EXPAND II: UM TRADUTOR PARA UMA LINGUAGEM EXTENSÍVEL ATRAVÉS DE MACROS SINTÁTICAS

Olinto José Varela Furtado

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.) EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. José Lucas M. Rangel Netto
Presidente

Prof. Michael Stanton

Prof. Sérgio de M. Schneider

Prof.ª Sueli Mendes dos Santos

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 1984

FURTADO, OLINTO JOSÉ VARELA

EXPAND II: Um Tradutor para uma Linguagem Extensível Através de Macros Sintáticas (Rio de Janeiro) 1984.

VII , 138 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1984).

Tese - Universidade Federal do Rio de Janeiro, COPPE.

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

A minha esposa MAFALDA.

Aos meus pais HERMES e ZENITA.

AGRADECIMENTOS

Ao Prof. José Lucas Mourão Rangel Netto, pelas idéias, conhecimentos, orientação e amizade.

A João Batista Furtuoso (NPD/UFSC), pelo apoio, incentivo e amizade constantes.

A Miguel Argollo Jr., pelas sugestões e amizade.

Ao Prof. Clávio Coutinho Filho, (UFSC) pelo seu empenho para que eu pudesse realizar esse curso.

Aos amigos da COPPE e da UFSC, pelo apoio que deles recebi.

A meus familiares, que colaboraram e estimularam na execução deste trabalho.

A D. Ilda pelo acolhimento.

A Denise, pela datilografia.

A UFSC, a CAPES e ao CNPq pelo recursos fornecidos.

Resumo da Tese Apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências. (M.Sc.).

EXPAND II: UM TRADUTOR PARA UMA LINGUAGEM EXTENSÍVEL ATRAVÉS DE MACROS SINTÁTICAS

Olinto José Varela Furtado

SETEMBRO DE 1984

Orientador: José Lucas Mourão Rangel Netto
Programa: Engenharia de Sistemas e Computação

Este trabalho descreve a implementação de um tradutor para uma linguagem extensível (EXPAND II), que permite a aceitação de novas construções definidas através de macros sintáticas (as quais deverão satisfazer algumas condições verificáveis na fase de compilação); sendo o resultado da tradução, um programa na linguagem ALGOL.

A extensão do analisador (semelhante a um analisador SLR(1)) dá-se através da construção de novos estados, criados em função das macros introduzidas, os quais são acrescidos aqueles do analisador construído para a gramática original; esta extensão é dinâmica, efetuada em tempo de compilação.

Como resultado da extensão, o analisador aceitará a linguagem base ou estendida, evitando o reprocessamento de construções repetidas na expansão das macros.

Constam deste trabalho: especificação do esquema de extensão utilizado, especificação sintática da linguagem EXPAND II, análise léxica, tratamento de macros, análise sintática, recuperação de erros, esquema de tradução e a implementação de um diretório/dicionário de macros.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

EXPAND II: TRANSLATOR FOR EXTENSIBLE LANGUAGE USING SYNTAX MACROS

Olinto José Varela Furtado

SEPTEMBER 1984

Chairman: José Lucas Mourão Rangel Netto
Department: Engenharia de Sistemas e Computação

We describe here the implementation of a translator which accepts as source an extensible language, with facilities to accept new constructions defined by means of syntax macros (subject to some conditions verifiable during compilation); the output code is a program in ALGOL.

The extension of the parser (akin to a SLR(1) parser) is made by the construction of new states, created as needed during the examination of the declared macros; those states are effectively added to those of the original parser. This extension is realized during compile-time.

The resulting extended parser accepts the base language, or the extended language, avoiding the repeated parsing of constructions repeated during macroexpansion.

Included here are: the specification of the extension scheme used; the syntatic specification of the EXPAND II language; algorithms for scanning, macro treatment, parsing, error recovery, translation scheme and the implementation of a small macro directory/dictionary.

ÍNDICE

Página

Página

CAPÍTULO I

INTRODUÇÃO

A utilização de macros deu-se inicialmente em substituição a procedures e subrotinas cujos corpos correspondentes não justificavam o tempo e o espaço requeridos para a link-edição e para os mecanismos de passagem de parâmetros, e caracterizava-se principalmente pela substituição da chamada pelo texto correspondente a sua definição, com a devida associação dos parâmetros, evitando a repetição do texto de sua definição em cada uso; estas macros tem sido usadas (principalmente) em conjunção com linguagens de montagem (Assembly Languages) e também em linguagens de alto nível (como é o caso do DEFINE do ALGOL do B6700). O tratamento destas macros, normalmente é efetuado antes ou durante a análise léxica, com pouco ou nenhum envolvimento com a sintaxe da linguagem base.

|CHEATHAM 8| e |LEAVENWORTH 9|, separadamente, introduziram o conceito de macro sintática visando a extensão de linguagens de alto nível. Uma macro sintática é uma macro pertencente a alguma categoria sintática da linguagem base (isto é, derivável na linguagem base), sendo que seus parâmetros também deverão pertencer a alguma categoria sintática. As macros sintáticas são utilizadas para acrescentar novas construções sintáticas à linguagem base, sendo por isso mais compatíveis com a idéia de extensão de linguagens, do que as macros convencionais.

A vantagem das macros sintáticas sobre outros tipos de macros dependerá sempre da aplicação em questão, e reside no fato de que os programas que utilizam estas macros serão sempre programas sintaticamente corretos, desde que as regras de definição e utilização especificadas sejam observadas. Além disso, outra vantagem considerável é a possibilidade do uso de macros com formato livre; permitindo assim uma maior liberdade criativa e resultando na satisfação das necessidades do usuário, da maneira que, dentro de certos limites, melhor lhe convier. A necessidade de realizar o processo de macro expan-

são durante a análise sintática (uma vez que em fases posteriores a linguagem estendida não será reconhecida e nas fases anteriores as informações sobre as categorias sintáticas não estarão disponíveis) desencadeará em uma nova vantagem: a possibilidade de se efetuar a análise sintática dos parâmetros reais (atuais), sem que o texto correspondente seja examinado mais do que uma vez.

A idéia de fazer a expansão das macros sintáticas durante uma análise sintática simplificada (paralela a análise léxica e em um passo anterior a análise sintática propriamente dita) foi empregada na implementação de um compilador extensível para a linguagem EXPAND no microcomputador MITRA 15 do Laboratório de Automação e Simulação de Sistemas do Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ; esta implementação está descrita em |KULLOCK 13| e |ZAKIMI 14|.

O presente trabalho trata de uma implementação a nível experimental de uma variação do esquema de extensão proposto em |RANGEL 1|; tendo como objetivo testar a implementabilidade do referido esquema e tornar disponível um compilador extensível nesses moldes, com a finalidade de verificar a aceitabilidade do esquema proposto visando uma posterior e definitiva implementação. A saída do compilador (tradutor) será um programa na linguagem Algol; optou-se por esta tradução em função da característica experimental desta implementação e da semelhança entre a linguagem base considerada e a linguagem ALGOL.

O esquema de extensão considerado, preve a expansão dinâmica da linguagem base com novas construções sintáticas, introduzidas através do uso de macros sintáticas; neste esquema a análise sintática de referências a macros, dar-se-á da mesma forma que a análise sintática das construções inerentes a linguagem base, sendo necessário para isto que as tabelas do analisador/macroexpansor sejam aumentadas sempre que uma nova macro seja declarada, de forma que as macros poderão ser utilizadas (no escopo de suas declarações) como se realmente fossem construções da linguagem base; enquanto que a expansão das macros ocorrerá após a análise da estrutura usada para referenciá-las, isto é, após o reconhecimento dos parâmetros reais e da referida estrutura. Uma característica importante do proces

so de macroexpansão, é o fato de que na análise sintática da definição os parâmetros não serão reanalisados, ao contrário, serão simplesmente associados aos parâmetros reais correspondentes utilizados na referência à macro em questão; logo, as várias ocorrências de um mesmo parâmetro na definição, serão associadas a um único parâmetro real, de maneira que para todas as ocorrências, o código gerado será o mesmo.

A linguagem base utilizada é a linguagem EXPAND II, a qual foi definida em função do esquema proposto e da implementação efetuada, com o objetivo de suportar o mecanismo de extensão, ser simples e possuir alguns recursos de programação comparáveis aos de algumas linguagens mais difundidas.

A organização deste trabalho é a seguinte: inicialmente é apresentado o esquema de extensão proposto, através de definições básicas, exemplos e algoritmos; em seguida é especificada a linguagem EXPAND II, especificação esta que além da sintaxe das construções originais, apresenta a sintaxe a ser considerada na declaração e na utilização de macros sintáticas. Após isto, nos capítulos IV, V e VI, encontra-se descrita a implementação propriamente dita, sendo que: o capítulo IV é destinado ao tratamento efetuado sobre macros, abrangendo os procedimentos referentes a: análise de declarações de macros, extensão das tabelas do analisador, utilização e expansão, tratamento de erros e uso do diretório de macros; o capítulo V destina-se a descrição do esquema de tradução utilizado, o qual consiste na transformação do programa EXPAND II em uma árvore sintática, a partir da qual será gerado um programa na linguagem ALGOL, programa este que será a saída do processo de compilação; no capítulo VI estão descritos os procedimentos referentes à análise léxica, análise sintática e recuperação de erros, efetuados durante o reconhecimento do fonte EXPAND II. O trabalho se completa com a apresentação de algumas conclusões e considerações finais.

CAPÍTULO II

FUNDAMENTOS TEÓRICOS

Neste capítulo apresentaremos os aspectos teóricos em que se fundamenta este trabalho, os quais compreendem definições e algoritmos necessários para o tratamento de macros sintáticas de uma forma prática: através do aumento da tabela SLR (o que ocorre no fim de cada declaração de macro encontrada) de maneira tal que o analisador sintático associado passe a aceitar a linguagem estendida resultante da introdução de macros sintáticas.

As definições fundamentais e os algoritmos aqui expostos podem ser encontrados em |RANGEL 1|, cujo artigo originou o presente trabalho e do qual este capítulo é um apanhado geral; todavia, algumas diferenças podem ser encontradas entre os algoritmos aqui apresentados e os propostos em |RANGEL 1| de maneira que os primeiros são variações dos últimos; variações essas que foram introduzidas em função de restrições impostas à formação da estrutura de macros na presente implementação. Além das definições e dos algoritmos, em |RANGEL 1| pode ser encontrado também a prova de correção dos algoritmos utilizados; note que apesar de estarmos trabalhando com variações dos algoritmos propostos em |RANGEL 1| (os quais são mais gerais), as propriedades fundamentais são mantidas e portanto a prova de correção é a mesma.

Nesta exposição estamos supondo que o leitor está familiarizado com a teoria de linguagens livres de contexto e principalmente com gramáticas e linguagens SLR e seus analisadores (cujos conceitos, definições e algoritmos básicos podem ser encontrados em |AHO & ULLMANN 2|).

A notação aqui utilizada segue basicamente |AHO & ULLMANN 3| onde o leitor também poderá encontrar definições e propriedades fundamentais.

II.1. DEFINIÇÕES FUNDAMENTAIS, ANÁLISE E EXPANSÃO DE MACROS

Dada uma gramática livre de contexto $G = (N,\Sigma,P,S)$, denominada gramática base, podemos estender a linguagem base $L(G)$ através de macros sintáticas, usando o seguinte esquema de extensão:

DEFINIÇÃO - Um esquema de extensão de uma gramática livre de contexto, é uma construção $E = (G,\Delta,R)$ onde:

$G = (N,\Sigma,P,S)$ é a gramática base.

Δ é um conjunto de símbolos auxiliares, também considerados terminais.

R é um conjunto de macros que são construções da forma $A \rightarrow \alpha , \beta$ onde:

- A é um não-terminal da gramática base o qual especifica a categoria sintática da macro.
- α é uma sequência em $(N \cup \Sigma \cup \Delta)^+$, onde o primeiro símbolo é o nome da macro e deve pertencer a Δ, denominada estrutura da macro a qual não deverá conter símbolos não-terminais consecutivos.
- β é uma sequência em $(N \cup \Sigma \cup \Delta)^*$ denominada definição da macro.

As produções de G, a gramática base, devem estar numeradas de 1 a p e a produção $0: S' \rightarrow S\$$ deve ser acrescentada a P (onde S' é um símbolo não-terminal novo e \$ é a marca de final de sentença). As macros de R, são produções que quando acrescidas a gramática base dão origem a gramática estendida G_E, e devem ser numeradas consecutivamente a partir de p + 1.

A correspondência semântica entre α e β é feita através da numeração (posição) dos não-terminais em α. Esta numeração deve ser feita de maneira que a cada não-terminal de β corresponda um e somente um não-terminal de α; não havendo restrições quanto ao número de símbolos de β que correspondam a um mesmo símbolo de α.

DEFINIÇÃO - Dado um esquema de extensão $E = (G,\Delta,R)$, definimos a gramática estendida de E como sendo:

$$G_E = (N,\Sigma,\Delta,P \cup R',S)$$

onde $R' = \{A \rightarrow \alpha \mid A \rightarrow \alpha, \beta \in R\}$

$L(G_E)$ será a linguagem estendida correspondente.

A análise de uma sequência da linguagem estendida é feita através de sua expansão, a qual consiste em substituir cada ocorrência de α, a estrutura de uma macro $A \rightarrow \alpha,\beta$, por β a definição correspondente. Por ser um algoritmo essencialmente ascendente (bottom-up), o algoritmo que vamos apresentar faz essa expansão de dentro para fora, isto é, expande inicialmente as referências mais internas às macros.

ALGORITMO DE ANÁLISE E EXPANSÃO DE MACROS

Este algoritmo é uma variante do algoritmo de análise sintática dos analisadores SLR, e pode ser aplicado se as duas condições abaixo forem satisfeitas:

1) A gramática estendida G_E deve ser SLR(1).
2) Para cada macro i: $A \rightarrow \alpha,\beta$ a definição β deve ser derivável em G_E a partir de A, sem que seja feito uso das produções i, i + 1,... de G_E.

As variações introduzidas em um analisador SLR(1) convencional |AHO & ULLMANN 2| para que G_E possa ser analisada, são as seguintes:

1) A entrada do analisador poderá conter símbolos não-terminais. Assim uma configuração do algoritmo será da forma $(\alpha,\beta\$)$, onde α e β são sequências de terminais e não-terminais; α representando o conteúdo da pilha sintática e β representando a entrada a ser analisada (na realidade, a sequência α não é empilhada mas define os estados a serem empilhados).

2) As ações GOTO serão tratadas como ações de empilhamento (SHIFT), isto é, os não-terminais serão empilhados da mesma forma que os terminais.

3) Na redução por uma produção i: A → α (i > p) proveniente da macro i: A → α,β, em vez de substituir α por A no topo da pilha, retiraremos α da pilha e acrescentaremos β à esquerda da entrada a ser analisada:

$$(\gamma\alpha\ ,\ \delta\$) \vdash (\gamma\ ,\ \beta\delta\$)$$

OBS.: A saída do analisador não foi especificada, entretanto fica claro que o processo de macro-expansão é simulado de maneira que os segmentos repetidos são examinados apenas uma vez. Essa é a razão da inclusão de não-terminais na entrada.

EXEMPLO: Considere a seguinte gramática:

```
0: S → PROG $                    9: E → E + T
1: PROG → LISTA .                10:    | T
2: LISTA → LISTA; COM            11: T → T * F
3:       | COM                   12:    | F
4: COM   → id : COM              13: F → (E)
5:       | id := E                14:    | id
6:       | begin LISTA end
7:       | if E then COM
8:       | goto id
```

a qual acrescentaremos as macros

COM → *while* E *do* COM, id: if E then begin COM; goto id end
F → *quad* F , (F * F)
F → *cubo* F , (quad F * F)

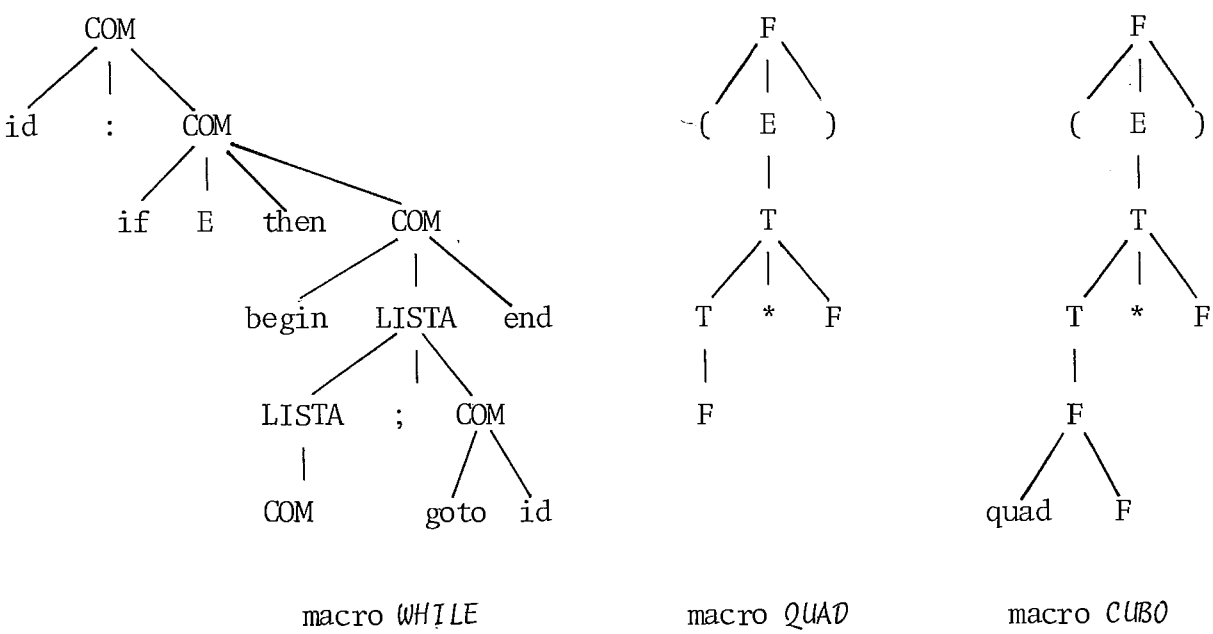
Figura II.1 - Gramática Base e Macros Consideradas

Neste exemplo, a gramática base G, constitui-se das produções ∅ a 14 enquanto que a gramática estendida G_E é definida pela união da gramática base com as produções correspondentes às macros introduzidas, as quais deverão ser numeradas consecutivamente às produções de G. A gramática estendida resultante é mostrada a seguir.

```
0: S  → PROG $                    9: E → E + T
1: PROG  → LISTA.                10:    | T
2: LISTA → LISTA; COM             11: T → T * F
3:       | COM                   12:    | F
4: COM   → id: COM                13: F → (E)
5:       | id:= E                 14:    | id
6:       | begin LISTA end        15: COM → while E do COM
7:       | if E then COM          16: F   → quad F
8:       | goto id                17:     | cubo F
```

Figura II.2 - Gramática Estendida

A primeira condição imposta pelo algoritmo só poderá ser testada durante a extensão do analisador (e será mostrada na seção seguinte); já a segunda condição do algoritmo em questão, pode ser mostrada através das árvores de derivação da definição das macros introduzidas. Para a definição das macros utilizadas em nosso exemplo, teríamos as seguintes árvores de derivação:



macro *WHILE* macro *QUAD* macro *CUBO*

Figura II.3 - Derivação da Definição das Macros na Gramática Estendida.

OBS.: Note que a árvore de derivação da macro CUBO envolve a produção 16, a qual corresponde a uma macro definida anteriormente.

Continuando nosso exemplo, mostraremos agora a utilização das macros apresentadas e o efeito do analisador sintático quando da análise das mesmas; para isso utilizaremos a seguinte sequência da linguagem estendida:

```
begin
   while id1
   do id1 := id1 + cubo id2
end .
```

a qual produz, em têrmos da linguagem base, a sequência abaixo:

```
begin
   id: if id1
       then begin
               id1 := id1 + ((id2 * id2) * id2);
               goto id
            end
end .
```

A seguir são mostradas as árvores de derivação do segmento utilizado.

