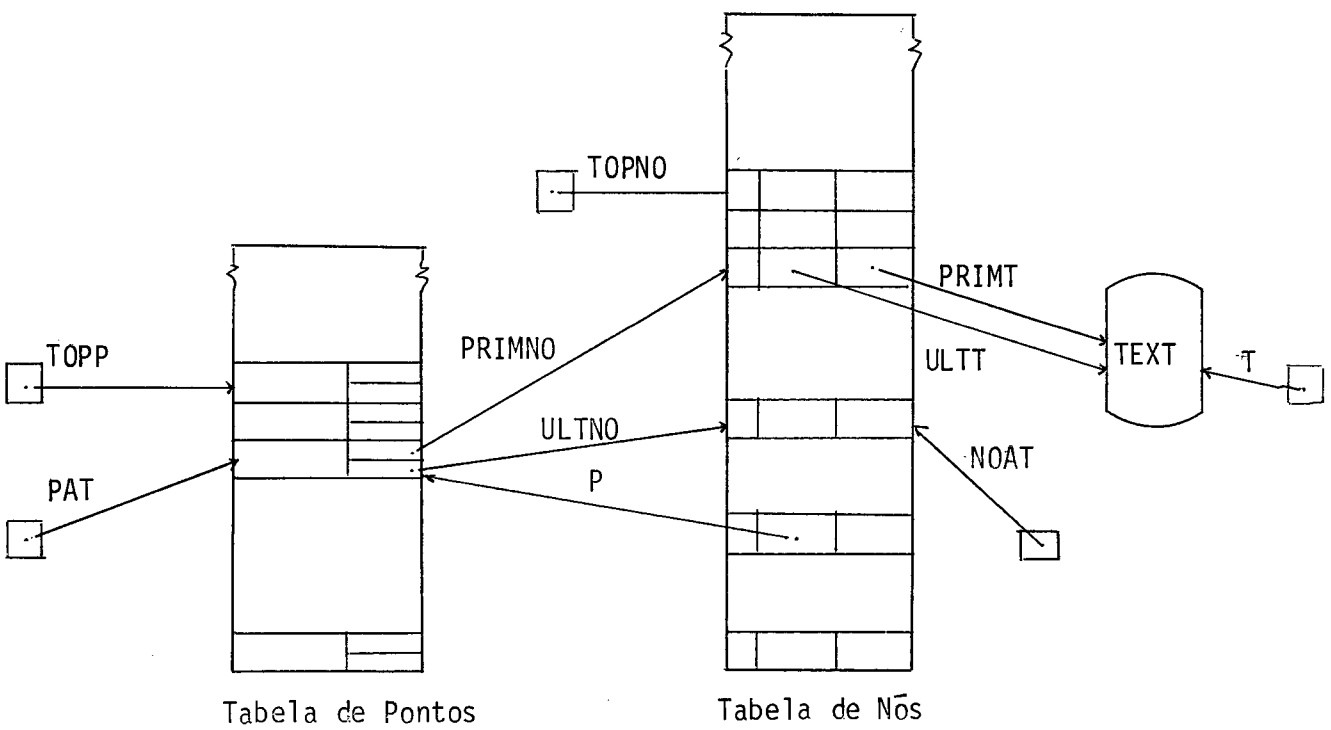


Fig.5.3.2.(d) - Estrutura de Dados para Construção da Árvore do Programa.

Algoritmos Utilizados

I) Inicialização (Rotina INICIO)

```
IDPONTO (1) ← 'ROOT' ;
PRIMNO (1)  ← ULTNO (1) ← Ø ;
TOPP    ← PAT  ← 1 ;
TOPNO   ← NOAT ← Ø ;
```

II) Processamento de Comando de Inserção (Rotina OPERA,OP=09)

(a) Busca Inserção, na Tabela de Pontos, de ponto identificado em IDAUX:

```
IDPONT (TOPP + 1) ← IDAUX;
For PAUX::=1 step 1 until IDPONTO(PAUX)=IDAUX DO { };
if PAUX > TOP
then do {TOPP ← PAUX;
         PRIMNO(PAUX) ← ULTNO(PAUX) ← Ø} ;
endif;
```