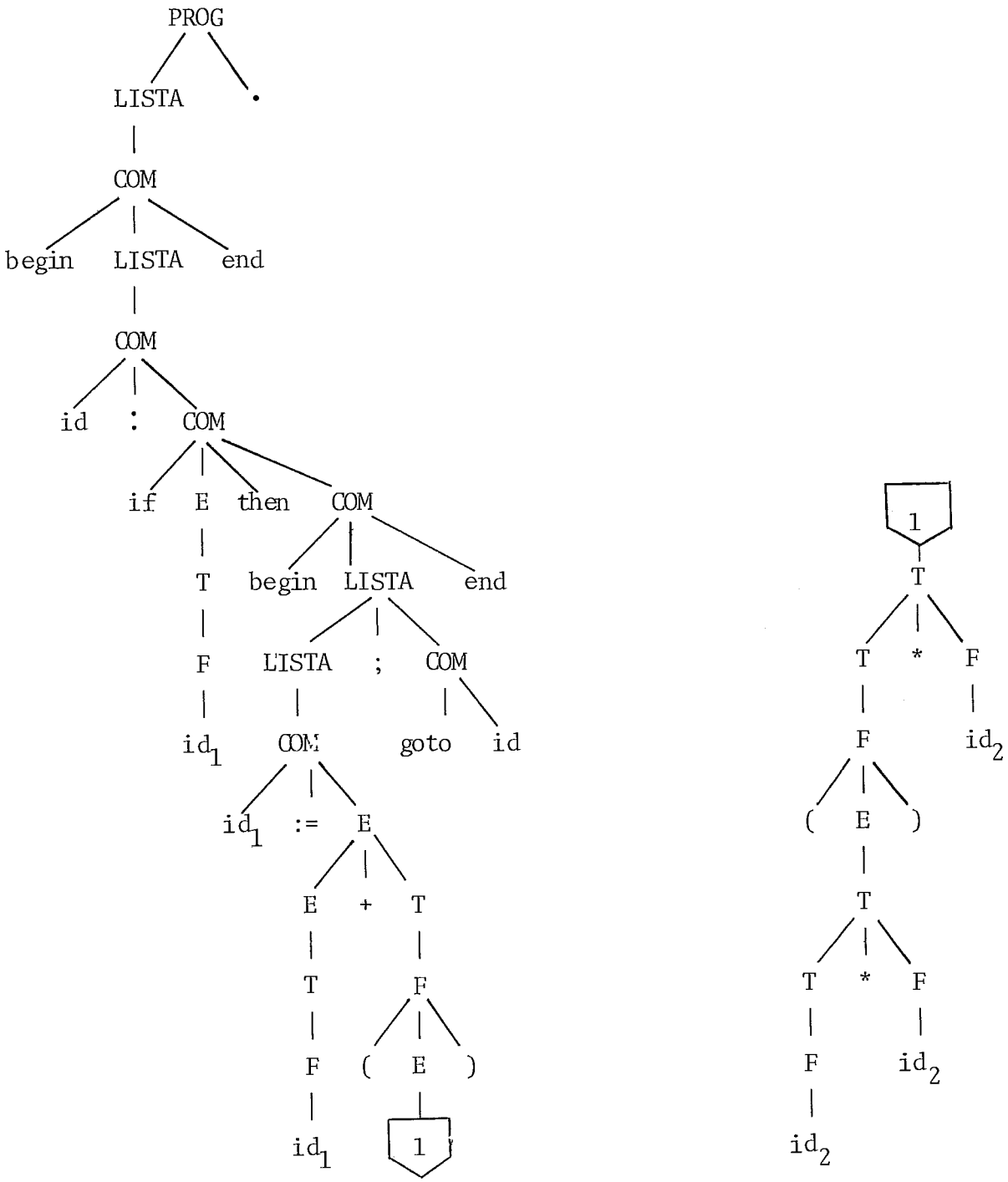


Figura II.4 - Derivação na Gramática Base

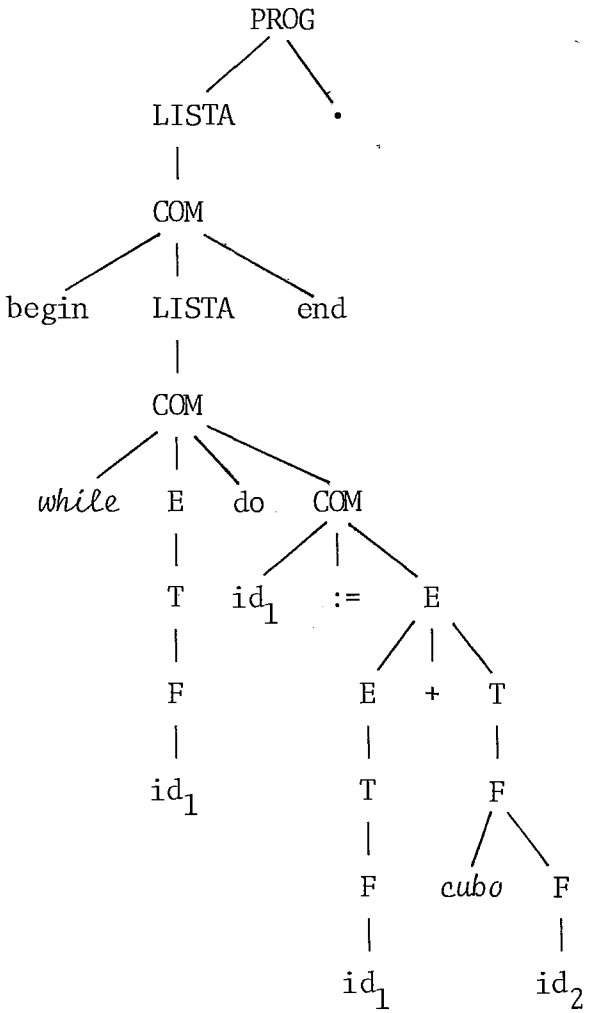


Figura II.5 - Derivação na Gramática Estendida

Na verdade, o algoritmo apresentado não constrói tais árvores; da mesma forma que os analisadores ascendentes usuais de um passo único, ele utiliza apenas uma pilha para a análise sintática e macro-expansão e é dirigido pela tabela de análise sintática do analisador.

Na prática, além da pilha sintática, é usada uma pilha semântica (paralela a pilha sintática) para possibilitar a correspondência entre os parâmetros (representados pelos não-terminais) da estrutura e da definição.

Vejamos como analisar e expandir o segmento da linguagem estendida usado anteriormente; para isso utilizaremos a tabela sintática da figura II.8 construída para a gramática estendida em questão.

Nº	CONFIGURAÇÃO	OBS
1	$\emptyset$ begin while id_1 do id_1 := id_1 + cubo id_2 end•$	1
2	$\emptyset_{\text{begin } 5}$ while id_1 do id_1 := id_1 + cubo id_2 end•$	2
3	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ id_1 do id_1 := id_1 + cubo id_2 end•$	
4	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $id_{1\ 17}$ do id_1 := id_1 + cubo id_2 end•$	
⋮	⋮ ⋮	
10	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ do id_1 := id_1 + cubo id_2 end•$	
11	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ id_1 := id_1 + cubo id_2 end•$	
12	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$:= id_1 + cubo id_2 end•$	
13	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ id_1 + cubo id_2 end•$	
14	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $id_{1\ 17}$ + cubo id_2 end•$	
⋮	⋮ ⋮	
21	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ cubo id_2 end•$	
22	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{\text{cubo } 46}$ id_2 end•$	
23	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{\text{cubo } 46}$ $_{id_{2\ 17}}$ end•$	
24	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{\text{cubo } 46}$ $\underline{F}$ end•$	
25	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{\text{cubo } 46}$ $_{F\ 47}$ end•$	3
26	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ (quad F * F) end•$	
27	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{(\ 16}$ quad F * F) end•$	
28	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{(\ 16\text{quad } 44}$ F * F) end•$	
29	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{(\ 16\text{quad } 44}$ $_{F\ 45}$ * F) end•$	4
30	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ $id_{1\ 4}$ $_{:=\ 11}$ $_{E\ 21+\ 24}$ $_{(\ 16}$ (F * F) * F) end•$	
31	$\emptyset_{\text{begin } 5}$ $_{\text{while } 40}$ $_{E\ 41}$ $_{\text{do } 42}$ id $_{4\ :=\ 11}$ $_{E\ 21+\ 24}$ $_{(\ 16}$ $_{(\ 16}$ F * F) * F) end•$	
⋮	⋮ ⋮	

Nº	CONFIGURAÇÃO		OBS
41	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24_{(}\ 16_{(}\ 16_{E}\ 26_{)}\ 30$	* F) end•$	
42	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24_{(}\ 16$	$\underline{F}$ * F) end•$	
43	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24_{(}\ 16_{F}\ 15$	* F) end•$	
⋮	⋮	⋮	
52	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24_{(}\ 16_{E}\ 26_{)}\ 30$	end•$	
53	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24$	$\underline{F}$ end•$	
54	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24_{F}\ 15$	end•$	
55	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24$	$\underline{T}$ end•$	
56	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21_{+}\ 24_{T}\ 28$	end•$	
57	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11$	$\underline{E}$ end•$	
58	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{id_1}\ 4_{:=}\ 11_{E}\ 21$	end•$	
59	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42$	$\underline{COM}$ end•$	
60	$\emptyset_{begin}\ 5_{while}\ 40_{E}\ 41_{do}\ 42_{COM}\ 43$	end•$	5
61	$\emptyset_{begin}\ 5$	id : if E then begin COM; goto id end end•$	
62	$\emptyset_{begin}\ 5_{id}\ 4$	: if E then begin COM; goto id end end•$	
63	$\emptyset_{begin}\ 5_{id}\ 4_{:}\ 10$	if E then begin COM; goto id end end•$	
64	$\emptyset_{begin}\ 5_{id}\ 4_{:}\ 10_{if}\ 6$	E then begin COM; goto id end end•$	6
65	$\emptyset_{begin}\ 5_{id}\ 4_{:}\ 10_{if}\ 6_{E}\ 13$	then begin COM; goto id end end•$	
66	$\emptyset_{begin}\ 5_{id}\ 4_{:}\ 10_{if}\ 6_{E}\ 13_{then}\ 23$	begin COM; goto id end end•$	
67	$\emptyset_{begin}\ 5_{id}\ 4_{:}\ 10_{if}\ 6_{E}\ 13_{then}\ 23_{begin}\ 5$	COM; goto id end end•$	
68	$\emptyset_{begin}\ 5_{id}\ 4_{:}\ 10_{if}\ 6_{E}\ 13_{then}\ 23_{begin}\ 5_{COM}\ 3$	; goto id end end•$	
⋮	⋮	⋮	

Nº	CONFIGURAÇÃO		OBS
78	$\emptyset_{\text{begin}}\ {}^{5}\text{id}\ {}^{4}\text{:}\ {}^{10}\text{if}\ {}^{6}\text{E}\ {}^{13}\text{then}\ {}^{23}\text{begin}\ {}^{5}\text{LISTA}\ {}^{12}\text{end}\ {}^{22}$	end•$	
79	$\emptyset_{\text{begin}}\ {}^{5}\text{id}\ {}^{4}\text{:}\ {}^{10}\text{if}\ {}^{6}\text{E}\ {}^{13}\text{then}\ {}^{23}$	COM end•$	
80	$\emptyset_{\text{begin}}\ {}^{5}\text{id}\ {}^{4}\text{:}\ {}^{10}\text{if}\ {}^{6}\text{E}\ {}^{13}\text{then}\ {}^{23}\text{COM}\ {}^{27}$	end•$	
81	$\emptyset_{\text{begin}}\ {}^{5}\text{id}\ {}^{4}\text{:}\ {}^{10}$	COM end•$	
82	$\emptyset_{\text{begin}}\ {}^{5}\text{id}\ {}^{4}\text{:}\ {}^{10}\text{COM}\ {}^{20}$	end•$	
83	$\emptyset_{\text{begin}}\ {}^{5}$	COM end•$	
84	$\emptyset_{\text{begin}}\ {}^{5}\text{COM}\ {}^{3}$	end•$	
85	$\emptyset_{\text{begin}}\ {}^{5}$	LISTA end•$	
86	$\emptyset_{\text{begin}}\ {}^{5}\text{LISTA}\ {}^{12}$	end•$	
87	$\emptyset_{\text{begin}}\ {}^{5}\text{LISTA}\ {}^{12}\text{end}\ {}^{22}$	•$	
88	$\emptyset$	COM•$	
89	$\emptyset_{\text{COM}}\ {}^{3}$	•$	
90	$\emptyset$	LISTA•$	
91	$\emptyset_{\text{LISTA}}\ {}^{2}$	•$	
92	$\emptyset_{\text{LISTA}}\ {}^{2}$ •8	$	
93	$\emptyset$	PROG $	
94	$\emptyset_{\text{PROG}}\ {}^{1}$	$	7
95	ACEITA!		

Figura II.6 - Configurações do Analisador Mostrando a Análise e a Expansão do Segmento Utilizado.

Observações anotadas ao longo da sequência de configurações:

(1) Configuração inicial do analisador.
(2) Início da análise da macro *WHILE*.
(3) Completado o reconhecimento da estrutura da macro *CUBO*, ocorre a ação de redução pela produção 17, que acrescenta a definição da macro *CUBO* na entrada do analisador, fazendo a associação adequada dos parâmetros (representados pelos não-terminais) envolvidos.
(4) Reconhecimento da estrutura da macro *QUAD*, e a correspondente ação R16.
(5) Reconhecimento da estrutura da macro *WHILE*, e a correspondente ação R15.
(6) Observe a ocorrência de um não-terminal na cadeia de entrada e por isso, tendo a ação de GOTO interpretada como SHIFT; note que a subexpressão correspondente não será reexaminada.
(7) Configuração final do analisador, aceitando o segmento da linguagem estendida analisado.

II.2. CONSTRUÇÃO DA TABELA DO ANALISADOR/MACROEXPANSOR

Nesta seção mostraremos como construir a tabela sintática do analisador SLR(1) da gramática estendida (G_E), partindo da tabela sintática do analisador SLR(1) da gramática base (G), central para o algoritmo de análise sintática mostrado na seção anterior.

Para simplificar e sem perda de generalidade, suporemos que apenas uma macro deve ser acrescentada (o que é equivalente a estender a linguagem em vários passos, acrescentando uma macro por vez).

ALGORITMO DE CONSTRUÇÃO DOS ESTADOS AUXILIARES

Os "estados auxiliares" do analisador são estados construídos em função da gramática base e não são alcançáveis a partir do estado inicial (estado $\emptyset$). A construção destes estados, dá-se da seguinte forma:

Para cada não-terminal A de G, é criado o estado q(A) definido como sendo:

$$q(A) = \text{CLOSURE}\ (\{A \rightarrow \cdot\ \alpha \mid A \rightarrow \alpha \in P\}),$$

sendo que a função CLOSURE é definida como habitualmente, |AHO & ULLMANN 2|.

Para cada um dos estados assim criados, da mesma forma que se faz a partir do estado inicial $\emptyset$ ou equivalentemente q(S), calculamos as transições com todos os símbolos terminais e não-terminais, criando os estados que se fizerem necessários. Para a gramática exemplo da seção anterior, a partir do estado inicial $\emptyset$ foram criados os estados 1 a 30, os quais são mostrados na figura II.7.

De acordo com a definição acima, o estado q(PROG) da gramática considerada será:

q(PROG) = CLOSURE ({PROG → •LISTA '.'})
= PROG → •LISTA '.', LISTA → •LISTA; COM,
LISTA → •COM, COM → •id : COM, COM → •id := E,
COM → •begin LISTA end, COM → •if E then COM,
COM → •goto id},

que embora seja um sub-conjunto do estado inicial, não é alcan

çável a partir dele. A este estado é atribuído o nº 31 e dele são alcançados os estados 2 a 7 já existentes. O processo é repetido para q(LISTA) (estado 32), a partir do qual é obtido o estado 33 (novo) e os estados 3 a 7 já existentes, q(COM) (34), q(E) (35), q(T) (37) e q(F) (39), sendo criado adicionalmente os estados 36 e 38 alcançáveis de q(E) = 35 e q(T) = 37 respectivamente. A composição dos estados da gramática base e dos estados auxiliares é mostrada a seguir.

Ø. S → • PROG $ (q(s))
PROG → • LISTA '•'
LISTA → • LISTA; COM
| • COM
COM → • id ; COM
| • id := E
| • begin LISTA end
| • if E then COM
| • goto id

1. S → PROG •$

2. PROG → LISTA • '•'
LISTA → LISTA • ; COM

3. LISTA → COM •

4. COM → id • : COM
| id • := E

5. COM → begin • LISTA end
LISTA → • LISTA; COM
| • COM
COM → • id : COM
| • id := E
| • begin LISTA end
| • if E then COM
| • goto id

6. COM → if • E then COM
E → • E + T
| • T
T → • T * F
| • F
F → • (E)
| • id

7. COM → goto • id

8. PROG → LISTA '•' •

9. LISTA → LISTA; • COM
COM → • id : COM
| • id := E
| • begin LISTA end
| • if E then COM
| • goto id

10. COM → id : • COM
| • id : COM
| • id := E
| • begin LISTA end
| • if E then COM
| • goto id

11. COM → id := •E
    E → • E + T
    | • T
    T → • T * F
    | • F
    F → • (E)
    | • id

12. COM → begin LISTA • end
    LISTA → LISTA • ; COM

13. COM → if E • then COM
    E → E • + T

14. E → T •
    T → T • * F

15. T → F •

16. F → (• E)
    E → • E + T
    | • T
    T → • T * F
    | • F
    F → • (E)
    | • id

17. F → id •

18. COM → goto id •

19. LISTA → LISTA; COM •

20. COM → id : COM •

21. COM → id := E •
    E → E • + T

22. COM → begin LISTA end •

23. COM → if E then • COM
    COM → • id : COM
    | • id := E
    | • begin LISTA end
    | • if E then COM
    | • goto id

24. E → E + • T
    T → • T * F
    | • F
    F → • (E)
    | • id

25. T → T * • F
    F → • (E)
    | • id

26. F → (E •)
    E → E • + T

27. COM → if E then COM •

28. E → E + T •
    T → T • * F

29. T → T * F •

30. F → (E) •

31. PROG → • LISTA '•' (q(PROG))
    LISTA → • LISTA; COM
    | • COM
    COM → • id : COM
    | • id := E
    | • begin LISTA end
    | • if E then COM
    | • goto id

32. LISTA → • LISTA; COM (q(LISTA))
| • COM
COM → • id : COM
| • id := E
| • begin LISTA end
| • if E then COM
| • goto id

33. LISTA → LISTA • ; COM

34. COM → • id : COM (q(COM))
| • id := E
| • begin LISTA end
| • if E then COM
| • goto id

35. E → • E + T (q(E))
| • T
T → • T * F
| • F
F → • (E)
| • id

36. E → E • + T

37. T → • T * F (q(T))
| • F
F → • (E)
| • id

38. T → T • * F

39. F → • (E) (q(F))
| • id

Figura II.7 - Estados do analisador SLR(1) de G e estados auxiliares.

OBS.: Os estados auxiliares só conterão reduções se a gramática considerada possuir produções da forma A → ε; entretanto, quando isto ocorrer o processo de construção dos estados auxiliares não será alterado, uma vez que as reduções aparecem naturalmente do follow.

A utilização dos estados auxiliares verifica-se durante o processo de extensão da tabela do analisador.

ALGORITMO DE EXTENSÃO DA TABELA PARA ACEITAÇÃO DE UMA NOVA MACRO.

ENTRADA - A tabela sintática do analisador SLR(1) da gramática base.
- Indicação do estado q(A), para todo A $\in$ N.
- Informação sobre o FOLLOW de todos os não-terminais.
- Uma macro $A \rightarrow \alpha , \beta$.

SAÍDA - A tabela sintática do analisador SLR(1) da gramática estendida, preparada para comandar a análise e a expansão (de acordo com a macro considerada) caso a gramática estendida seja SLR(1) e β, a definição da macro em questão, seja derivável a partir de A na gramática base.
- Informação sobre o FOLLOW de todos os não-terminais, atualizada para incluir o efeito da produção $A \rightarrow \alpha$.

FASE 1 - Verifique se na formação da macro as seguintes regras foram observadas:
a - A categoria sintática deve ser especificada por um símbolo não-terminal.
b - O nome da macro (primeiro símbolo de α) deve ser um identificador.
c - A estrutura da macro (α) não pode conter não-terminais (parâmetros) consecutivos.
d - A correspondência entre os não-terminais de α e β deve ser tal, que a cada não-terminal de β corresponda um único não-terminal de α.
(Se qualquer destas regras foi violada, a macro não poderá ser aceita).

FASE 2 - Verifique se $A \Rightarrow^{+} \beta$. Para isso pode ser utilizado o próprio analisador, partindo-se do estado q(A). Para permitir as reduções, um símbolo pertencente a FOLLOW(A) deve ser acrescentado a direita de β, tomando-se precaução para que o mesmo não seja empilhado.

FASE 3 - Criação dos estados novos.

Suponhamos que $\alpha = X_1 X_2 \ldots X_k$. Inicialmente devem ser criados k estados temporários os quais deverão corresponder aos itens LR(1) $i_1: A \rightarrow X_1 \cdot X_2 \ldots X_k$, $i_2: A \rightarrow X_1 X_2 \cdot X_3 \ldots X_k$, ..., $i_k: A \rightarrow X_1 X_2 \ldots X_k \cdot$, e serão denominados $S_1, S_2, \ldots, S_k$, respectivamente. Para os estados $S_1, S_2, \ldots, S_{k-1}$, a ação na linha de S_j e na coluna de X_{j+1} deve ser SHIFT S_{j+1}; enquanto que S_k corresponderá ao estado de redução da macro e será tratado na fase 5.

Nesta fase, um estado novo será uma expressão da forma $q_1 \cup q_2 \cup \ldots \cup q_m$, onde os qi's podem ser os estados temporários $S_1, S_2, \ldots, S_k$ ou estados pré-existentes; não sendo permitido a presença de estados criados nesta fase (exceção feita aos estados temporários) dentre os qi's. A cada estado novo será atribuído um número. Em seguida, crie estados $P_1, P_2, \ldots$ a medida que se tornarem necessários, da seguinte maneira:

$P_1 = S_1 \cup q(X_2)$ se X_2 for um não-terminal ou
$P_1 = S_1$ se X_2 for um terminal.

Supondo criados os estados P_1 a P_i, e preenchidas as linhas da tabela sintática para $P_1, \ldots, P_{j-1}$, a linha da tabela sintática correspondente a P_j ($j < i$) é preenchida como segue:

Se $P_j = q_1 \cup q_2 \cup \ldots \cup q_m$, então para cada $X \in (N \cup \Sigma \cup \Delta)$ siga as instruções correspondentes ao caso que se aplicar:

CASO 1 - Para $q_1 \ldots q_m$ as ações correspondentes ao símbolo X são de erro. Então a ação correspondente a X na linha de P_j também deverá ser de erro.

CASO 2 - Para $q_1 \ldots q_m$ as ações correspondentes ao símbolo X são de empilhamento ou de erro. Então a ação para X na linha de P_j deve ser de empilhamento, sendo criado para esse fim um novo estado (P_{i+1})o qual será igual a união dos estados alcançados em cada caso em que a ação é diferente de erro. Se algum S_L cons-

tar dessa lista, a ação será de empilhamento para o estado P_L (o qual ainda não foi analisado), sendo os demais estados alcançados incluidos na lista dos estados que formam P_L; observe que o estado P_{i+1} não será construído neste caso, nem nos casos em que apenas um estado for alcançado a partir de P_j.

CASO 3 - As ações correspondentes a X são de reduções pela mesma produção ou de erro. Neste caso a ação correspondente na linha de P_j será de redução por esta produção.

Nos demais casos, a gramática obtida acrescentando-se $A \rightarrow \alpha$ às produções da gramática base não é SLR(1) e a macro em questão não deve ser considerada.

No final desta fase os estados temporários, criados inicialmente, podem ser abandonados, pois não mais serão utilizados.

FASE 4 - Transições para os estados recém-criados. Para cada estado q, pré-existente ou recém-criado, tal que a ação q(A) é definida, a ação de q com o símbolo X_1 deve ser de empilhamento para o estado P_1.

FASE 5 - Redução da Macro. Atualize a função FOLLOW para incluir o efeito de $A \rightarrow \alpha\cdot$ da seguinte forma:

a) Acrescente FOLLOW(A) em FOLLOW(X_k), se X_k for um símbolo não-terminal.

b) Para i = 1, k-1, se X_i for um não-terminal, acrescente X_{i+1} em FOLLOW(X_i).

c) Propague estas alterações (atualizações).

Atualizado o FOLLOW, proceda como segue:

- Faça (P_L,X) = "Reduza $A \rightarrow \alpha , \beta$", onde P_L é o estado novo que contem $S_k = (A \rightarrow \alpha\cdot)$ e onde X é um símbolo $\in (\Sigma \cup \Delta)$ presente em FOLLOW(A).
- Atualize as reduções em toda a tabela de acordo com o FOLLOW atualizado.

Nesta última fase, se em alguma situação surgir algum conflito, a gramática estendida não é SLR(1).

Se o algoritmo acima foi executado até o fim sem que nenhum erro ou conflito tenha surgido, a gramática é SLR(1), e exceto pela possível repetição de estados, a tabela sintática obtida é a tabela do analisador SLR(1).

No que segue, através da Figura II.8, é mostrada a tabela resultante da aplicação dos algoritmos (aqui apresentados) à gramática base (G) considerando-se a introdução das macros WHILE, QUAD e CUBO propostas na seção anterior, a qual nada mais é do que a tabela do analisador da gramática estendida (G_E) considerada. Nesta tabela, os símbolos "PROG" a "F" são os não-terminais de G, os símbolos "id" a "$" são os terminais originais ($\in \Sigma$) de G e os símbolos "while", "do", "quad" e "cubo" são os terminais criados em função das macros introduzidas ($\in \Delta$); os estados 1 a 30 são os estados originais (acrescidos de algumas entradas criadas em função das macros) do analisador de G, os estados 31 a 39 são os estados auxiliares criados em função dos não-terminais de G e os estados 40 a 47 são os estados novos, correspondentes às macros WHILE (40 a 43), QUAD (44 a 45) e CUBO (46 a 47).

	PROG	LISTA	COM	E	T	F
Ø	1	2	3			
1						
2						
3						
4						
5		12	3			
6				13	14	15
7						
8						
9			19			
10			20			
11				21	14	15
12						
13						
14						
15						
16				26	14	15
17						
18						
19						
20						
21						
22						
23			27			
24					28	15
25						29
26						
27						
28						
29						
30						
31		2	3			
32		33	3			
33						
34						

	PROG	LISTA	COM	E	T	F
35				36	14	15
36						
37					38	15
38						
39						
40				41	14	15
41						
42			43			
43						
44						45
45						
46						47
47						

	id	+	*	(	)	begin	end	if	then	goto	:	:=	;	•	$	WHILE	do	quad	cubo
Ø	4					5		6		7						40			
1															HALT				
2													9	8					
3							R3						R3	R3					
4											10	11							
5	4					5		6		7						40			
6	17			16														44	46
7	18																		
8															R1				
9	4					5		6		7						40			
10	4					5		6		7						40			
11	17			16														44	46
12							22						9						
13		24							23										
14		R10	25		R10		R10		R10				R10	R10			R10		
15		R12	R12		R12		R12		R12				R12	R12			R12		
16	17			16														44	46
17		R14	R14		R14		R14		R14				R14	R14			R14		
18							R8						R8	R8					
19							R2						R2	R2					
20							R4						R4	R4					
21		24					R5						R5	R5					
22							R6						R6	R6					
23	4					5		6		7						40			

	id	+	*	(	)	begin	end	if	then	goto	:	:=	;	•	$	WHILE	do	quad	cubo
24	17			16														44	46
25	17			16														44	46
26		24			30														
27							R7						R7	R7					
28		R9	25		R9		R9		R9				R9	R9			R9		
29		R11	R11		R11		R11		R11				R11	R11			R11		
30		R13	R13		R13		R13		R13				R13	R13			R13		
31	4					5		6		7						40			
32	4					5		6		7						40			
33													9						
34	4					5		6		7						40			
35				16														44	46
36		24																	
37	17			16														44	46
38			25																
39	17			16															
40	17			16														44	46
41		24															42		
42	4					5		6		7						40			
43							R15						R15	R15					
44	17			16														44	46
45		R16	R16		R16		R16		R16				R16	R16			R16		
46	17			16														44	46
47		R17	R17		R17		R17		R17				R17	R17			R17		

Figura II.8 - Tabela SLR(1) de G_E

CAPÍTULO III

DESCRIÇÃO DA LINGUAGEM

III.1. ESTRUTURA DA LINGUAGEM

EXPAND II é uma linguagem de alto nível composta por um conjunto reduzido de declarações e comandos (linguagem base), o qual pode ser expandido através de macros sintáticas.

A expansão da linguagem é consequência da expansão da gramática base (causada pela inclusão de macros) que implica na extensão do analisador.

Uma macro sintática é uma construção pertencente a alguma categoria sintática da gramática base, a qual quando acrescentada à gramática base, resulta em uma gramática expandida, e a partir deste ponto tem-se uma nova linguagem denominada linguagem estendida, que é a união da linguagem base com a macro incluída.

A declaração de uma macro deverá sempre ser feita num contexto que aceite declarações, e compõem-se de ESTRUTURA e DEFINIÇÃO, onde a estrutura é a nova construção sintática que será adicionada a gramática base e a definição é um conjunto de construções (comandos e/ou declarações) que definem a estrutura em termos de linguagem estendida, sem contudo fazer referência a macro que está sendo declarada.

A utilização de uma macro dar-se-á sempre num contexto que aceite a categoria sintática especificada para a macro em sua declaração.

Sintaticamente, um programa EXPAND II é um bloco seguido por um "." (ponto), o qual determina o final físico do programa fonte. Um bloco, por sua vez, é constituído por uma lista de declarações (separadas por ";") seguida de uma lista de comandos (também separados por ";"), tudo encerrado entre as palavras BEGIN e END. A especificação sintática (gramática) da linguagem base encontra-se no apêndice 1.

Devido a característica experimental desta implementação e a semelhança sintática e semântica entre EXPAND II e a

linguagem ALGOL, |BURROUGHS 4|, |BURROUGHS 5|, |BURROUGHS 6| e |SEGRE 7| (no que segue será utilizado como referência apenas |BURROUGHS 4| por ser mais completo e ter sido mais utilizado), optou-se pela construção de um tradutor, ou seja, em vez de código de montagem ou de máquina, este analisador gera um programa na linguagem ALGOL. Assim sendo, a análise semântica será, quase que totalmente, feita pelo compilador ALGOL quando da compilação do programa gerado por este tradutor.

III.2. ESPECIFICAÇÃO DA LINGUAGEM BASE

III.2.1. FORMATO DOS PROGRAMAS

Um programa EXPAND II é formado por um conjunto de registros de 80 posições, sendo que as posições de 1 a 72 inclusive, devem ser usadas para o programa em si, e constituem o campo útil do registro, enquanto que as posições de 73 a 80 podem ser utilizadas para identificação e numeração dos registros, ou simplesmente deixadas em branco.

Os programas podem ser editados continuamente, isto é, um mesmo registro pode conter diversos comandos e/ou declarações; ou ainda, um comando ou uma declaração podem utilizar vários registros, desde que para isso não seja necessária a divisão de ítens sintáticos tais como identificadores, constantes, etc.

Por outro lado, não é necessário que todo o campo útil de um registro seja utilizado; em qualquer caso, um novo registro sempre será continuação do campo útil do registro que o precede.

Os comentários são introduzidos num programa EXPAND II com o uso do símbolo "%", o qual indica final do campo útil do registro, isto é, tudo o que vem após o símbolo "%" no registro em questão, será transparente ao analisador.

III.2.2. ELEMENTOS LÉXICOS

III.2.2.1. IDENTIFICADORES

Em EXPAND II, os identificadores podem ser simples ou concatenados, sendo que um identificador simples é formado por uma sequência de letras e dígitos onde o primeiro caracter deverá ser obrigatoriamente uma letra e o número máximo de caracteres não deverá ultrapassar 64, de forma que os excedentes serão ignorados; enquanto que um identificador concatenado é uma sequência de identificadores simples separados pelo símbolo "#" (jogo da velha), o tamanho total de um identificador assim constituído não tem implicações léxicas, entretanto os identi-

ficadores simples que o constituem devem satisfazer às regras de formação pré-estabelecidas.

Os caracteres utilizados na formação de identificadores são as letras maiúsculas de A a Z e os dígitos de 0 a 9, além do símbolo "#" usado em identificadores concatenados.

EXEMPLO: identificadores válidos - TOTAL
MEDIA#PONDERADA
TOT500
PROX#BLOCO1

identificadores inválidos - AB@
5PORCENTO
UM#DOIS#3
#ABC
VALOR##INICIAL

RESTRIÇÕES - Devido às características desta implementação, os identificadores de variáveis criadas pelo programador, além de serem distintos das palavras reservadas da linguagem EXPAND II, deverão também ser distintos das palavras reservadas da linguagem ALGOL|BURROUGHS 4| (para a qual o fonte será traduzido) e das temporárias criadas pelo analisador; o não cumprimento dessas restrições não constitui erro para o compilador EXPAND II, mas poderá causar erros sintáticos e/ou semânticos no programa Algol gerado. A lista completa das palavras que não podem ser utilizadas como identificadores de variáveis do programador encontra-se no apêndice 2.

Além disso, existe um limite quanto ao número de identificadores e o número de caracteres utilizados na formação dos mesmos suportados em um programa. Estes limites estão especificados na seção de limitações (Seção III.2.4) deste capítulo.

III.2.2. CONSTANTES

Devido as características desta implementação as constantes não serão analisadas como habitualmente, apenas serão guardadas na forma original, para posteriormente serem passadas para o programa Algol gerado por este analisador; portanto, as regras de formação a serem observadas são as da linguagem Algol |BURROUHGS 4|. Apesar disso, existe um limite quanto ao número máximo e quanto ao tamanho total (em caracteres) das constantes utilizadas em um programa, o qual está especificado na seção de limitações (Seção III.2.4) deste capítulo.

III.2.3. CARACTERES ESPECIAIS

Os caracteres especiais aceitos pela linguagem EXPAND II são: ␣ (espaço em branco) • , ; + - * / > < = : [ ] ( ) # (usado para concatenação de identificadores) e % (usado para comentários).

Além dos caracteres especificados acima, nas especificações transparentes ao analisador (como é o caso, por exemplo, das especificações de formato do comando WRITE) outros caracteres poderão ser utilizados; nestes casos serão aceitos todos os caracteres válidos para a linguagem ALGOL |BURROUGHS 4|.

III.2.3. ESTRUTURAS SINTÁTICAS

III.2.3.1. DECLARAÇÕES

As declarações fornecem informações acerca dos dados que serão utilizados nos comandos de um programa e sempre que existirem deverão ser feitas antes dos mesmos, considerando o escopo a que pertencem.

Em EXPAND II, pode-se declarar variáveis simples, array's unidimensionais (vetores) e constantes do tipo INTEGER ou REAL, rótulos (label's), procedimentos com e sem tipo, composição de declarações (através da declaração DECLARE), arquivos, e obviamente MACROS SINTÁTICAS (as quais podem ser declaradas explicitamente ou através da declaração ARQMACRO).

As declarações acima mencionadas, exceção feita à macros, serão analisadas apenas sob o aspecto sintático, uma vez que a análise semântica será efetuada apenas sobre o programa ALGOL gerado; entretanto, é importante observar que esta análise será efetuada e portanto o usuário deverá preocupar-se em observar as regras semânticas da linguagem ALGOL |BURROUGHS 4|. Do ponto de vista semântico, podemos adiantar que todas as variáveis utilizadas no programa deverão ser previamente declaradas e que sua utilização deverá considerar as características de cada uma (especificadas em sua declaração), bem como o escopo de sua declaração.

Quanto a declaração de macros, além da análise sintática e da análise específica inerente ao tratamento das mesmas, será efetuada a análise de escopo, isto é, uma macro só terá validade no bloco em que foi declarada e nos blocos internos a ele. A seção 3.1 deste capítulo é totalmente dedicada a declaração de MACROS.

Além das declarações inerentes a linguagem base, outras declarações, tais como array's bidimensionais, procedimentos com parâmetros, etc..., podem ser introduzidas através do uso de macros sintáticas; o limite para tal, reside na possibilidade de definir tais declarações em termos da linguagem base.

DECLARAÇÃO DE VARIÁVEIS SIMPLES E ARRAY'S

Variáveis simples e array's unidimensionais em EXPAND II podem ser do tipo inteiro ou real e são declarados através das declarações INTEGER e REAL respectivamente, cuja forma geral consiste da especificação do tipo, seguida de uma lista de variáveis separadas por ",". No caso de array's, a dimensão é dada através de uma expressão aritmética entre colchetes.

```
EXEMPLOS: INTEGER A, B, I#B1;
          REAL X[10], Z, Y[N];
          REAL  MEDIA#PONDERADA, VET#PRECO [2*(N + 1)];
```

DECLARAÇÃO DE CONSTANTES

Esta declaração atribui um tipo (INTEGER ou REAL) a uma variável e a inicializa através de uma expressão aritmética.

```
EXEMPLOS: INTEGER  N: 50;
          REAL  DESC: DESC1
          INTEGER  X: (A + B)*10
          REAL    PI: 3.1415
```

DECLARAÇÃO LABEL

Para possibilitar referências a um comando pode-se atribuir nomes ao mesmo; estes nomes são denominados rótulos e sintaticamente são identificadores simples ou concatenados, os quais devem ser declarados, um a um, através da declaração LABEL.

```
EXEMPLOS:  LABEL FIM;
           LABEL  FIM#BLOCO5;

CONTRAEXEMPLO: LABEL L1, L2;
```

DECLARAÇÃO DE PROCEDIMENTOS

A declaração de um procedimento é utilizada para definir um segmento de programa e dar-lhe um nome de modo que esse segmento possa ser ativado por uma chamada.

Em EXPAND II os procedimentos não possuem parâmetros e podem ou não ter tipo, caso tenham o mesmo deverá ser INTEGER ou REAL.

Os procedimentos sem tipo (também denominados rotinas) são procedimentos cujo corpo é um bloco e tem como função realizar uma tarefa específica; e como não há parâmetros, para que os resultados sejam propagados os mesmos devem ser atribuídos à variáveis globais. Este tipo de procedimento é ativado através do comando CALL.

Os procedimentos com tipo (também denominados funcões) são procedimentos que calculam e retornam um valor, usando para isso o próprio nome da função; o corpo de uma função é um bloco result, o qual sintaticamente é um bloco, mas difere deste pelo fato de retornar um valor (resultado) e suportar o comando RESULT.Estes procedimentos são ativados por chamadas através do próprio nome,quando este é usado como operando de uma expressão aritmética; e o valor retornado será do tipo inteiro ou real, dependendo do tipo atribuído ao procedimento em sua declaração.

EXEMPLOS:
```
PROCEDURE SOMA;
BEGIN
   ----
   ----
   ----
   ----
END;
INTEGER PROCEDURE MAIOR;
BEGIN
   ----
   ----
   ----
END;
```

DECLARAÇÃO COMPOSTA

Esta declaração engloba um conjunto de outras declarações separadas por ";" e encerradas entre as palavras DECLARE e END, com o objetivo de efetuar várias declarações simultaneamente em um único ponto do programa. A utilização desta declaração está diretamente associada com a declaração de macros, cuja categoria sintática seja DECL; entretanto, nada impede que a declaração DECLARE seja utilizada para outro fim, como por exemplo reunir declarações referentes a uma mesma estrutura ou comuns a um procedimento particular.

EXEMPLO:
```
DECLARE
    INTEGER I, X[50], TOT;
    REAL  MEDIA;
    INTEGER N: 50
END;
```

DECLARAÇÃO FILE

Esta declaração especifica os arquivos que serão utilizados no programa, bem como os atributos de cada um necessários para sua utilização.

Sintaticamente apenas a palavra FILE é analisada, e quando encontrada, o analisador percorre o programa fonte até encontrar um ";" guardando as informações encontradas, as quais devem satisfazer as regras da linguagem ALGOL |BURROUGHS 4| para declarações de arquivos, pois serão passadas na forma original para o programa ALGOL gerado.

Existem limites quanto ao número de declarações FILE suportadas por um programa, e também quanto ao tamanho (em caracteres) das especificações das mesmas; estes limites estão associados às limitações impostas aos comandos de entrada e saída e estão especificados na seção de limitações(Seção III.2.4) deste capítulo.

DECLARAÇÃO ARQMACRO

Esta declaração foi criada com o objetivo de evitar que em cada uso, o usuário necessite declarar as macros explicitamente. Sua função é fornecer ao analisador uma lista de nomes de macros (separadas por ","), cuja declaração completa encontra-se no diretório de macros (Apêndice 4); assim, a declaração das macros, cujos nomes encontram-se nessa lista, tornam-se disponíveis ao analisador.

Quanto ao tratamento, uma macro assim declarada, passará pela mesma análise que uma macro declarada explicitamente no programa fonte.

EXEMPLOS: ARQMACRO SE, ENQUANTO;
ARQMACRO PARA, REPITA, MATRIZ;

DECLARAÇÃO NEW

Esta declaração tem como objetivo criar um novo identificador, o qual não terá nenhuma relação com identificadores de mesmo nome utilizados em outros pontos do programa, uma vez que durante o processo de tradução cada referência a ele (no escopo de sua declaração) será substituída por um novo identificador criado pelo analisador. A forma geral desta declaração é NEW IDS, onde IDS deverá ser um identificador simples, nunca concatenado. Os identificadores declarados como NEW, deverão ser redeclarados através de outra declaração disponível, quando efetivamente tornam-se disponíveis.

EXEMPLO: NEW L1; NEW AUX;
LABEL L1;
INTEGER AUX;

III.2.3.2. COMANDOS

Como já foi citado, a linguagem EXPAND II é composta por um conjunto reduzido de comandos; e foi assim especificada com o objetivo de se definir uma linguagem necessária e suficiente (linguagem base), a qual pode ser expandida através do uso de macros sintáticas.