(b) Posiciona ponteiro PAT e NOAT:

```
PAT  ← PAUX;
NOAT ← Ø;
```

III) Processamento do Comando de Definição de Ponto (Rotina OPERA, OP=07)

(a) Busca Inserção, na Tabela de Pontos, do ponto identificado em IDAUX (idem II-a);

(b) Gera nó de definição de ponto na Tabela de Nós:

```
TOPNO ← TOPNO + 1;
TP(TOPNO)      ← 1;
PROXNO(TOPNO)  ← Ø;
P(TOPNO)       ← PAUX;
```

(c) Insere nó gerado no ponto de inserção ativo:

```
NOAUX ← ULTNO (PAT);
if NOAUX = Ø
then do {PRIMNO(PAT) ← ULTNO(PAT) ← TOPNO};
else do {PROXNO(NOAUX) ← ULTNO(PAT) ← TOPNO};
endif;
```

(d) Posiciona ponteiro NOAT:

```
NOAT ← Ø ;
```

IV) Processamento de comando COBOL (Rotina OPERA, OP=05)

se NOAT = Ø :

(a) Gera nó de segmento de texto na Tabela de Nós:

```
TOPNO ← TOPNO + 1;
TP(TOPNO) ← 2;
PROXNO(TOPNO) ← Ø;
PRIMT(TOPNO) ← ULTT(TOPNO) ← T
```

(b) Insere nó gerado no ponto de inserção ativo (idem III-C);

(c) Posiciona ponteiro NOAT:

```
NOAT ← TOPNO;
```

se NOAT ≠ Ø.

(d) Inclui Comando COBOL no nó de segmento de texto

```
ULTT(NOAT) ← T ;
```

V) Gravação dos nós de segmento de texto da árvore no arquivo TEXT (Rotina FIM)

(a) Inicialização da pelha auxiliar S, com o 1º nó inserido em 'ROOT'

```
S(1) ← PRIMNO(1);
TOPS ← 1;
```

(b) Visita aos nós da árvore:

```
while TOPS > 0
do{  NOAUX ← S(TOPS),
     TOPS ← TOPS - 1;
     while NOAUX ≠ Ø
     do{ if TP(NOAUX) = 1
         then { TOPS ← TOPS + 1;
                D (TOPS) ← PROXNO(NOAUX);
                PAUX ← P(NOAUX);
                NOAUX ← PRIMNO(PAUX);}
        else  { grava nó de segmento de texto apontado por NOAUX;
                NOAUX ← PROXNO(NOAUX)}
        endif. }
     end do;}
```

5.3.3. Programa GPC-3

a) Objetivo - concatenar os vários segmentos de texto gerados na ordem especificada, produzindo o texto fonte COBOL.

b) Arquivos utilizados

Entrada - Texto Gerado (TEXT)

Saída - Texto Fonte COBOL (PROG)

c) Lógica - A tabela contida nos registros especificados no Rótulo do Arquivo é percorrida na ordem em que se apresenta, recuparando-se os segmentos de texto especificados e reproduzindo-se na saída os vários comandos COBOL correspondentes. Cada comando COBOL é reproduzido em um ou mais registros, dependendo do seu comprimento, aplicando-se as regras para continuação de registros definidos pelo COBOL. Os registros gravados recebem uma nova numeração, a partir de 000100 com incremento de 100 a cada um, e reproduzem no campo destinado à identificação (posições 73 a 80) o sequencial do registro do texto de entrada que o originou.

5.3.4. Programa GPC-4

a) Objetivo - efetuar as modificações especificadas na Biblioteca de Macros

b) Arquivos utilizados

Entrada - Especificações para atualização (SPEC)
Arquivo Intermediário (CODE)
Biblioteca de Macros Anterior (MLIB)

Saída - Biblioteca de Macros (MLIB)
Relatório de Ocorrências (RELOC)

c) Lógica - Os cartões de especificação são lidos e interpretados, produzindo-se uma tabela de alterações a efetuar, com os nomes das macros associados a um código indicativo do tipo de alteração (Inclusão, Substituição ou Exclusão).