Apesar de poucos, os comandos em EXPAND II são comandos básicos (existentes na grande maioria das linguagens de programação de alto nível), a partir dos quais torna-se possível construir comandos mais poderosos (tais como IF-THEN-ELSE, WHILE-DO, FOR, etc...), bem como a criação de comandos elaborados para situações particulares e de interesse específico de um determinado usuário. Obviamente, o limite da criação de novos comandos reside na possibilidade de definí-los através da linguagem base e também no fato de que os mesmos devem ser aceitos por analisadores SLR(1).

Todo comando de um programa EXPAND II, pode ser identificado por um rótulo (que deverá ter sido declarado anteriormente em uma declaração LABEL), o qual deverá preceder o comando e estar separado deste pelo símbolo ":" (dois pontos). O objetivo dessa identificação é possibilitar referências a determinados comandos através do comando GOTO. Aqui, cabe observar que nem toda referência é semanticamente válida, embora o seja do ponto de vista sintático; por isso, o usuário deverá observar as regras de validade vigentes na linguagem Algol |BURROUGHS 4|, já que estas serão verificadas quando da compilação do programa ALGOL gerado.

A seguir, apresentaremos os comandos disponíveis na linguagem base, acompanhados de comentários e exemplos.

COMANDO DE ATRIBUIÇÃO

A forma geral deste comando é VAR:= ARIT, onde VAR é uma variável simples ou subscrita, ARIT é uma expressão aritmética e ":=" é o sinal de atribuição (significando que o valor de ARIT será atribuido a VAR).

EXEMPLOS: I := 0;
MAT[L] := (A + B)*C;
VET[MAT [I]] := 0.25

COMANDO DE DESVIO INCONDICIONAL (GOTO)

A forma geral deste comando é GOTO IDEN, onde IDEN deve ser um rótulo declarado em uma declaração LABEL antes de sua referência no programa e deve estar prefixando um único comando num determinado contexto (isto é, no mesmo contexto não é permitido que dois ou mais comandos tenham o mesmo rótulo).

Este comando causa um desvio incondicional para a instrução (o comando) rotulada por IDEN.

EXEMPLOS: GOTO LE#REGISTRO;
GOTO FIM;

COMANDO CONDICIONAL (COND)

A forma geral deste comando é:

COND $EXPR_1$ => COM_1;
$EXPR_2$ => COM_2;
⋮
$EXPR_N$ => COM_N;
=> COM_{N+1}
END

onde: $EXPR_1$, $EXPR_2$ ... $EXPR_N$ são expressões relacionais ou aritméticas; COM_1, COM_2 ... COM_{N+1} são comandos simples ou compostos; COND e END são os delimitadores do comando e o símbolo " => " é um separador interno usado para separar as expressões dos comandos.

Seu funcionamento pode ser assim explicitado: Se a $EXPR_1$ após avaliada resultar um valor verdade, o COM_1 será executado e o controle será desviado para a primeira instrução após o END correspondente ao COND; senão, a $EXPR_2$ será avaliada e se verdadeira o COM_2 será executado e o controle passará

para a primeira instrução após o END do COND; e assim sucessivamente até a $EXPR_N$. Se nesse transcurso nenhuma expressão avaliada apresentar resultado verdadeiro, o COM_{N+1} será executado incondicionalmente, já que não está subordinado a nenhuma condição; portanto, o COM_{N+1} é a alternativa existente quando todas as condições especificadas forem falsas, e por imposição sintática, deverá sempre estar presente - mesmo que seja um comando vazio.

Uma expressão relacional é sempre composta por duas expressões aritméticas separadas por um operador relacional, as quais estão subordinadas a ele. Os operadores relacionais disponíveis são: > (maior), < (menor), = (igual), >= (maior ou igual), <= (menor ou igual) e < > (diferente), todos com significado óbvio.

Quando EXPR for uma expressão aritmética, o resultado será considerado verdadeiro se o valor obtido após sua avaliação for diferente de Ø e falso em caso contrário.

EXEMPLOS: a) COND A > B => COM_1;
B > A => COM_2;
=> COM_3
END

Neste caso, o COM_1 será executado se A for maior que B; o COM_2 será executado se B for maior que A e o COM_3 será executado somente se A for igual a B, isto é, quando as condições existentes apresentarem valor resultado falso.

b) COND M >= N + 1 => COM_1;
N + 1 => COM_2;
=>
END

Aqui, podemos observar que se M for maior ou igual ao resultado da expressão N + 1, o COM_1 será executado; senão, se N + 1 for diferente de Ø será executado o COM_2 e em caso contrário nada será executado (note a presença do comando vazio).

Em ambos os casos, podemos notar que após o último comando (associado com a negação de todas as condições existen

tes) não existe ";", o delimitador neste caso será o próprio END.

Outra possibilidade, bastante interessante, é a formação de "ninhos" de comandos COND, isto é, um comando associado a uma determinada condição pode ser ou conter outro comando COND e este por sua vez pode conter novos comandos COND e assim sucessivamente, não havendo limite para este aninhamento.

```
EXEMPLO: COND A > B => COND A > C => COND A > D => COM1;
                                                => COM4
                                       END;
                                C > D => COM3;
                                      => COM4
                         END;
                B > C => COND B > D => COM2;
                                    => COM4
                         END;
                C > D => COM3;
                      => COM4
         END
```

Neste exemplo, temos que: o COM_1 será executado se "A" for o maior dos quatro valores; o COM_2 se o maior valor for "B"; o COM_3 se "C" for o maior valor e, finalmente, se "D" for o maior valor, o COM_4 será executado.

COMANDO CALL

A forma geral deste comando é CALL IDEN (onde IDEN deverá ser um identificador de procedimento sem tipo), e sua função é ativar o procedimento identificado por IDEN.

Apesar de pouco elegante, a palavra CALL foi introduzida por ser necessária para eliminar conflitos surgidos quando da construção do analisador SLR(1) para a gramática da linguagem EXPAND II.

COMANDO RESULT

A forma geral deste comando é RESULT ARIT, (onde ARIT é uma expressão aritmética) e sua função é estabelecer um resultado para um bloco result; este comando deverá aparecer sempre dentro de um bloco result, o qual poderá conter vários comandos RESULT. Se o bloco result em questão for o corpo de um procedimento com tipo (função) o comando RESULT especifica o valor resultado a ser atribuído à variável nome da função, resultado este que será inteiro ou real dependendo do tipo da função; senão, o valor resultado da expressão aritmética especificada pelo comando RESULT será atribuído a uma temporária criada pelo analisador, cujo tipo será sempre real.

```
EXEMPLO:  INTEGER PROCEDURE MAIOR;
          BEGIN
             COND X > Y => RESULT X;
                  X < Y => RESULT Y;
                        => RESULT Ø
             END;
          END;
```

O valor retornado por esta função será: X, Y ou Ø, dependendo se X for maior, menor ou igual a Y.

COMANDO COMPOSTO

Na linguagem EXPAND II, um comando composto é definido como sendo um bloco, e é utilizado para introduzir um conjunto de declarações (opcionais) e comandos num contexto que sintaticamente aceita apenas um comando.

COMANDOS DE ENTRADA E SAÍDA

Os comandos de E/S disponíveis em EXPAND II são os comandos READ e WRITE, os quais fornecem informações acerca dos arquivos em que as operações de leitura/impressão serão efetua

das, especificam a disposição dos dados de entrada e dos resultados de saída, além de especificarem a lista de variáveis cujo valor deverá ser lido/impresso.

Nesta implementação as informações a respeito dos arquivos e do formato para leitura/impressão serão guardadas na forma original, e só serão analisadas quando da compilação do programa ALGOL gerado por este analisador; assim sendo, as especificações de arquivo e formato associadas aos comandos READ e WRITE deverão estar de acordo com as regras vigentes na linguagem ALGOL |BURROUGHS 4| e, logicamente associadas aos tipos de variáveis de E/S permitidos pela linguagem EXPAND II.

A forma geral dos comandos de E/S é:

1 - READ (ARQFOR, LEES),

2 - WRITE (ARQFOR, LEES),

3 - WRITE (ARQFOR)

onde: ARQFOR é a parte referente as especificações de arquivo e formato, a qual é transparente a este analisador. Cabe ressaltar aqui, a possibilidade do uso de formato livre tanto pa leitura como para impressão, neste caso as regras a serem observadas também serão as da linguagem ALGOL |BURROUGHS 4|. No caso específico do comando WRITE, ARQFOR poderá conter literais (usados para cabeçalhos e mensagens explicativas), sendo que a opção 3 é usada exclusivamente para este fim.

LEES é a lista de expressões de entrada e saída (EES) a serem lidas ou impressas. Uma expressão de entrada será sempre uma variável simples ou subscrita, enquanto que uma expressão de saída poderá ser qualquer expressão aritmética válida na linguagem EXPAND II. Além disso, existe a opção FOR usada para leitura/gravação de array's, a qual tem a seguinte forma geral:

FOR IDEN:= $ARIT_1$ STEP $ARIT_2$ UNTIL $ARIT_3$ DO EES, onde IDEN deverá ser um identificador necessariamente do tipo INTEGER, $ARIT_1$, $ARIT_2$ e $ARIT_3$ deverão ser expressões aritméticas cujo resultado deverá ser do tipo INTEGER, e EES é uma nova expressão de E/S.

EXEMPLOS: READ (ENT, /, A, B, C)
WRITE (SAI, < "TOTAL GERAL = ", F5.2 >, TOT)
READ (ENT, < 10I2 >, FOR I:= 1 STEP 1 UNTIL 10 DO
IDADE[I])
WRITE (SAI, < "RESULTADOS FINAIS", /, >)

Devido as características do tratamento dado aos comandos READ e WRITE (e também a declaração FILE), o uso dos mesmos é limitado; este limite existe em função do tamanho das estruturas utilizadas para o armazenamento das especificações de arquivo e formato e está especificado na seção de limitações (Seção III.2.4 deste capítulo.

Ainda a respeito das especificações de arquivo e formato, chama-se a atenção do usuário para o cuidado que o mesmo deverá ter nessas especificações, com o intuito de evitar problemas causados por especificações errôneas, principalmente pela falta de delimitadores terminais das referidas especificações.

COMANDO VAZIO

É uma opção sintática para evitar a criação de comandos fictícios em contextos do programa que por imposição sintática necessitem de pelo menos um comando (e que do ponto de vista da lógica do programa nada deve ser feito), e principalmente para evitar erros causados pelo uso errôneo de um ";" após uma lista de comandos.

III.2.3.3. EXPRESSÕES ARITMÉTICAS

Uma expressão aritmética é composta de operandos e operadores dispostos de acordo com as regras sintáticas da gramática.

Um operando na linguagem EXPAND II, pode ser uma variável simples ou subscrita, uma constante, uma expressão entre parênteses, uma chamada de função (procedimento com tipo) ou ainda um bloco result.

Os operadores disponíveis, em ordem decrescente de prioridade, são:

1) - e + (UNÁRIOS)
2) *, /, DIV e MOD
3) + e -

Nesta implementação é analisada apenas a estrutura sintática das expressões, entretanto, deve-se lembrar que as mesmas serão analisadas semanticamente no programa ALGOL gerado e por isso, devem ser observadas as regras semânticas e de avaliação de expressões da linguagem Algol |BURROUGHS 4|, quando da construção das expressões aritméticas usadas no programa EXPAND II.

Como foi citado acima, um dos possíveis operandos de uma expressão aritmética é o bloco result, o qual do ponto de vista sintático é idêntico a um bloco, a diferença reside no contexto em que os mesmos podem ocorrer bem como no significado semântico de cada um. A função de um bloco result é definir um valor resultado, o qual será sempre do tipo real e fornecido através de um comando RESULT pertinente ao bloco em questão. Se o fluxo de execução do bloco não passar por nenhum comando RESULT, o resultado do referido bloco será $\emptyset.\emptyset$.

Do ponto de vista semântico, um bloco result pertencente a uma expressão (relacional ou aritmética) associada ao comando COND, será tratado como uma função, a qual será avaliada exatamente quando o operando, que no caso era um bloco result e que agora é uma chamada de função, for referenciado para execução ; já os blocos result pertinentes a expressões aritméticas associadas com os comandos de atribuição, comandos RESULT ou comandos WRITE, serão avaliados imediatamente antes da execução

dos comandos que os contêm (como se fossem comandos compostos), portanto, ao definir uma expressão que utilize blocos result como operandos, o usuário deverá considerar que a ordem de cálculo dos elementos (operandos) que a compõem será arbitrária e por isso o programa que depender, para seu bom funcionamento, de uma ordem específica de avaliação será considerado errôneo.

Cabe ressaltar ainda, que uma mesma expressão aritmética pode conter vários blocos result aninhados ou não, também neste caso a avaliação será idêntica à explicitada no parágrafo anterior, observando-se a ordem bem como o escopo de cada ocorrência. O termo "AVALIAÇÃO" acima utilizado é força de expressão, uma vez que o programa EXPAND II será traduzido para um programa ALGOL; sendo que nessa tradução é que serão efetuados os procedimentos necessários para que os blocos result sejam avaliados da maneira acima especificada quando da execução do programa ALGOL gerado.

EXEMPLOS:
```
(A - C) * BD#E
-B DIV C / (A MOD C)
0.5 * BEGIN
        READ (ENT, /, A, B);
        COND B > A => RESULT A;
                   => RESULT B
        END
      END + Z
```

RESTRIÇÃO: Um bloco result não poderá ser usado como operando de expressões aritméticas associadas às declarações de array's e de constantes. A violação desta restrição não causará erro sintático no programa EXPAND II, entretanto, o programa Algol gerado poderá apresentar erros sintáticos e/ou semânticos.

III.2.4. LIMITAÇÕES

A quantidade bem como o tamanho total (em número de caracteres) dos identificadores, das constantes e das especificações de arquivo e formato, foram limitados nesta implementação em função do tamanho das estruturas utilizadas para o armazenamento das mesmas. Entretanto, sempre que necessário e desde que autorizado para tal, o usuário poderá redefinir estes limites, bastando para isso alterar o fonte do compilador, compilando-o novamente.

A tabela abaixo apresenta os limites definidos na implementação e a variável a ser utilizada em cada caso (esta variável foi introduzida no fonte do compilador através da cláusula DEFINE |BURROUGHS 4| e encontra-se no princípio do mesmo).

LIMITAÇÕES	MÁXIMO	VARIÁVEL A SER REDEFINIDA
Número de Identificadores	351	NMAXIDEN
Tamanho Total dos Identificadores	3600	MAXTAMIDEN
Número de Constantes	256	NMAXCONST
Tamanho Total das Constantes	512	MAXTAMCONST
Número de Especificações de Arquivo e Formato	256	NMAXARQFOR
Tamanho Total das Especificações de Arquivo e Formato	4096	MAXTAMAF

Tabela III.1 - Limitações das Estruturas

Além das limitações especificadas acima, existe uma série de outras limitações referentes a macros, as quais serão especificadas na seção de limitações do capítulo seguinte.

III.3. DECLARAÇÃO E UTILIZAÇÃO DE MACROS

III.3.1. DECLARAÇÃO

A declaração de macros deve ser feita em um contexto de programa que sintaticamente aceite declarações e só estará disponível ao usuário no escopo de sua declaração.

A declaração de uma macro implica na expansão da gramática e consequentemente na extensão do analisador de forma dinâmica, isto é, a cada saída de um bloco, a gramática bem como o analisador voltam a ser como eram, quando da entrada no referido bloco.

A forma geral para declaração de macros é:

MACRO <estrutura da macro>
DEFINE <definição da macro>
ENDMACRO

onde:

- MACRO, DEFINE e ENDMACRO são os delimitadores da declaração;
- <estrutura da macro> é um conjunto de símbolos terminais, não-terminais e parâmetros, onde o primeiro símbolo será necessariamente um dos símbolos não-terminais da gramática base e sua função é especificar a categoria sintática da macro; o segundo símbolo da estrutura deverá ser um identificador (que não seja palavra reservada), o qual será o nome da macro; o restante da estrutura é uma combinação de parâmetros e símbolos terminais, de forma que nunca apareçam parâmetros consecutivos. Um parâmetro é composto por um símbolo não-terminal, o qual especifica a categoria sintática do referido parâmetro, e por um identificador, o qual não deverá ser palavra reservada e tem como função identificar a ocorrência do parâmetro em questão. A estrutura da macro pode conter vários parâmetros com a mesma categoria sintática, entretanto, os identificadores associados deverão ser distintos. Os símbolos terminais usados na estrutura da macro, nada mais são que os ítens sintáticos (identificadores, constantes e símbolos especiais) da gramática base. Os identificadores que não forem palavras reservadas e que foram usados na estrutura da macro (exceto aque

les associados aos parâmetros) passarão a ser palavras reservadas no escopo de validade da macro.

<definição da macro> é um conjunto de símbolos terminais e parâmetros (sem a especificação da categoria), o qual deve formar um segmento de programa derivável na gramática atual (gramática base + o conjunto de macros disponíveis no momento) a partir da categoria especificada para a macro. Nesta implementação não é permitido a utilização de uma macro na sua própria definição, entretanto é possível utilizar outras macros desde que previamente declaradas. Os parâmetros (que na definição são representados apenas pelo seu identificador) da definição, ao contrário da estrutura, podem se repetir e devem estar associados aos parâmetros da estrutura. A definição de uma macro pode conter a declaração de outras macros (desde que a referida definição, sintaticamente, aceite declarações), neste caso a macro declarada internamente pode utilizar, tanto na estrutura como na definição, os parâmetros da macro mais externa; quando este recurso for utilizado, a macro interna será declarada (e consequentemente tornar-se-á disponível ao usuário) somente quando a macro mais externa for referenciada (utilizada).

EXEMPLO:
```
MACRO COM SE EXPR E1 ENTÃO COM C1 SENAO COM C2
DEFINE COND E1 => C1;
               => C2
       END
ENDMACRO
```

onde:

- MACRO - indica o início da declaração da macro
- COM - (primeira ocorrência) especifica a categoria sintática da macro
- SE - é o nome atribuído a macro
- EXPR E1 - é o primeiro parâmetro da macro, onde EXPR é a sua categoria sintática e E1 seu identificador
- ENTAO e SENAO - são delimitadores internos da macro e serão considerados palavras reservadas enquanto a macro estiver disponível.

COM C1 e COM C2 - são, também, parâmetros da macro, onde COM é a categoria sintática de ambos e C1 e C2 são seus respectivos identificadores.

DEFINE - indica o fim da estrutura da macro e o início de sua definição, a qual é constituída de tudo o que se encontra entre o DEFINE e o ENDMACRO, que nada mais é que uma representação da estrutura em termos da linguagem atual.

ENDMACRO - indica o fim da declaração da macro.

Cabe ressaltar aqui a importância dos delimitadores MACRO, DEFINE e ENDMACRO cuja ausência pode ter consequência desastrosa no que diz respeito à integridade da macro declarada e a continuidade da análise do programa que a contém.

III.3.2. UTILIZAÇÃO

Após sua declaração, uma macro correta torna-se disponível ao usuário no escopo em que foi declarada. Para utilizá-la, o usuário deverá referenciá-la num contexto de programa que aceite sua categoria sintática, através de sua estrutura sem a especificação da categoria e substituindo os parâmetros por segmentos de programas que pertençam a categoria sintática dos mesmos.

Logo, é responsabilidade do usuário referenciar a macro de acordo com a estrutura especificada na declaração, bem como garantir que os segmentos de programa que substituirão os parâmetros ocorram na mesma ordem da declaração e sejam da mesma categoria sintática.

No caso de macros internas (declaradas na definição de outras macros), as mesmas só podem ser referenciadas (utilizadas) após a macro externa a que estão subordinadas ter sido referenciada, mesmo porque, em outra situação essas macros não estarão disponíveis.

Cabe ainda ressaltar que os parâmetros reais numa referência a uma macro podem ser ou conter novas referências a mesma ou a outras macros disponíveis, desde que a categoria das mesmas seja compatível com a categoria dos parâmetros em questão.

EXEMPLOS:

- utilização da macro *SE* declarada na seção anterior:

  a) *SE* A >= B
     ENTAO MAIOR:= A
     SENAO MAIOR:= B

  onde: A >= B é a expressão que substituirá o parâmetro identificado por E1, cuja categoria é EXPR.
  MAIOR:= A e MAIOR:= B são comandos que substituirão os parâmetros identificados por C1 e C2, respectivamente.

  A expansão dessa referência a macro *SE*, resultaria no seguinte segmento da linguagem base:

```
COND A >= B => MAIOR:= A;
            => MAIOR:= B
END
```

b)
```
SE INDICE > 0
ENTAO SE INDICE <= 500
      ENTAO CALL PESQTAB1
      SENAO CALL PESQTAB2
SENAO WRITE (SAI,< "INDICE INVALIDO", I4 >, INDICE);
```

Note que neste caso o parâmetro C1 da primeira ocorrência da macro *SE* é uma nova referência a macro *SE*; isto é possível, porque tanto o parâmetro C1 como a macro *SE* possuem a mesma categoria sintática.

III.4. UTILIZAÇÃO DO COMPILADOR

III.4.1. OPÇÕES

As opções de compilação disponíveis, estão relacionadas com o fluxo de execução e com a emissão da listagem do fonte; e são as seguintes:

OPÇÃO 1 - Compila, lista e traduz para o ALGOL.

OPÇÃO 2 - Compila e traduz para o ALGOL (neste caso serão listadas apenas as mensagens de erro).

OPÇÃO 3 - Compila e lista.

A opção escolhida pelo usuário deverá ser especificada na primeira coluna do primeiro registro do arquivo que contém o fonte EXPAND II (observe que o primeiro registro deverá conter apenas a opção).

Além disso, no que diz respeito a declaração de macros, o usuário poderá declará-las explicitamente no fonte ou então utilizar o DIRETÓRIO DE MACROS (vide seção IV.5 e Apêndice 4), declarando as macros através da declaração ARQMACRO (seção III.2.3.1).

III.4.2. COMANDOS DE CONTROLE

Para ativar o compilador EXPAND II, o usuário deverá utilizar o seguinte comando de controle:

RUN (COS99104) OBJECT/EXPAND II

Os arquivos, juntamente com seus atributos (considerados default nesta implementação), manipulados pelo compilador EXPAND II são os seguintes:

ARQUIVOS	ATRIBUTOS
ENT	KIND = PACK, FILETYPE = 7, FAMILYNAME = USERPACK
SAI	KIND = PACK, PROTECTION = SAVE
ARQGER	KIND = PACK, PROTECTION = SAVE, FILEKIND = ALGOL-SYMBOL, BLOCKSIZE = 800, MAXRECSIZE = 80, UNITS = CHARACTER
ARQMACRO	KIND = PACK, TITLE = DIRECTORIO/MACROS, FILETYPE = 7, FAMILYNAME = USERPACK

onde:

ENT	é o arquivo que deverá conter o programa EXPAND II a ser compilado.
SAI	é o arquivo (gerado pelo compilador) que conterá a listagem do fonte juntamente com o resultado da compilação.
ARQGER	é o arquivo (gerado pelo compilador) que conterá o programa ALGOL gerado a partir do programa EXPAND II de entrada.
ARQMACRO	é o arquivo que contém o diretório de macros.

Os arquivos ENT, SAI e ARQGER poderão ter seus atributos alterados de acordo com as necessidades do usuário (ficando sob sua responsabilidade o resultado das alterações efetuadas), bastando para isso especificar os novos atributos juntamente com o comando usado para ativar o compilador.

EXEMPLO:

```
RUN (COS99104) OBJECT/EXPAND II;
FILE ENT (TITLE = NOME1, FAMILYNAME = PACK);
FILE SAI (TITLE = NOME2);
FILE ARQGER (TITLE = NOME3)
```

CAPÍTULO IV

TRATAMENTO DE MACROS

Várias rotinas desta implementação são dedicadas ao tratamento de macros, que é o objetivo maior deste trabalho, e serão abordadas neste capítulo.

Antes de abordarmos o tratamento de macros em si, faz-se necessário uma descrição das estruturas de dados utilizadas para armazenamento das macros e de informações diversas a elas relacionadas.

Basicamente, todas as estruturas usadas pelo analisador estão envolvidas com o tratamento de macros numa ou em outra etapa e com maior ou menor intensidade; entretanto, algumas estruturas estão mais intimamente ligadas ao tratamento de macros pois são utilizadas praticamente em todas as etapas do referido tratamento. Estas estruturas são: TABSIMB, VETMACRO, DESCMACRO e DESCPAR as quais são descritas a seguir:

TABSIMB - é a tabela de símbolos, a qual guarda para cada identificador, seu nome, sua representação interna e uma informação extra dizendo se o referido identificador é nome de macro, identificador de parâmetro ou delimitador de macro, além de um ponteiro para um descritor se o identificador for nome de macro ou identificador de parâmetro.

VETMACRO - é um vetor que contém os elementos que compõem a estrutura e a definição da macro (a representação interna dos mesmos) e é formado por dois campos, onde o primeiro contém a representação interna do elemento e o segundo contém informações a respeito do elemento do primeiro campo (tais como apontadores, ordem dos parâmetros, etc...).

DESCMACRO - é o descritor da macro e contém as seguintes informações: nome, categoria, tamanho da estrutura, tamanho da definição, ponteiros para a posição inicial da estrutura e da definição no vetor VETMACRO, além do nome da macro ex

terna a que pertence (se pertencer a alguma) e de uma informação extra a respeito da existência ou não de macros internas a ela.

DESCPAR - é o descritor de parâmetros e contém as seguintes informações: categoria, nome da macro a que pertence, ordem de sua ocorrência e um ponteiro para o descritor da macro que o contém.

Além das estruturas descritas acima, outras estruturas serão abordadas no decorrer deste capítulo sempre que se fizer necessário.

Nas diversas seções deste capítulo, serão usados exemplos juntamente com comentários e ilustrações julgados necessários para facilitar a compreensão dos diversos procedimentos descritos e também para possibilitar uma visão real do processo de macro-expansão e de seus efeitos sobre o analisador. Estes exemplos na medida do possível estarão relacionados entre si; além disso serão sempre exemplos reais, isto é, criados para este analisador.

IV.1. ESTRUTURA

Como vimos no capítulo anterior, a estrutura de uma macro é composta por um conjunto de símbolos terminais e parâmetros (onde um parâmetro é formado pela combinação de um não-terminal e de um identificador), o qual define sintaticamente uma nova construção da linguagem.

A estrutura da macro é tratada pela procedure varre-macro, a qual é ativada quando a palavra MACRO é encontrada, e sua função é analisar a estrutura da macro e guardar informações da mesma, se esta estiver correta.

De maneira geral as regras de formação, verificadas durante esta análise, são as seguintes:

- O primeiro elemento deverá ser um dos símbolos não-terminais da gramática, o qual especifica a categoria sintática da macro.
- O segundo elemento, nome da macro, deverá ser um identificador que não seja palavra reservada.
- Todo parâmetro deve ser composto por um símbolo não-terminal (o qual estabelece a categoria sintática do parâmetro) e de um identificador, o qual identifica o referido parâmetro e não deverá ser palavra reservada.
- Os identificadores de parâmetros devem ser distintos, exatamente para distinguir a ocorrência dos mesmos.
- Os parâmetros não devem aparecer consecutivamente, isto é, deve existir pelo menos um separador entre dois parâmetros.

Durante a análise, além da verificação das regras acima, são efetuados os seguintes procedimentos:

- Transformação dos identificadores utilizados, exceto os de parâmetros e os de palavra reservada, em novas palavras reservadas, isto é, criação de novos terminais para a gramática, os quais existirão apenas no escopo em que a macro está sendo declarada. Para maior clareza do usuário, é desejável que os identificadores utilizados na estrutura, sempre que possível, sejam identificadores novos; entretando, é aceitável identificadores já utilizados, desde que os mesmos, no momento

de sua utilização, estejam de acordo com as necessidades e restrições de cada caso.

- Armazenamento de informações a respeito do nome, dos delimitadores (que forem identificadores) e dos identificadores de parâmetros da macro na tabela de símbolos (TABSIMB).

- Armazenamento dos elementos que compõem a estrutura da macro no vetor VETMACRO.

- Montagem dos descritores dos parâmetros (DESCPAR).

- Montagem parcial do descritor da macro, (DESCMACRO) de acordo com as informações disponíveis até o momento.

A existência de qualquer erro na estrutura da macro, implica imediatamente no cancelamento da análise da macro, sendo que o usuário será notificado do erro ocorrido através de uma mensagem emitida pelo analisador. Por outro lado, se a estrutura estiver correta, o próximo passo será a análise da definição da macro.

EXEMPLO: Seja a seguinte estrutura de macro:

MACRO COM *SE* EXPR E1 *ENTÃO* COM C1 *SENÃO* COM C2

após sua análise, a situação das estruturas envolvidas seria:

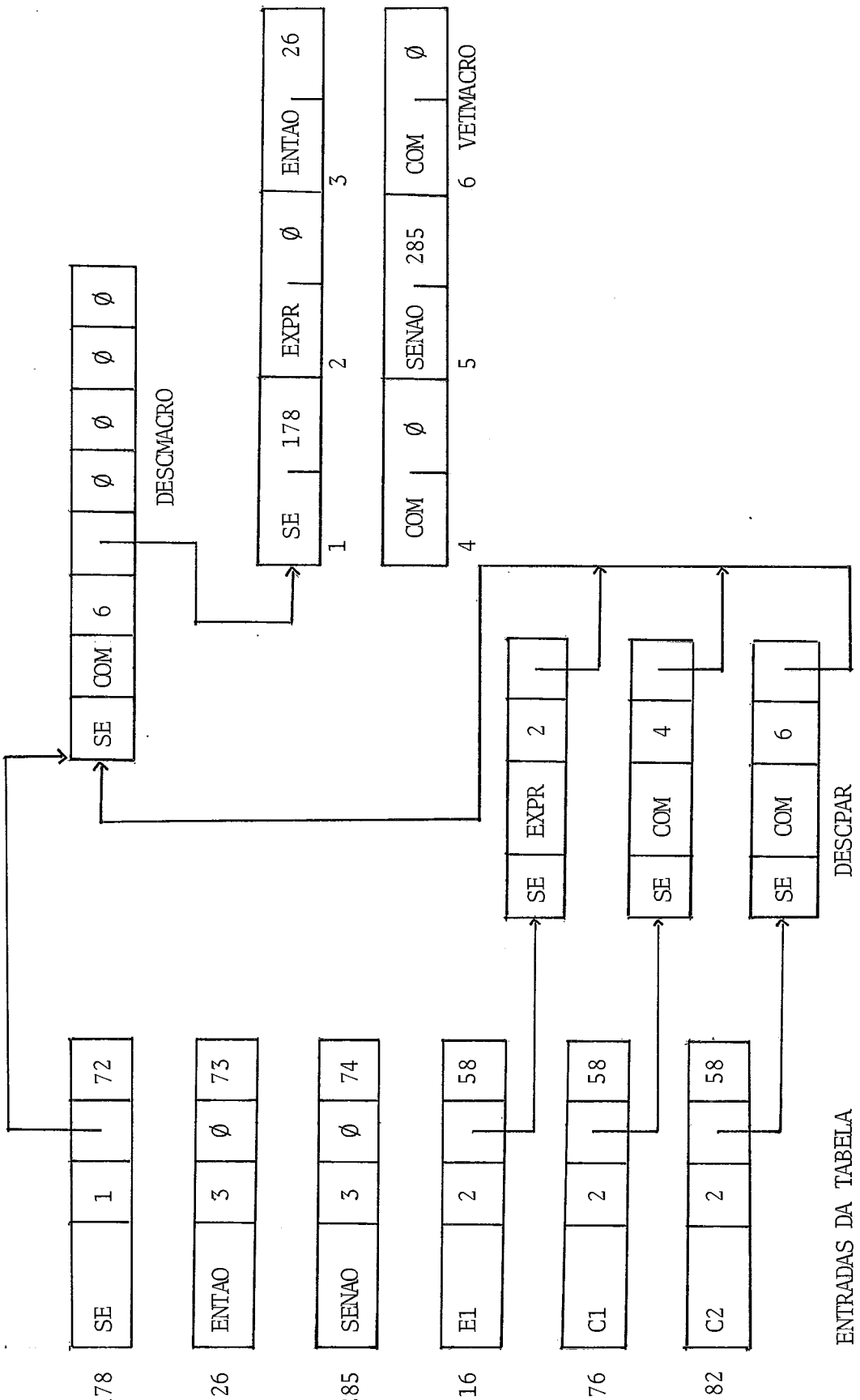


Figura IV.1 - Situação das Estruturas de Dados, Após a Análise da Estrutura da Macro *SE*.

IV.2. DEFINIÇÃO

A definição da macro é uma representação de sua estrutura em termos da linguagem estendida (linguagem base + conjunto de macros disponíveis) e é composta por um conjunto de símbolos terminais, incluindo identificadores de parâmetros.

A definição da macro é tratada pela procedure defmacro, a qual é ativada quando a palavra DEFINE (que delimita a estrutura e a definição) é encontrada, e sua função, obviamente, é analisar a definição da macro e guardar informações da mesma obtidas durante a análise.

A análise da definição consiste basicamente em armazenar os elementos que compõem a definição no vetor VETMACRO e concluir a montagem do descritor da macro, observando os seguintes pontos:

- Quando um identificador de parâmetro é encontrado,o mesmo não será inserido no vetor VETMACRO, em vez disso será inserida sua categoria (obtida do descritor do parâmetro em questão) no campo 1 e o número da ordem de sua ocorrência na estrutura da macro no campo 2 do referido vetor.

- Se a categoria da macro for DECL (declaração) é possível encontrar-se em sua definição a declaração de uma nova macro; caso isto ocorra, a declaração da nova macro será transparente ao analisador neste ponto e a única ação efetuada é a inserção dos elementos que a compõem (tanto os da estrutura como os da definição, além dos delimitadores MACRO, DEFINE e END MACRO) no vetor VETMACRO, os quais serão objeto de uma análise futura.

- Quando a macro em análise for uma macro interna a outra, sua estrutura e/ou definição eventualmente poderão conter identificadores de parâmetros da macro externa, neste caso é inserido em VETMACRO além da categoria do referido parâmetro, o nodo raiz da sub-árvore do parâmetro real correspondente; como foi dito, uma macro interna a outra só será analisada quando a macro externa for utilizada (no momento de sua expansão), daí a possibilidade de se fazer referências aos parâmetros reais utilizados.

- A utilização de outras macros na definição da macro

em análise, não requer nenhum tratamento especial uma vez que as mesmas, se disponíveis, são consideradas construções normais da linguagem estendida.

Durante a análise da definição podem ser detectados alguns erros, tais como: existência de parâmetros sem correspondência na estrutura (a situação inversa não causa erro, entretanto não faz sentido); utilização de identificadores de parâmetros que não sejam da macro em questão nem da macro externa a ela; falta do delimitador ENDMACRO (a qual somente será notada quando o vetor VETMACRO estiver cheio, o final do programa fonte for detectado, ou uma situação de erro ocorrer).

Em qualquer dos casos acima, o usuário será notificado através de uma mensagem de erro e a análise da referida macro será cancelada. Todavia, se nenhum erro for detectado, o próximo passo será a extensão do analisador.

EXEMPLO: Seja a seguinte definição (correspondente a estrutura da macro usada na seção anterior):

```
DEFINE COND E1 => C1;
               => C2
       END
ENDMACRO
```

após sua análise, teríamos as estruturas envolvidas, na seguinte situação:

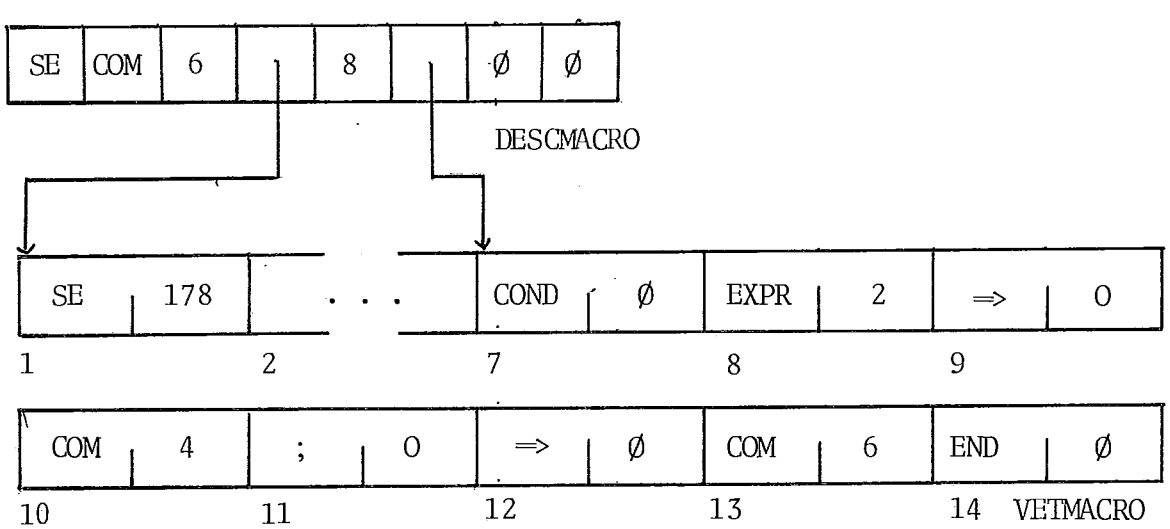


Figura IV.2 - Situação das Estruturas DESCMACRO e VETMACRO após a Análise da Definição da Macro *SE*.

IV.3. EXTENSÃO DO ANALISADOR

Como o próprio nome sugere, nesta etapa as tabelas utilizadas pelo analisador são aumentadas para que a macro em análise torne-se sintaticamente disponível e analisável; em outras palavras, é incluída uma nova produção (construção sintática)na gramática e as tabelas do analisador são aumentadas para que o mesmo suporte esta nova construção sintática. A extensão é feita pela procudure expandtab, cujo procedimento é descrito a seguir:

Inicialmente, antes da extensão propriamente dita,é verificado se a definição da macro em questão é derivável na gramática atual, o que é feito da seguinte forma:

- Interrompe-se momentaneamente o processo de análise sintática normal, guardando informações que permitam o retorno ao ponto atual de análise.

- Coloca-se na pilha sintática o estado auxiliar correspondente ao não-terminal que especifica a categoria sintática da macro; os estados auxiliares foram construídos de acordo com o algoritmo de construção dos estados auxiliares (seção 2, capítulo II) antes da implementação, e encontram-se armazenados no vetor NAOTERMINAL.

- Acrescenta-se na definição da macro um símbolo terminal que seja FOLLOW (seguidor válido) da categoria da macro (isto é feito para possibilitar as reduções durante a análise da definição).

- Ativa-se o analisador sintático (PARSER) para proceder a análise sintática da definição da macro. Quando ativado nesta situação, o PARSER utiliza a procedure PEGABETA, em vez da procedure SCANNER normalmente usada, para obter os itens sintáticos (tokens) necessários para a análise e inibe o processo de geração de árvores (quando da redução por produções da gramática base) e o processo da macro-expansão (quando da redução por produções correspondentes a macros).

Se ao final desta análise o elemento do topo da pilha sintática for o estado auxiliar da categoria da macro (empilhado anteriormente) e o token corrente for o não-terminal que es

pecifica a categoria da mesma, conclui-se que a definição da macro em questão é derivável na gramática estendida atual, neste caso, volta-se à situação anterior de análise e inicia-se os procedimentos de extensão.

Se, por outro lado, durante a análise ocorreu algum erro sintático ou no final da análise não se obteve a configuração acima especificada, conclui-se que a definição da macro não é derivável, neste caso, o usuário é notificado, os procedimentos de extensão são suspensos e a macro será ignorada. A figura IV.3 apresenta, através de configurações do analisador, a análise sintática da definição da macro *SE* tomada como exemplo.

CONFIGURAÇÃO	OBS
γ 152 cond EXPR => COM ; => COM end ; δ	1
γ 152$_{cond}$ 11 EXPR => COM ; => COM end ; δ	2
γ 152$_{cond}$ 11$_{EXPR}$ 35 => COM ; => COM end ; δ	
γ 152$_{cond}$ 11$_{EXPR}$ 35$_{=>}$ 72 COM ; => COM end ; δ	2
γ 152$_{cond}$ 11$_{EXPR}$ 35$_{=>}$ 72$_{COM}$ 102 ; => COM end ; δ	
γ 152$_{cond}$ 11$_{EXPR}$ 35$_{=>}$ 72$_{COM}$ 102; 126 => COM end ; δ	
γ 152$_{cond}$ 11$_{EXPR}$ 35$_{=>}$ 72$_{COM}$ 102; 126$_{=>}$ 36 COM end ; δ	2
γ 152$_{cond}$ 11$_{EXPR}$ 35$_{=>}$ 72$_{COM}$ 102; 126$_{=>}$ 36$_{COM}$ 73 end ; δ	
γ 152$_{cond}$ 11$_{EXPR}$ 35$_{=>}$ 72$_{COM}$ 102; 126 LCOND end ; δ	
γ 152$_{cond}$ 11$_{EXPR}$ 35$_{=>}$ 72$_{COM}$ 102; 126$_{LCOND}$ 134 end ; δ	
γ 152$_{cond}$ 11 LCOND end ; δ	
γ 152$_{cond}$ 11$_{LCOND}$ 34 end ; δ	
γ 152$_{cond}$ 11$_{LCOND}$ 34$_{end}$ 71 ; δ	
γ 152 COM ; δ	3

Figura IV.3 - Análise Sintática da Definição da Macro *SE*

Observarções anotadas ao longo das configurações:

(1) Situação inicial do teste sintático a ser realizado; onde:
γ é o conteúdo da pilha sintática até o momento;
152 é o estado auxiliar correspondente ao não-terminal COM (categoria sintática da macro em questão);
cond...end é a definição da macro *SE* (a qual será analisada);
; é um símbolo ∈ FOLLOW(COM);
δ é a entrada normal do analisador.