Em seguida as entradas desta tabela são intercaladas com as entradas do Índice da Biblioteca de Macros Anterior, dando origem a um índice atualizado, já com as exclusões especificadas. Este índice contém, associado ao nome da macro, um indicador do arquivo de onde será retirada a definição atualizada, podendo ser: 1 para especificar o arquivo CODE no caso das macros a incluir ou substituir, ou 2 para especificar o arquivo MLIB no caso das macros inalteradas.

Finalmente o índice atualizado é percorrido, reproduzindo-se na nova Biblioteca de Macros as definições indicadas.

CAPÍTULO VI

CONCLUSÕES

6.1. Resumo e Discussão

A utilização de programas auxiliares no desenvolvimento de programas de aplicação tem grande importância para a otimização dos resultados que se pode obter através dos métodos de programação convencionais.

Os processadores de macros de uso geral constituem uma importante classe de programas auxiliares, que conferem grande flexibilidade ao processo de programação, e cujos benefícios dependem em grande parte da habilidade do programador na sua utilização.

Os programas auxiliares de uso restrito à programação COBOL dividem-se quanto à sua única ou principal função em: abreviadores, tradutores de tabelas de decisão, geradores de programas tipo e extensores da linguagem COBOL.

Tanto os abreviadores como os extensores da linguagem COBOL possuem características semelhantes às de um processador de macros, sendo que, no caso dos abreviadores as facilidades oferecidas são reduzidas a um mínimo, e, no caso dos extensores da linguagem, as definições de macros são intrínsicas ao programa auxiliar, e correspondem às estruturas de controle do tipo IF-THEN-ELSE, blocos BEGIN-END, e outras.

Os tradutores de tabelas de decisão e geradores de programas-tipo servem a propósitos específicos, não se apli

ando ao desenvolvimento de qualquer programa indistintamente. Os geradores de programas tipo oferecem a maior rapidez na elaboração de um programa, da classe de programas a que se aplica, limitando, porém, a otimização dos programas produzidos às características do programa base intrínsico ao gerador.

O programa auxiliar apresentado, o GPC, visa compartilhar as vantagens de um gerador de programas - tipo - rapidez de programação - com as vantagens oferecidas pelo processadores de macro - flexibilidade - e que permitem um aperfeiçoamento constante da qualidade dos programas produzidos.

O GPC apresenta-se como um processador de macros orientado para a programação COBOL, cujas características essenciais são:

a) a geração de um programa foi dividida em duas etapas: a geração do texto e a sua composição na ordem adequada. Com isto as dificuldades decorrentes da rígida estrutura exigida pela linguagem COBOL são facilmente superadas;

b) o texto de entrada é tratado integralmente de uma mesma forma extendendo-se as facilidades de geração de texto e composição a todo o texto escrito pelo programador, e não só às definições de macros;

c) a linguagem utilizada foi definida visando as necessidades específicas da geração de programas, ficando assim reduzida a um pequeno número de componentes de fácil compreensão e utilização;

d) as facilidades oferecidas para a transformação das listas de argumentos nas chamadas de macro, tais como argumentos

simbólicos e pseudo-argumentos, facilitando a definição de conjuntos de macros equivalentes, para diferentes 'dialetos' da linguagem COBOL (ou diferentes equipamentos), mas que possuam chamadas uniformizadas.

A implementação do GPC foi projetada visando a maior independencia possível de equipamentos, utilizando-se unicamente a linguagem COBOL e demais recursos mínimos indispensáveis. Estas instruções acarretam uma certa perda na eficiência do GPC, sob o ponto de vista de execução, que deverá ser avaliado em cada caso específico de implementação e, quando recomendável, reduzida através da recodificação dos trechos menos eficientes utilizando-se os recursos específicos disponíveis.