(2) Observe que em vez do identificador do parâmetro especificado na definição, é utilizado o não-terminal que especifica sua categoria.

(3) Configuração final da análise da definição, de onde conclui-se que a definição da macro em questão é derivável. Neste ponto, a configuração do analisador volta a ser (γ, δ), isto é, volta-se a situação anterior ao teste sintático da definição.

Posteriormente, começa-se a extensão propriamente dita, na qual são utilizadas todas as estruturas de dados envolvidas no processo de análise sintática, o vetor que contém os elementos da macro, o descritor da macro e uma série de estruturas auxiliares que serão mencionadas quando se fizer necessário.

O primeiro passo da extensão é a criação de um conjunto de listas (P_1, P_2 ... P_k) de estados temporários, correspondentes aos ítens LR(Ø) obtidos da estrutura da macro, e pré-existentes (fase 3 do algoritmo de expansão descrito no capítulo II); em seguida é atualizado o FOLLOW dos não-terminais (para incluir o efeito da macro em questão) de acordo com a fase 5 do algoritmo de expansão. A próxima etapa é a construção dos estados novos (continuação da fase 3 do algoritmo de expansão) a partir das listas P_1, P_2, ... P_k, criadas anteriormente e das novas listas possivelmente criadas nesta fase; para isso, inicialmente é montada uma matriz auxiliar (MATUNIAO) na qual as linhas são os símbolos terminais e não-terminais da gramática atual, as colunas são os estados que compõem uma determinada lista e o conteúdo de cada posição será a indicação de uma ação SHIFT, GOTO, REDUCE ou ERRO obtida a partir do símbolo-linha no estado-coluna em questão. Em seguida começa-se a construção de um novo estado em função da matriz recém-criada, e é nesta

fase que se detectará a necessidade ou não de se acrescentar novos estados às listas existentes ou até mesmo de se criar novas listas (isto ocorrerá quando, para um mesmo símbolo existir mais de uma ação SHIFT ou GOTO para estados distintos, na lista de estados em questão); após a análise de cada símbolo (isto é, de cada linha da matriz MATUNIAO), se necessário, é criada uma ação SHIFT/GOTO para o referido símbolo no estado que está sendo construído e no final da análise de cada lista (ou equivalentemente da matriz MATUNIAO) é verificada a necessidade ou não de se incluir uma ação de redução no estado recém-criado e se necessário, são registradas as informações necessárias para tal.

A etapa de criação de novos estados estará concluída quando todas as listas, pré-existentes ou criadas nesta etapa, tiverem sido analisadas. Durante a criação de estados novos é verificada a existência ou não de conflitos SHIFT-REDUCE (transições SHIFT e REDUCE simultaneamente com o mesmo símbolo) e REDUCE-REDUCE (presença de transições REDUCE distintas para o mesmo símbolo num mesmo estado); se algum conflito foi detectado, significa que a gramática aumentada não é SLR(1) (portanto não é analisável por este analisador), neste caso o usuário é notificado, o processo de extensão é interrompido e a macro será ignorada. Deve ser notado aqui, que a presença de mais de uma ação SHIFT/GOTO para um determinado símbolo em uma determinada lista, não constitui conflito SHIFT-SHIFT, isto porque nestes casos será criado um novo estado, igual a união dos vários estados alcançados em cada ação SHIFT/GOTO existentes. Todavia, se durante a criação dos estados novos nenhum conflito surgir, a gramática aumentada é SLR(1) (portanto analisável por este analisador) e a macro que estava sendo analisada passa a ser uma nova construção sintática da gramática, disponível ao usuário.

Finalmente, o processo de extensão do analisador encerra-se com a criação de transições (com o nome da macro) para os estados recém-criados, o que é feito da seguinte forma: para cada estado q, pré-existente ou recém-criado tal que (q, A) é definido (onde A é o não-terminal que especifica a categoria da macro), cria-se uma transição $(q, X_1) = P_1$, onde X_1 é o nome da

macro e P_1 é o primeiro estado novo criado.

Concluída a extensão, as tabelas do analisador (tais como tabela de ação, tabela de estados, tabela de produções, etc...) já estarão atualizadas, uma vez que a atualização é efetuada gradativamente de acordo com as informações disponíveis em cada momento.

No que segue, as várias fases que compõem o processo de extensão do analisador, são exemplificadas através dos resultados da extensão causada pela introdução da macro *SE* (especificada nas seções anteriores deste capítulo) à gramática base (no caso,a gramática da linguagem EXPAND II).

1. CRIAÇÃO DE LISTAS

P_1 = {1001, 168}, onde: 1001 corresponde ao ítem LR(0) (1001, EXPR) = 1002.
168 é o estado auxiliar correspondente ao não-terminal EXPR ou equivalentemente q(EXPR).

P_2 = {1002} , onde: 1002 corresponde ao ítem LR(0) (1002, *ENTAO*) = 1003.

P_3 = {1003, 152}, onde: 1003 corresponde ao ítem LR(0) (1003, COM) = 1004.
152 é o estado auxiliar q(COM).

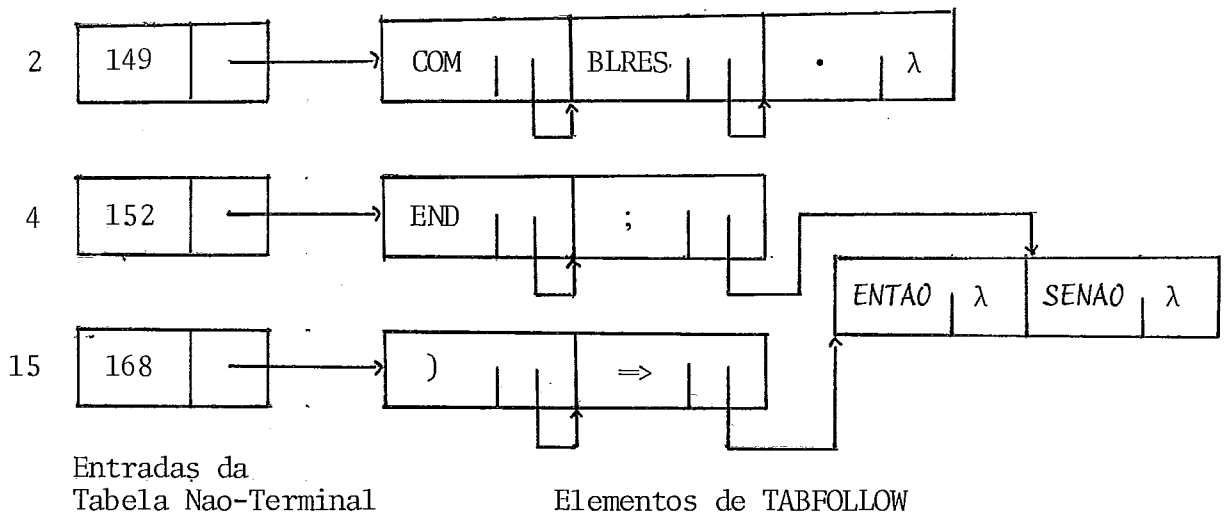
P_4 = {1004} , onde: 1004 corresponde ao ítem LR(0) (1004, *SENAO*) = 1005.

P_5 = {1005, 152}, onde: 1005 corresponde ao ítem LR(0) (1005, COM) = 1006.
152 é o estado auxiliar q(COM).

P_6 = {1006} , onde: 1006 corresponde ao estado de redução da macro.

Figura IV.4 - Listas de Estados Criadas Inicialmente.

2. ATUALIZAÇÃO DO FOLLOW



OBS.: Note que os novos terminais *ENTAO* e *SENAO* foram incluidos na lista dos símbolos FOLLOW's dos não-terminais EXPR(15) e COM(4) respectivamente; entretanto devido à organização lógica das estruturas envolvidas, os mesmos tornam-se automaticamente disponíveis em todas as listas nas quais os não-terminais envolvidos estiverem presentes (este é o caso do não-terminal BLOCO(2) que contém o não-terminal COM(4) na lista de seus FOLLOW's - o que significa que os FOLLOW's de COM são também FOLLOW's de BLOCO.

Figura IV.5 - Efeito da Atualização do Follow Quando da Introdução da Macro *SE*.

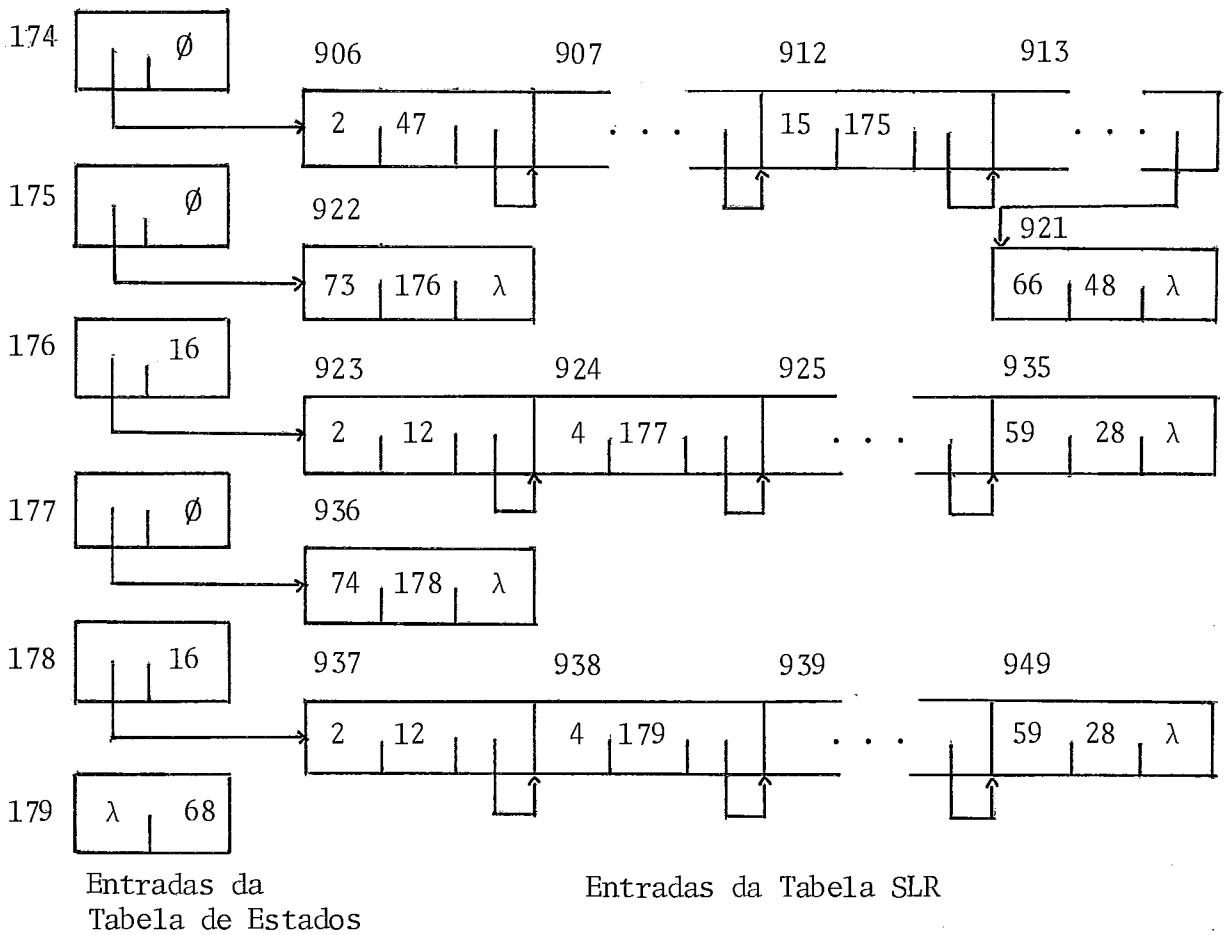
3. ESTADOS NOVOS CRIADOS

174 → P_1 = {1001, 168}
175 → P_2 = {1002}
176 → P_3 = {1003, 152}
177 → P_4 = {1004}
178 → P_5 = {1005, 152}
179 → P_6 = {1006}

OBS.: Neste caso, durante a análise das listas criadas inicialmente, nenhum novo estado foi incluído nas mesmas e nenhuma nova lista foi criada.

Figura IV.6 - Estados Novos.

4. COMPOSIÇÃO DOS NOVOS ESTADOS



OBS.: Note que 179 é o estado em que é efetuada a redução da macro e 68 é o número da produção correspondente a macro *SE* (COM → *SE* EXPR *ENTAO* COM *SENAO* COM). As reduções pela produção 16 (COM → ε) dos estados 176 e 178 provêm do estado 152 usado na formação dos mesmos. Na figura acima, por restrições físicas, em vez dos símbolos utilizou-se a representação interna dos mesmos (vide Tabela VI.1, no capítulo VI), este esquema será adotado também nas figuras subsequentes.

Figura IV.7 - Composição dos Estados Criados para a Macro *SE*.

5. TRANSIÇÕES CRIADAS COM O NOME DA MACRO

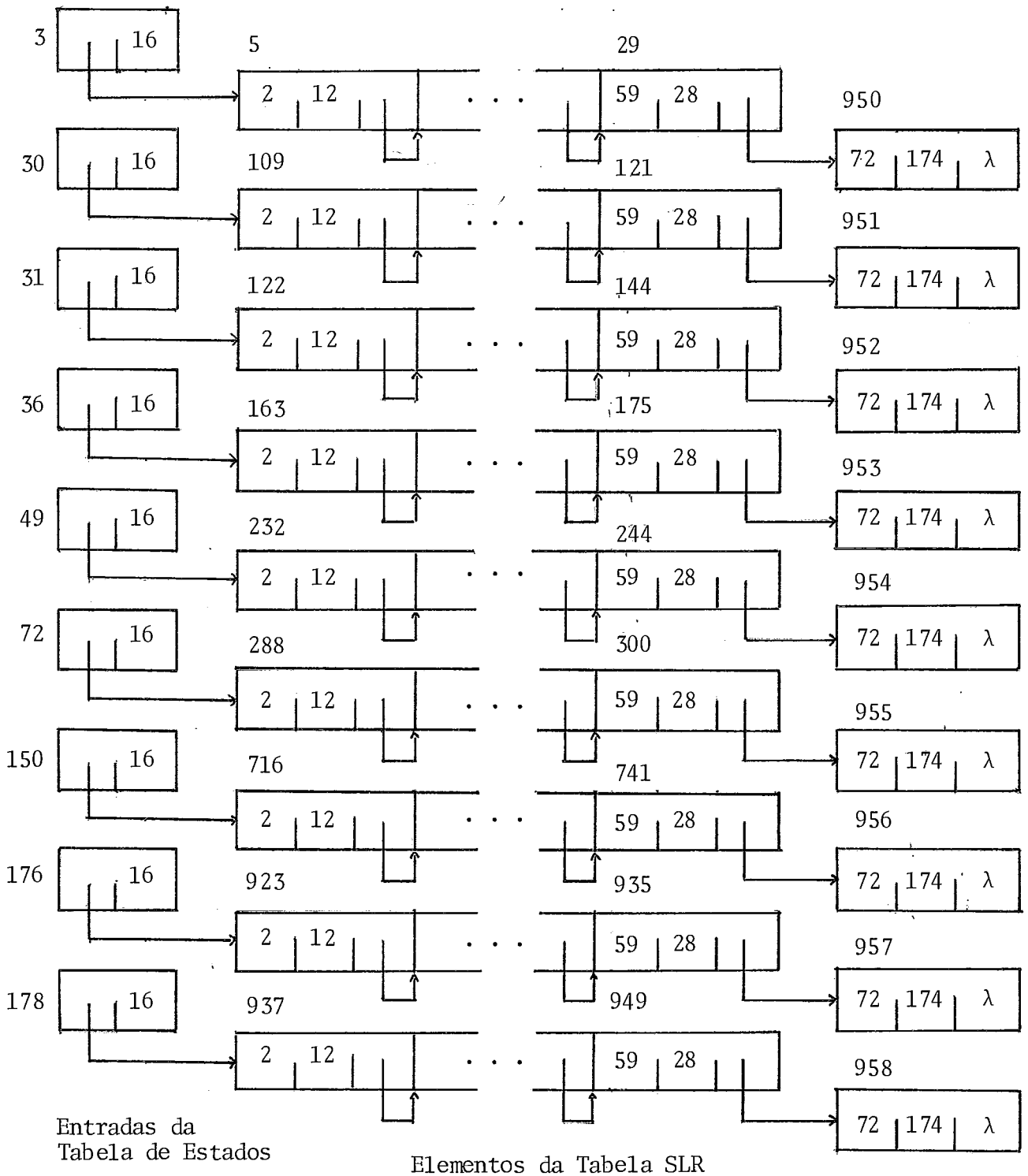


Figura IV.8 - Situação das Tabelas após a Criação das Transições com o Nome da Macro (SE(72)).

IV.4. UTILIZAÇÃO E EXPANSÃO DE MACROS

IV.4.1. UTILIZAÇÃO

A utilização de uma macro nada mais é que uma referência a uma macro, a qual é feita através da estrutura da macro sem a especificação da categoria sintática da mesma e com a substituição dos parâmetros formais por parâmetros atuais, devendo ocorrer num contexto de programa que aceite a categoria sintática especificada para a macro.

A análise sintática efetuada sobre a macro referenciada é normal, uma vez que o analisador foi estendido para suportar o efeito das macros incluídas, portanto, a macro no momento de sua utilização é uma construção sintática válida, analisável pelo analisador sintático. A única diferença existente entre a análise de uma construção da gramática base e de uma construção sintática criada através de uma macro é verificada no momento da redução, sendo que no primeiro caso será construída uma árvore sintática correspondente a construção reconhecida, e no segundo, ocorrerá o processo de macro expansão (descrito na seção seguinte).

Cabe ainda ressaltar que os parâmetros atuais usados na referência a uma macro, podem ser ou conter referências a esta ou a outras macros disponíveis; neste caso a análise também será normal, pois a medida que as novas referências são encontradas elas são analisadas e expandidas.Em todos os casos os parâmetros atuais deverão ocorrer na mesma ordem que os formais correspondentes na estrutura, além do que deverão ser construções cuja categoria sintática seja a mesma do parâmetro correspondente.

Quanto a presença de erros sintáticos nas referências a macros, eles podem estar tanto na especificação da estrutura utilizada como nos segmentos de programa usados como parâmetros atuais, e são tratados normalmente de acordo com o método de recuperação de erros utilizado nesta implementação.

No exemplo a seguir, é mostrada a análise sintática efetuada sobre uma referência a nossa macro exemplo, através de configurações do analisador.

CONFIGURAÇÃO	OBS
γ ; 30 se a > b entao maior := a senao maior := b ; δ	1
γ ; 30_{se} 174 a > b entao maior := a senao maior := b ; δ	
⋮ ⋮	
γ ; 30_{se} 174_{ARIT} $37_{>}$ 76_{ARIT} 105 entao maior := a senao maior := b ; δ	
γ ; 30_{se} 174 EXPR entao maior := a senao maior := b ; δ	
γ ; 30_{se} 174_{EXPR} 175 entao maior := a senao maior := b ; δ	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176 maior := a senao maior := b ; δ	
⋮ ⋮	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176_{VAR} $9_{:=}$ 32_{ARIT} 70 senao maior := b ; δ	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176 COM senao maior := b ; δ	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176_{COM} 177 senao maior := b ; δ	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176_{COM} 177_{senao} 178 maior := b ; δ	
⋮ ⋮	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176_{COM} 177_{senao} 178_{VAR} $9_{:=}$ 32_{ARIT} 70 ; δ	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176_{COM} 177_{senao} 178 COM ; δ	
γ ; 30_{se} 174_{EXPR} 175_{entao} 176_{COM} 177_{senao} 178_{COM} 179 ; δ	2

OBSERVAÇÕES

(1) Configuração do analisador no momento em que inicia-se a análise de uma referência a macro *SE*; onde γ e δ representam respectivamente parte do conteúdo da pilha sintática e parte da entrada a ser analisada, as quais não tem importancia neste exemplo.

(2) Configuração do analisador em que deverá ser efetuada a redução da macro (através da produção 68 : COM → *SE* EXPR *ENTAO* COM *SENAO* COM), ou equivalentemente sua expansão.

Figura IV.9 - Análise Sintática de uma Referência a Macro *SE* .

IV.4.2. EXPANSÃO

O processo de expansão de macros ocorrerá quando da realização de uma redução por uma produção correspondente a uma macro. Tal processo consiste em retirar a estrutura da macro da pilha sintática e colocar na entrada os elementos que compõem a definição da referida macro (os quais serão obtidos do vetor VETMACRO) com algumas ligeiras alterações. A alteração mais significativa efetuada sobre a definição da macro,diz respeito aos parâmetros; lembre-se que quando da análise da definição, ao ser encontrado um identificador de parâmetro, era guardado sua categoria sintática e sua ordem de ocorrência,neste caso a categoria do mesmo será o elemento a ser analisado enquanto que a ordem servirá para a obtenção do nodo raiz da sub-árvore sintática criada para os segmentos de programa correspondentes aos parâmetros reais (atuais) utilizados na presente referência a macro (este nodo será obtido a partir da pilha de nodos, paralela a pilha sintática).