6.2. Recomendações

Em prosseguimento ao trabalho ora apresentado, recomenda-se que após um período de observação dos resultados obtidos com a utilização do GPC por diferentes equipes de programação, se procedam aos seguintes estudos:

a) avaliação dos benefícios reais obtidos com o uso do GPC, comparando-se os resultados obtidos pelas equipes usuárias com resultados obtidos por equipes não-usuárias do GPC. Podem ser planejados experimentos baseados em programas de diferentes níveis de complexidade, comparando-se estatísticamente os tempos de programação, e outras medidas de performance das várias equipes (nº de erros, tempos de execução, dentre outros);

b) extensão das facilidades oferecidas para a geração de um conjunto de programas relacionados entre si, a partir de uma especificação conjunta das várias funções desempenhadas pelos programas;

c) análise da necessidade e conveniência, e posterior inclusão, de outras facilidades de geração de texto tais como:

- definição de variáveis internas às macros e instruções de atribuição de valor e operação sobre essas variáveis e os parâmetros formais de macro;
- definição de parâmetros globais a várias macros, a fim de permitir a comunição entre macros de um mesmo nível.

BIBLIOGRAFIA

[1] BERNSTEIN, M.I. - Hardware is Easy: It's Software that's Hard. Datamation. Chicago 24(12):32-6, nov.15-1978.

[2] McCRACKEN, Daniel D. - The Changing Face of Applications Programming. Datamation. Chicago 24(12):25-30, nov.15-1978.

[3] YOHE, J.M. - An Overview of Programming Practices. Computing Surveys. New York 6(4):221-45, dec.1974.

[4] HOARE, C.A.R. - Hints on Programming Language Design. Comp. Sci. Dep. Rep. nº CS-403. Stanford, 1973.

[5] REIFER, Donald J. & TRATTNER, Stephen. A Glossary of Software Tools and Techniques. Computer. New York 10(7): 52-60, july 1977.

[6] NAFTALY, Staneley M. et alii.- COBOL Support Packages. New York, John Willey & Sons, 1972.182 p.

[7] BROWN, P.J. - A Survey of Macro Processors. Annual Review in Automatic Programming. Oxford 5(3):37-87, 1969.

[8] COLE, A.J. - Macro Processors. Cambridge, Cambridge University Press, 1976. 230 p.

[9] POOCH, Udo W. - Translation of Decision Tables. Computing Surveys . New York, 6(2):125-51, june 1974.

[10] BABENKO, L.P. & SINYAGOSKAYA, V.V. MAKROBOL - A Generator of COBOL Programs. Cybernetics. New York 12(2):196-200, mar./abr.1976.

|11| WEINBERG, Gerald M. et alii- High Level COBOL Programming. Cambridge, Winthrop Publishes Inc., 1977. 252 p.

|12| TAUSWORTHE, Robert C. - Standardized Development of Computer Software. New Jersey, Prentice-Hall Inc., 1977. 379 p.

|13| PARNAS, D.L. - A Technique for Software Module Specification with Examples. Comm.ACM . New York 15(5): 330-6, may 1972.

|14| PARNAS, D.L. - On the Criteria To Be Used in Decomposing Systems into Modules. Comm. ACM. New York 15(12):1053-8, dec.1972.

|15| PARNAS, D.L. - On the Design and Development of Program Families. IEEE Trans. Softs.Eng. , New York SE-2(1):1-9, mar.1976.

|16| HARDING, D.A. - Modular Programming - Why Not? The Australian Com.J. Chippendale 4(4):150-6, nov.1972.

|17| DIJKSTRA, E.W. - GO TO Statment Considered Harmful. Comm.ACM. New York 11(3):147-8, mar.1968.

|18| McCLURE, Carma - Structured Programming in COBOL. SIGPLAN Notices. New York 10(4):25:33, apr.1975.

|19| GERDER, AlleVan. - Structured Programming in COBOL. Comm.ACM. New York 20(1):2-12, jan.1977.

|20| WEINBERG, Gerald M. - The Psychology of Improved Programming Performance. Datamation. Chicago 17(11): 82-5, nov.1972.

[21] AHO, A.V. & ULLMAN, J.D. - The Theory of Parsing, Translation and Compiling, vol.I.

[22] GRIES, David - Compiler Construction for Digital Computer . New Yor, John Wiley & Sons Inc., 1971. 493 p.