Portanto, expandir uma macro significa efetuar a análise sintática de sua definição sem contudo repetir a análise dos parâmetros, os quais já foram considerados quando da análise da estrutura da macro.

O exemplo a seguir mostra os efeitos da macro expansão através de configurações do analisador sintático.

CONFIGURAÇÃO	OBS
γ ; 30_{se} 174_{EXPR} $175_{então}$ 176_{COM} $177_{senão}$ 178_{COM} 179 ; δ	1
γ ; 30 cond EXPR $\Rightarrow$ COM ; $\Rightarrow$ COM end ; δ	2

OBSERVAÇÕES

(1) Situação em que ocorrerá a redução pela produção 68, ocasionando a expansão da macro *SE*.

(2) Situação resultante da expansão da macro *SE*. Note na entrada a presença da definição da macro em questão, a qual será analisada normalmente; note também a presença dos não-terminais EXPR e COM (duas ocorrências), os quais correspondem aos parâmetros reais (atuais) utilizados na referência a macro *SE* cuja correspondência é feita com o auxílio da pilha semântica (mantida pelo analisador) e da ordem dos parâmetros (obtida a partir do vetor VETMACRO).

Figura IV.10 - Expansão da Macro *SE*

Ainda com respeito a expansão de macros, duas situações particulares podem ocorrer: a primeira ocorre quando a definição da macro que está sendo expandida contém a declaração de uma nova macro; neste caso, a análise da definição da macro em expansão segue normalmente até que a palavra MACRO seja encontrada, neste momento são iniciados os procedimentos de análise da declaração da macro encontrada, procedimentos estes que são os mesmos utilizados normalmente com a diferença de que os elementos que compõem a referida declaração, já estão codificados em tokens e encontram-se armazenados no vetor VETMACRO de onde serão lidos. Ao final da análise da referida macro, se ela estiver correta sob todos os aspectos considerados então a análise da definição (da macro em expansão) que estava sendo efetuada continua a partir do símbolo ENDMACRO, o qual é necessário para registrar a presença de uma declaração; senão a macro em questão, a análise iniciada e o bloco atual serão desconsiderados. A segunda situação ocorrerá quando uma referência a uma

macro for encontrada na definição da macro em questão; neste caso, a última macro referenciada será analisada e expandida antes que a análise da definição (da macro em expansão) que estava sendo efetuada, seja continuada.

Em todos os casos, normais ou particulares, os elementos da definição já estarão codificados em tokens e através da procedure MACROEXPANSOR são retirados do vetor VETMACRO e colocados em uma entrada especial, a qual será uma pilha, para possibilitar a expansão em profundidade (segunda situação, descrita acima); além disso a correspondência semântica entre parâmetros da estrutura e da definição será efetuada simultaneamente com a criação da pilha de entrada criada pela procedure MACRO-EXPANSOR.

IV.4.3. EXEMPLOS ESPECIAIS

Nesta seção serão exemplificadas as duas situações particulares de expansão, descritas na seção anterior, e também o caso em que uma referência a uma macro contém em seus parâmetros outras referências a macros.

EXEMPLO 1: Seja a seguinte declaração de MACRO:

```
MACRO DECL MATI IDEN I1 [ARIT A1, ARIT A2]
DEFINE DECLARE
          INTEGER I1#FISICO [(A1) * (A2)];
          MACRO  VAR I1 [ARIT A3, ARIT A4]
          DEFINE I1#FISICO [(A3-1) * (A2) + (A4)]
          ENDMACRO
       END
ENDMACRO
```

CONSIDERAÇÕES GERAIS SOBRE A ANÁLISE DA MACRO *MATI*:

- ANÁLISE DA ESTRUTURA - Normal.
- ANÁLISE DA DEFINIÇÃO - Normal até ser encontrada a palavra MACRO, a partir da qual todos os elementos até o delimitador ENDMACRO serão simplesmente armazenados em VETMACRO; sendo que a análise da definição, após o ENDMACRO da macro interna, continuará normalmente.
- EXTENSÃO DO ANALISADOR - O teste sintático da definição é feito de maneira que a macro declarada internamente seja transparente, exceção feita ao delimitador ENDMACRO necessário para representar uma declaração.
  A extensão propriamente dita será normal uma vez que a mesma é feita em função da estrutura da macro e não de sua definição.
- UTILIZAÇÃO E EXPANSÃO - É explicitada pela figura a seguir.

CONFIGURAÇÃO	OBS
γ 31 mati abc [10, 20] ; δ	1
⋮ ⋮	
γ 31_{mati} 174_{IDEN} 175 [176_{ARIT} 177 , 178_{ARIT} 179] 180 ; δ	2
γ 31 declare integer IDEN#fisico ... end ; δ	3
⋮ ⋮	
γ $31_{declare}$ 22_{LDECL} 65 ; macro ... endmacro end ; δ	
γ $31_{declare}$ 22_{LDECL} 65 ; 100 macro ... endmacro end ; δ	4
γ $31_{declare}$ 22_{LDECL} 65 ; 100 endmacro end ; δ	5
γ $31_{declare}$ 22_{LDECL} 65 ; $100_{endmacro}$ 25 end ; δ	
γ $31_{declare}$ 22_{LDECL} 65 ; 100 DECL end ; δ	
γ $31_{declare}$ 22_{LDECL} 65 ; 100_{DECL} 69 end ; δ	
γ $31_{declare}$ 22 LDECL end ; δ	
γ $31_{declare}$ 22_{LDECL} 65 end ; δ	
γ $31_{declare}$ 22_{LDECL} 65_{end} 101 ; δ	
γ 31 DECLARE ; δ	
γ 31_{DECL} 69 ; δ	6

Observações anotadas ao longo das configurações:

(1) Situação do analisador antes do início da análise de uma referência a macro *MAT1*; γ e δ representam respectivamente parte do conteúdo da pilha sintática e parte da entrada a ser analisada.

(2) Configuração do analisador ao término da análise de uma referência a macro *MAT1*; situação em que será efetuada a redução da macro em questão ou, equivalentemente, a macro-expansão.

(3) Configuração resultante da expansão da macro *MAT1*; note a definição da mesma na entrada.

(4) Configuração do analisador ao ser encontrada a declaração da macro, interna a macro *MAT1*; neste momento são ativadas as rotinas relacionadas ao tratamento de macros para que a macro interna (a qual efetivamente está sendo declarada neste momento) seja analisada. Note que os parâmetros I1 e A2 usados pela macro interna tem correspondência na macro externa, e que portanto não serão tratados como parâmetros e sim como "ABC" e "20" respectivamente (os quais são os

parâmetros atuais usados na referência a macro *MAT1* em questão); logo, o nome da nova macro será *ABC*.

(5) Configuração do analisador após a análise da macro *ABC*.

(6) Término da análise referente a declaração *MAT1* ABC [10,20]; cabe ressaltar aqui, que deste ponto em diante, qualquer referência a matriz *ABC*, no escopo da declaração *MAT1*, corresponderá a uma referência a uma macro.

EXEMPLO 2: Considere a macro ENQUANTO assim especificada:

```
MACRO COM ENQUANTO EXPR E1 FAÇA COM C1
DEFINE BEGIN
          NEW L1; LABEL L1;
          L1: COND E1 => BEGIN
                            C1;
                            GOTO L1
                         END;
                      =>
              END
      END
ENDMACRO
```

A seguir será especificada a macro *PARA*, cuja definição é construída em função da macro *ENQUANTO*:

```
MACRO COM PARA VAR V1 DE ARIT A1 PASSO ARIT A2 ATE ARIT A3 FAÇA COM C1
DEFINE BEGIN
        V1 := A1;
        ENQUANTO V1 <= A3
        FAÇA BEGIN
               C2;
               V1 := V1 + (A2)
             END
      END
ENDMACRO
```

CONSIDERAÇÕES GERAIS

Do ponto de vista de formação a macro *PARA* será analisada normalmente, entretanto quando estiver sendo efetuado o teste sintático de sua definição ao ser encontrada a referência a macro *ENQUANTO*, esta será analisada normalmente até o momento de sua redução quando, em vez de se realizar a macro-expansão, a estrutura da macro é retirada da pilha sintática e sua categoria é colocada na entrada, configurando assim um processo normal de redução (como se a macro *ENQUANTO* fosse uma construção da gramática base); em seguida o teste sintático da macro *PARA* é concluído e o analisador expandido para incluir o efeito da macro em questão. O efeito de uma referência a macro *PARA* é mostrado através das configurações da figura abaixo.

CONFIGURAÇÃO	OBS
γ 31 para I de 1 passo 1 ate 10 faça X[Y] := 0 ; δ	1
: :	
γ 31_{para} 178_{VAR} 179_{de} 180_{ARIT} 181_{passo} 182_{ARIT} 183_{ate} 184_{ARIT} $185_{\text{faça}}$ 186_{COM} 187 ; δ	2
γ 31 begin VAR := ARIT ; enquanto ... end end ; δ	3
: :	
γ 31_{begin} 3_{LISTA} $5_{;}$ 30 enquanto ... end end ; δ	4
: :	
γ 31_{begin} 3_{LISTA} $5_{;}$ 30_{enquanto} 174_{EXPR} $175_{\text{faça}}$ 176_{COM} 177 end ; δ	5
γ 31_{begin} 3_{LISTA} $5_{;}$ 30 begin new L1; ... end end ; δ	6
: :	
γ 31_{begin} 3_{LISTA} $5_{;}$ 30_{COM} 67 end ; δ	7
γ 31_{begin} 3 LISTA end ; δ	
γ 31_{begin} 3_{LISTA} 5 end ; δ	
γ 31_{begin} 3_{LISTA} 5_{end} 29 ; δ	
γ 31 BLOCO ; δ	
γ 31_{BLOCO} 12 ; δ	
γ 31 COM ; δ	
γ 31_{COM} 68 ; δ	8

Observações anotadas ao longo das configurações:

(1) Configuração do analisador antes do início da análise da referência a macro *PARA*.
(2) Configuração após a análise da referência a macro *PARA*; situação em que ocorrerá a macro expansão.
(3) Situação resultante da expansão da macro *PARA*. Note sua definição na entrada.
(4) Configuração do analisador no momento em que é encontrada a referência a macro *ENQUANTO* constante da definição da macro *PARA*.
(5) Configuração após a análise da referência a macro *ENQUANTO*.
(6) Situação resultante da expansão da macro *ENQUANTO*.
(7) Término da análise sintática da entrada resultante da expansão da macro *ENQUANTO*.
(8) Término da análise sintática da entrada resultante da expansão da macro *PARA*.

EXEMPLO 3: Considere a macro ENQUANTO utilizada no exemplo anterior e a macro SSE especificada a seguir:

```
MACRO SSE EXPR E2 ENTAO COM C2
DEFINE COND E2 => C2;
                =>
       END
ENDMACRO
```

Suponha o seguinte segmento de programa:

```
:
ENQUANTO I <= 10
FAÇA SSE X[I] > MAX
     ENTAO MAX := X[I];
:
```

o qual faz referência simultaneamente as macros consideradas. O efeito da análise sintática do segmento considerado é mostrado a seguir, através de configurações do analisador.

CONFIGURAÇÃO	OBS
γ 30 enquanto I <= 10 faça sse X[I] > max entao max := X[I] ; δ	1
γ $30_{enquanto}$ 174 I <= 10 faça sse X[I] > max entao max := [I] ; δ	
: :	
γ $30_{enquanto}$ 174_{EXPR} 175 faça sse X[I] > max entao max := X[I] ; δ	
γ $30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176 sse X[I] > max entao max := X[I] ; δ	2
γ $30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176_{sse} 178 X[I] > max entao max := X[I] ; δ	
: :	
γ $30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176_{sse} 178_{EXPR} 179 entao max := X[I] ; δ	
γ $30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176_{sse} 178_{EXPR} 179_{entao} 180 max := X[I] ; δ	
: :	
$\gamma 30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176_{sse} 178_{EXPR} 179_{entao} 180_{COM} 181 ; δ	3
$\gamma 30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176 cond EXPR => COM ; => end ; δ	4
: :	
$\gamma 30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176_{cond} 11_{LCOND} 34_{end} 71 ; δ	
$\gamma 30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176 COM ; δ	
$\gamma 30_{enquanto}$ 174_{EXPR} $175_{faça}$ 176_{COM} 177 ; δ	5
$\gamma 30$ begin new L1 ; ... end ; δ	6
: :	
$\gamma 30_{COM}$ 67 ; δ	7

Observações anotadas ao longo das configurações:

(1) Configuração do analisador no início da análise da referência a macro *ENQUANTO*.

(2) Configuração do analisador no início da análise da referência a macro *SSE*; observe que trata-se de um parâmetro da macro *ENQUANTO*.

(3) Configuração após a análise da referência a macro *SSE*; situação em que ocorrerá a macro-expansão.

(4) Situação resultante da expansão da macro *SSE*.

(5) Término da análise sintática da entrada resultante da expansão da macro *SSE*. Configuração resultante da análise da

referência a macro *ENQUANTO*; situação em que ocorrerá a macro expansão.

(6) Situação resultante da expansão da macro *ENQUANTO*.

(7) Término da análise sintática da entrada resultante da expansão da macro *ENQUANTO*.

IV.5. DIRETÓRIO DE MACROS

O diretório de macros tem por objetivo evitar que o usuário, a cada uso, necessite declarar explicitamente as macros por ele utilizadas; assim sendo, uma simples referência a um conjunto de macros feita através da declaração ARQMACRO, possibilitará ao analisador o acesso a declaração completa das referidas macros.

Esta opção além de tornar o programa fonte mais inteligível, evita que várias compilações sejam perdidas devido a presença de erros na formação da estrutura da macro e de erros sintáticos na definição, já que as macros do diretório estão corretas sob os aspectos de formação e de sintaxe.

As macros do diretório quando acrescentadas à gramática base, resultarão em uma gramática estendida analisável por este analisador, isto é, será uma gramática SLR(1); entretanto, esta característica não é garantida se as macros forem acrescentadas a gramática estendida, mesmo que a extensão seja resultado da introdução de macros deste diretório; em outras palavras, isto quer dizer que a introdução de macros separadamente resultará certamente em uma gramática estendida SLR(1), o que poderá não acontecer se as mesmas forem introduzidas simultaneamente. Além das macros disponíveis neste diretório, o usuário poderá, desde que autorizado para tal, incluir macros de seu interesse, sendo que será responsabilidade sua garantir que as macros incluídas apresentem as mesmas características das macros existentes, isto é, estejam corretas e sejam analisáveis; além disso, também será responsabilidade sua preservar a integridade do diretório como um todo.

A interface entre o analisador e o diretório é feita pela procedure PESQUISADIR, que tem ainda como função, verificar a sintaxe da declaração arqmacro e contornar os erros detectados, permitindo a continuação da pesquisa ao diretório sempre que possível.

O diretório de macros, do ponto de vista físico, é um arquivo sequencial formado por registros de 80 caracteres, dos quais os primeiros 72 compõem o campo útil, enquanto que os 8 últimos podem ser utilizados para especificação do endereço relativo do registro, dentro do arquivo.

A organização lógica do diretório encontra-se especificada no início do próprio diretório (cuja listagem encontra-se no apêndice 4), o qual pode ser esquematizado pela figura abaixo:

NÚMERO DO REGISTRO	1 CONTEÚDO 72	73 80	OBS
Ø	Z_1 Z_N		HEADER
1	*** ORGANIZAÇÃO DO DIRETÓRIO *** (TEXTO EXPLICATIVO)		
X_1	*** ESPECIFICAÇÃO DA MACRO 1 ***		CORPO DO DIRETÓRIO
Y_1			
X_2	*** ESPECIFICAÇÃO DA MACRO 2 ***		
Y_2			
⋮			
X_N	*** ESPECIFICAÇÃO DA MACRO N ***		
Y_N			
Z_1	X_1 Y_1 MACRO 1		REGISTROS DE REFERÊNCIA
Z_2	X_2 Y_2 MACRO 2		
⋮			
Z_N	X_N Y_N MACRO N		

Figura IV.11 - Estrutura do Diretório de Macros

PROCEDIMENTO PARA INCLUSÃO DE MACROS

1) Inclua o texto correspondente a declaração da macro após a última macro existente no corpo do diretório
2) Crie um registro de referência correspondente a macro incluída, após o último registro de referência existente.
3) Altere o registro "HEADER" para incluir o efeito da macro incluida e também para preservar a integridade do diretório.

PROCEDIMENTO PARA EXCLUSÃO DE MACROS

1) Exclua os registros correspondentes a declaração da macro em questão.
2) Exclua o registro de referência da referida macro.
3) Altere o "HEADER" de acordo com as modificações efetuadas no diretório.
4) Atualize todos os registros de referência seguintes ao excluído.

IV.6. TRATAMENTO DE ERROS

Como vimos nas seções anteriores deste capítulo, diversos tipos de erros podem ser detectados durante as várias etapas de análise que constituem o tratamento de macros.

Em todas as situações, exceto na utilização, o tratamento de erros será padrão, e será o seguinte:

- Emissão de uma mensagem notificando o usuário do erro ocorrido.
- Interrupção da análise que estava sendo efetuada no momento da detecção do erro.
- Ativação da procedure MACROERROR, cuja função é ignorar o restante da declaração da macro (se a mesma ainda não estava concluída) e do bloco que a contém (bem como dos blocos internos a ele). Isto justifica-se pelos seguintes fatos: a declaração de macros é opcional e a linguagem EXPAND II suporta declarações locais; estes fatos possibilitam ao usuário declarar uma macro somente quando precisar e no local onde precisar, por isso optou-se pela supressão da análise de determinados segmentos de programa que possivelmente contém referências a uma macro inexistente. Caso a macro errônea pertença a uma declaração global a compilação será, por motivos óbvios, encerrada neste ponto (no ponto de detecção do erro), senão a análise continuará a partir do elemento seguinte ao bloco ignorado.

```
EXEMPLO:  ----
          ----
          BEGIN
             ----
             "DECLARAÇÃO DA MACRO"     % PONTO DE DETECÇÃO DO
             ----                                    ERRO
             ----
             BEGIN
                ----
                ----
                ----
             END;
             ----
             ----
          END;                  % PONTO DE CONTINUIDADE DA ANÁLISE
          ----                    (A PARTIR DO ";")
          ----
```

Quanto aos erros detectados na utilização da macro, estes são considerados erros sintáticos normais, e por isso serão tratados da mesma forma que os erros sintáticos detectados na análise das construções da linguagem base (este tratamento está descrito na seção 3 do capítulo VI).

As mensagens de erro relacionadas ao tratamento de macros, encontram-se no apêndice 6.

IV.7. LIMITAÇÕES

Os fatores de limitação do método de expansão implementado são: do ponto de vista lógico, a necessidade de que a gramática estendida resultante da introdução de um conjunto de macros seja SLR(1) e que a definição das macros introduzidas sejam deriváveis na gramática estendida atual, isto é, na gramática estendida existente no momento em que as macros correspondentes foram introduzidas; enquanto que do ponto de vista físico, o limite da expansão, relacionado diretamente a extensão do analisador, é o tamanho das tabelas definido durante a implementação.

Na prática, o tamanho das tabelas não chega a constituir uma limitação definitiva, uma vez que os mesmos foram especificados dinamicamente, possibilitando sua redefinição sempre que se fizer necessário; para isto bastará ao usuário, autorizado para tal, aumentar o tamanho das tabelas o que pode ser feito simplesmente através da alteração do valor da variável correspondente a tabela que deve ser aumentada e consequentemente, da recompilação do fonte do compilador. As variáveis a serem alteradas, foram introduzidas no fonte através da cláusula DEFINE |BURROUGHS 4|, e encontram-se declaradas no início do mesmo (vide apêndice 3).

A tabela a seguir apresenta as limitações das várias tabelas do analisador usadas na extensão (especificadas na implementação), e as variáveis correspondentes a serem alteradas. Cabe ainda observar, que sempre que ocorra um estouro, será emitida uma mensagem ao usuário informando a tabela cujo limite foi ultrapassado e a variável a ser alterada.

LIMITAÇÕES	MÁXIMO	VARIÁVEL A SER ALTERADA
NÚMERO DE MACROS	200	NMAXMACRO
NÚMERO DE PARAMETROS	600	NMAXPAR
NÚMERO TOTAL DE ELEMENTOS DAS MACROS	4096	NMAXELEM
NÚMERO DE FOLLOW'S	256	NMAXFOL
NÚMERO DE ESTADOS	512	NMAXEST
NÚMERO DE AÇÕES SHIFT/GOTO	2048	NMAXSHIFT
NÚMERO DE NÍVEIS*	64	NMAXNIV
NÚMERO DE SÍMBOLOS TERMINAIS	256	NMAXCOL
NUMERO DE ESTADOS TERMPORÁRIOS	64	MAXTAMLISTA
NUMERO DE ELEMENTOS DA DEFINIÇÃO DE CADA MACRO**	200	MAXTOPO

* O número de níveis não diz respeito apenas ao tratamento de macros, mas sim ao processo de compilação como um todo.

** O limite imposto ao número de elementos da definição, está relacionado diretamente com o tamanho da pilha de entrada criada pelo macroexpansor quando da expansão de uma macro, e indiretamente com a pilha sintática usada para a análise não só de macros, mas da linguagem como um todo.

Os limites máximos especificados na tabela acima são resultantes do tamanho médio requerido para cada tabela verificado durante os testes e de uma tentativa de estabelecer um equilíbrio razoável entre o limite da extensão (considerando o aspecto dinâmico do analisador) e a quantidade de espaço (memória) requerida.

Outro ponto a observar é o alto grau de dependência existente entre as diversas limitações da extensão e as demais limitações do analisador; um caso típico dessa dependência é a limitação do número de macros, permitindo preliminarmente a existência de 200 macros em um mesmo programa, entretanto, este número depende também do total de identificadores utilizados (cuja limitação é de 351 identificadores, incluindo neste total as palavras reservadas da linguagem EXPAND II); assim sendo, por exemplo, se cada macro contiver 3 (três) novos identificadores

(um utilizado como nome, outro como delimitador e o outro como identificador de parâmetro) e se considerarmos que 51 identificadores foram usados para outros fins, o número máximo de macros suportadas (sem que haja estouro da tabela de símbolos) é 100.

Enfim, cabe salientar que apesar da possibilidade de redefinição do tamanho das tabelas (com relação ao número de elementos), o mesmo não pode ser feito (pelo menos de forma simples) com relação ao tamanho dos campos das tabelas; de maneira que as limitações físicas, na realidade, residem neste fato.

CAPÍTULO V

ESQUEMA DE TRADUÇÃO

O esquema de tradução implementado neste trabalho é composto por duas etapas: a geração de árvores que consiste na construção de árvores sintáticas correspondentes as construções sintáticas da linguagem EXPAND II e que é realizada em paralelo com o processo de análise sintática; e a geração de algol que consiste na criação de um programa na linguagem ALGOL (a partir da árvore sintática criada) equivalente ao programa EXPAND II de entrada e é realizada após o processo de análise sintática, se o programa em questão estiver correto sob todos os aspectos considerados.

V.1. GERAÇÃO DE ÁRVORES

Uma árvore sintática é uma estrutura cujo conteúdo representa o programa que a gerou. As árvores são construídas em função das construções sintáticas reconhecidas, isto é, cada vez que o analisador sintático efetua uma redução através de uma produção da gramática base é construída uma árvore equivalente a ela; e sempre que uma estrutura reconhecida envolve outras estruturas (ou construções), a árvore correspondente a primeira é construída em função das árvores das últimas, ou seja, as últimas passarão a ser sub-árvores da primeira. O resultado deste processo de geração, será uma única árvore sintática, a qual corresponderá exatamente ao programa envolvido.

A árvore sintática gerada é composta por nodos, os quais possuem a estrutura mostrada pela figura a seguir.

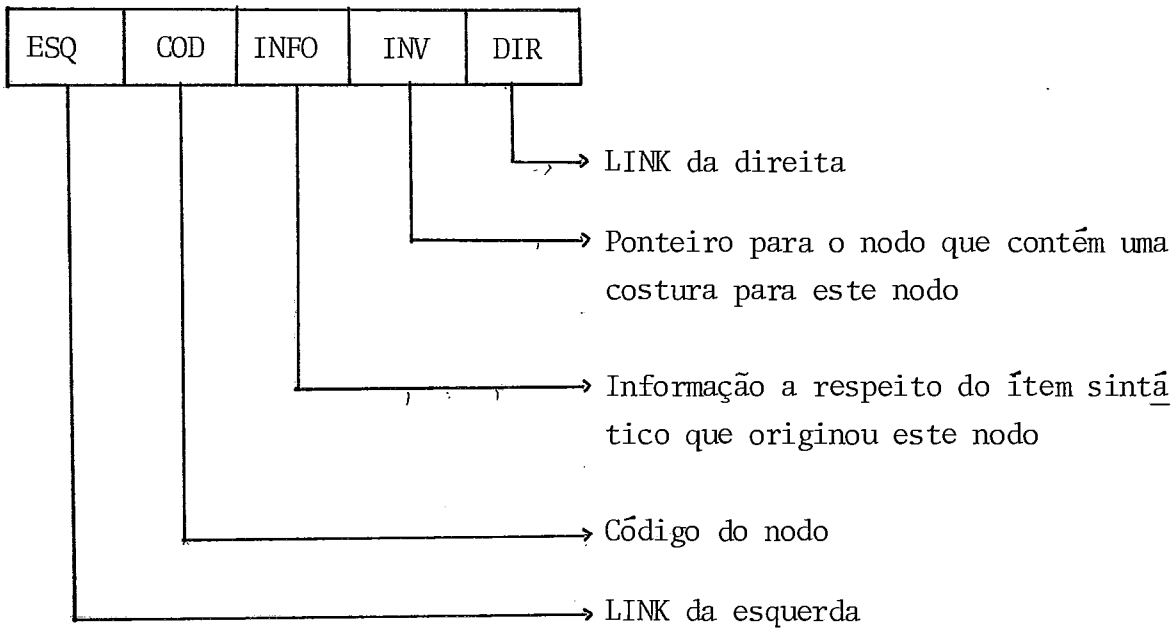


Figura V.1 - Estrutura de um Nodo

EXEMPLO: Considere a seguinte construção sintática:

B := A + C

Ao ser reconhecida a variável "B", é criado o nodo da figura V.2, o qual constitui a árvore da construção reconhecida (no caso uma variável simples).

Ø	ID	B	Ø	Ø

Figura V.2 - Nodo do Identificador "B".

Em seguida dois nodos semelhantes são criados para os operandos "A" e "C" definindo duas novas árvores. Quando a expressão aritmética "A + C" é reconhecida, é criada uma nova árvore em função das árvores construídas para os operandos "A" e "C", as quais passam a ser sub-árvores da árvore criada para a

expressão aritmética reconhecida, a qual é mostrada na figura abaixo.

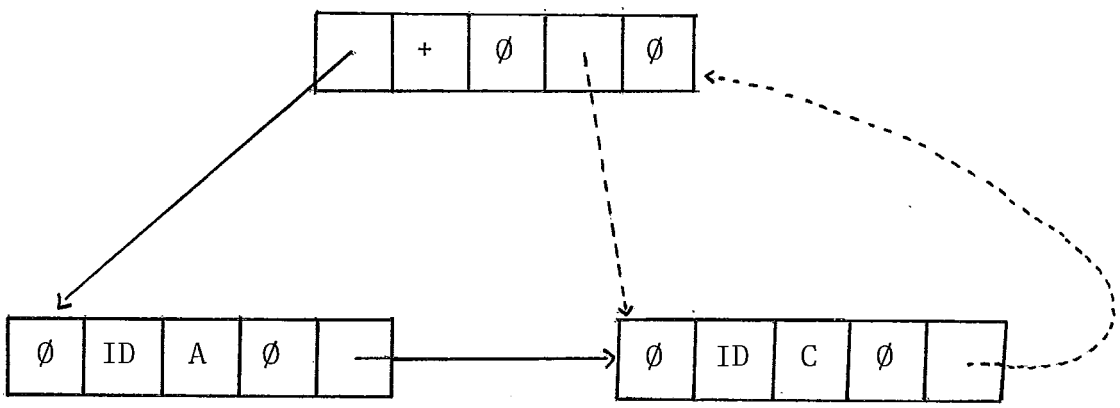


Figura V.3 - Árvore Correspondente a Expressão Aritmética A + C

O próximo estágio é a construção da árvore correspondente ao comando de atribuição "B := A + C", a qual utilizará as árvores da variável "B" e da expressão aritmética "A + C" já existentes, resultando na árvore da figura V.4.

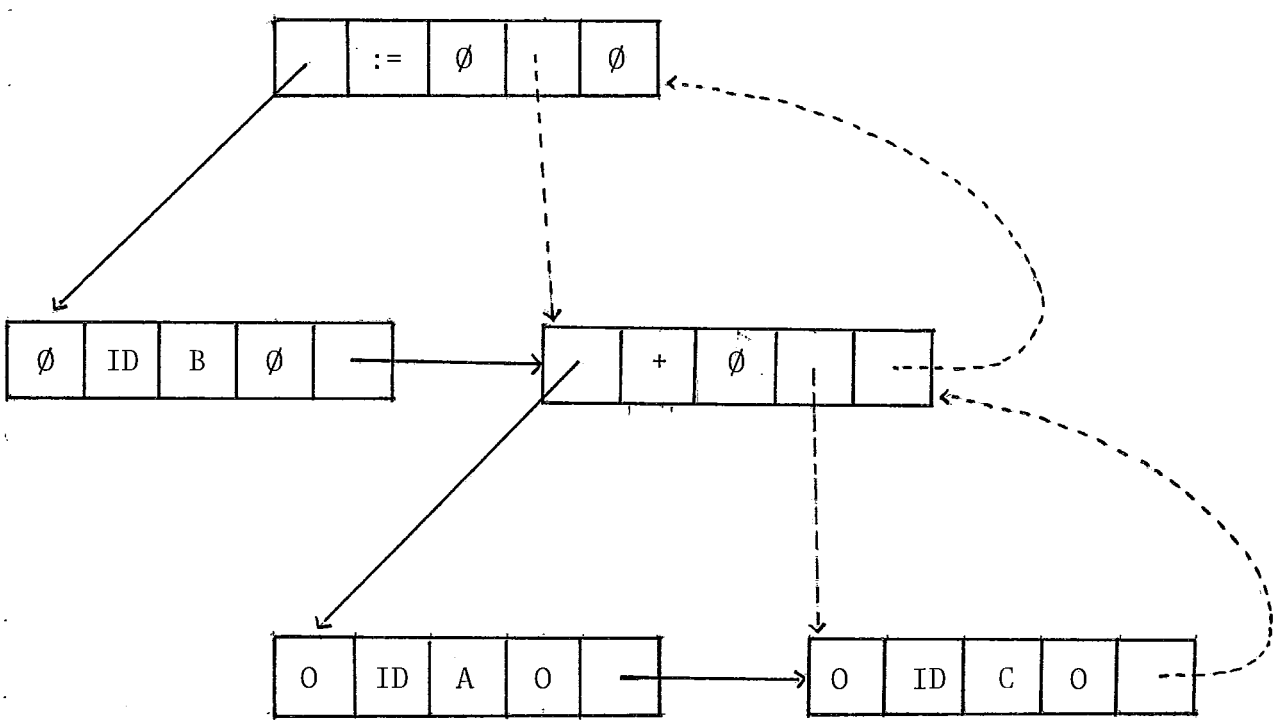


Figura V.4 - Árvore do Comando B := A + C

Desta forma são construídas as árvores correspondentes a todas as construções sintáticas existentes na linguagem, de maneira que a última árvore construída corresponderá a produção PROG → BLOCO '.', e será formada pelo nodo mostrado na figura V.5, o qual também será o nodo raiz da árvore correspondente ao programa que, neste momento, acaba de ser analisado.

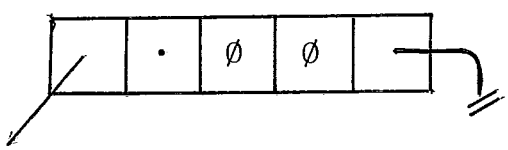


Figura V.5 - Nodo Raiz da Árvore Correspondente a um Programa.

O próximo passo na descrição dos procedimentos de geração de árvores seria a descrição dos procedimentos usados para a construção das árvores das diversas construções sintáticas da linguagem EXPAND II; entretanto, como isso seria massante e não muito proveitoso, daremos aqui apenas uma idéia geral destes procedimentos, através de construções sintáticas consideradas representativas. Assim sendo, descreveremos os procedimentos necessários para a construção de árvores correspondentes a listas, e em seguida faremos alguns comentários sobre árvores de expressões aritméticas, ressaltando o uso de bloco result como operando.

No que segue, representaremos os nodos pelo nome ou pelo símbolo associado aos mesmos, exceção feita aos nodos "ID" e "DIG", os quais serão representados respectivamente pelos identificadores e pelas constantes envolvidas; além disso, nas árvores apresentadas, o ponteiro referente ao campo INV não será representado (por tratar-se de um ponteiro auxiliar usado somente durante a construção das mesmas) e as sub-árvores, cuja especificação completa seja desnecessária, serão representadas pelo nome do nodo raiz escrito com letras minúsculas. O apêndice 7 contém a relação das construções sintáticas da gramática juntamente com o nome e o código dos nodos correspondentes.

LISTAS

Em EXPAND II várias construções sintáticas envolvem listas de diversos tipos, a saber: lista de declarações e comandos (LDC), lista de declarações (LD), lista de comandos (LC), lista de condições (LCOND), lista de variáveis (LVAR), lista de expressões de entrada e saída (LEES) e lista de identificadores (#); daí a escolha da descrição dos procedimentos de construção de árvores referentes a listas.

Suponha a seguinte lista de declarações e comandos:

$$d_1;\ d_2;\ \ldots;\ d_n;\ c_1;\ c_2;\ \ldots;\ c_m\ ,$$

onde d_1 a d_n representam declarações e c_1 a c_m representam comandos.

Após construídas as árvores correspondentes a d_1 e d_2, torna-se necessário a construção de uma nova árvore correspondente a uma lista de declarações (LD), a qual é mostrada na figura V.6 e no que segue será denominada "árvore LD"; em seguida, a medida que novas declarações são reconhecidas, as árvores

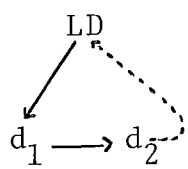


Figura V.6 - Árvore da LD Composta por "d_1" e "d_2"

correspondentes são anexadas a árvore LD existente, de forma que após o reconhecimento da n-ésima declaração, a árvore LD será:

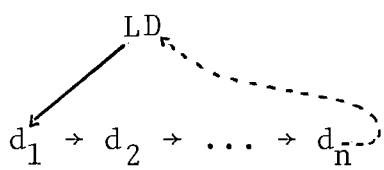


Figura V.7 - Árvore Completa da LD Considerada

O processo continua, e após o reconhecimento de c_1 e a construção de sua árvore, será criado o nodo LDC o qual será raiz da árvore LDC, construída em função das árvores LD e c_1 (já existentes), mostrada na figura V.8; a seguir, ao ser reconhecido o comando c_2, é construída uma árvore LC em função das

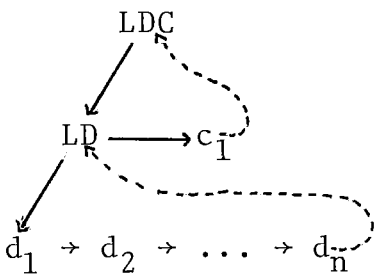


Figura V.8 - Árvore Parcial da LDC Considerada

árvores c_1 e c_2, a qual é anexada a árvore LDC existente. Finalmente, a medida que os comandos c_3, c_4 ... c_m são reconhecidos, é construída a árvore de cada um, as quais são anexadas à sub-árvore LC criada anteriormente (a qual está subordinada a árvore LDC em questão).

A árvore completa da lista de declarações e comandos considerada, é mostrada na figura abaixo.

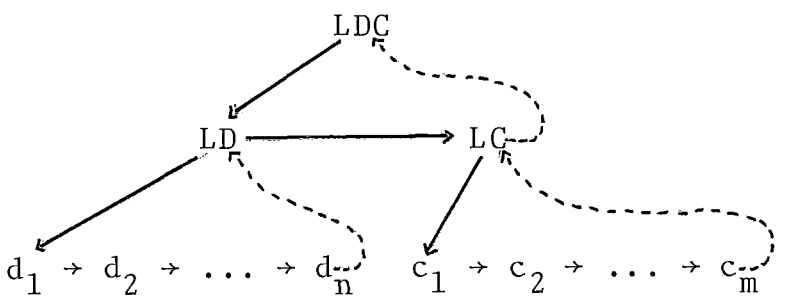


Figura V.9 - Árvore Completa da LDC Considerada

EXPRESSÕES ARITMÉTICAS

O processo de construção de árvores correspondente a expressões aritméticas, obedece exatamente a ordem em que as partes de uma expressão são reconhecidas; desta forma, é possível manter a prioridade inerente a cada operador bem como a estrutura da expressão especificada pelo programador. Assim, por exemplo, a expressão A * B + C dará origem a árvore da figura V.10a, enquanto que a expressão A * (B + C) originará a árvore mostrada na figura V.10b.

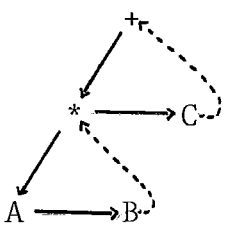


Figura V.10a - Árvore da Exp. A * B + C

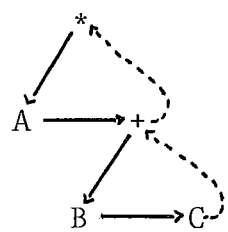
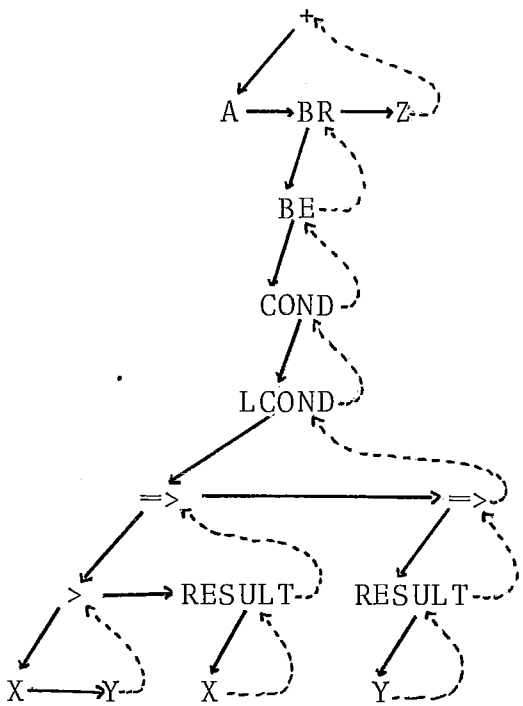


Figura V.10b - Árvore da Exp. A * (B + C)

Como sabemos, um bloco result pode ser utilizado como operando de uma expressão aritmética de um programa EXPAND II, entretanto como não existe operando equivalente na linguagem ALGOL (para a qual o programa EXPAND II será traduzido), são efetuados prodedimentos especiais para que o operando definido por um bloco result seja substituído antes da geração do programa ALGOL. Estes procedimentos (efetuados na presente etapa) dependem do contexto em que encontra-se a expressão aritmética que contém o bloco result e são descritos a seguir: inicialmente a árvore correspondente ao bloco result é construída normalmente, sendo que o nodo raiz da mesma será um dos operandos da expressão em questão.

EXEMPLO: A + BEGIN
COND X > Y => RESULT X; ==>
=> RESULT Y
END
END + Z

==>

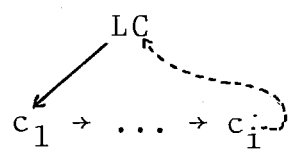


Entretanto, por ocasião do reconhecimento do comando que contém a expressão na qual encontra-se o bloco result, a sub-árvore correspondente ao bloco result é colocada antes do comando em questão na árvore LC que está sendo construída e em seu lugar, na lista de operandos da expressão, é criado um nodo TEMP, o qual estará relacionado com o bloco result em questão (através do campo info); esta situação é ilustrada no que segue:

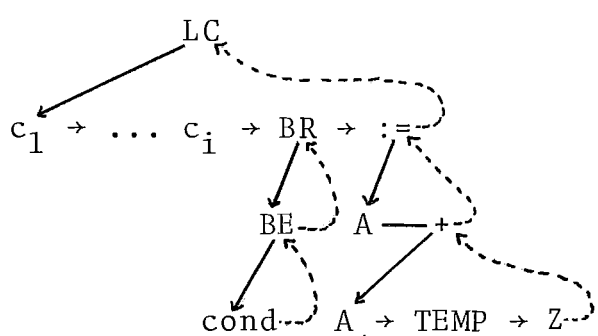
Seja o seguinte segmento de programa:

```
c1;
:
ci;
A := A + BEGIN
           COND X > Y => RESULT X;
                      => RESULT Y
           END
         END + Z
```

Antes do reconhecimento do comando que contém o bloco result teríamos a seguinte árvore:



E, após ter sido construída a árvore referente a expressão aritmética que contém o bloco result (especificada anteriormente), o comando de atribuição envolvido é reconhecido e a árvore resultante após o reconhecimento do mesmo é mostrada na figura abaixo:



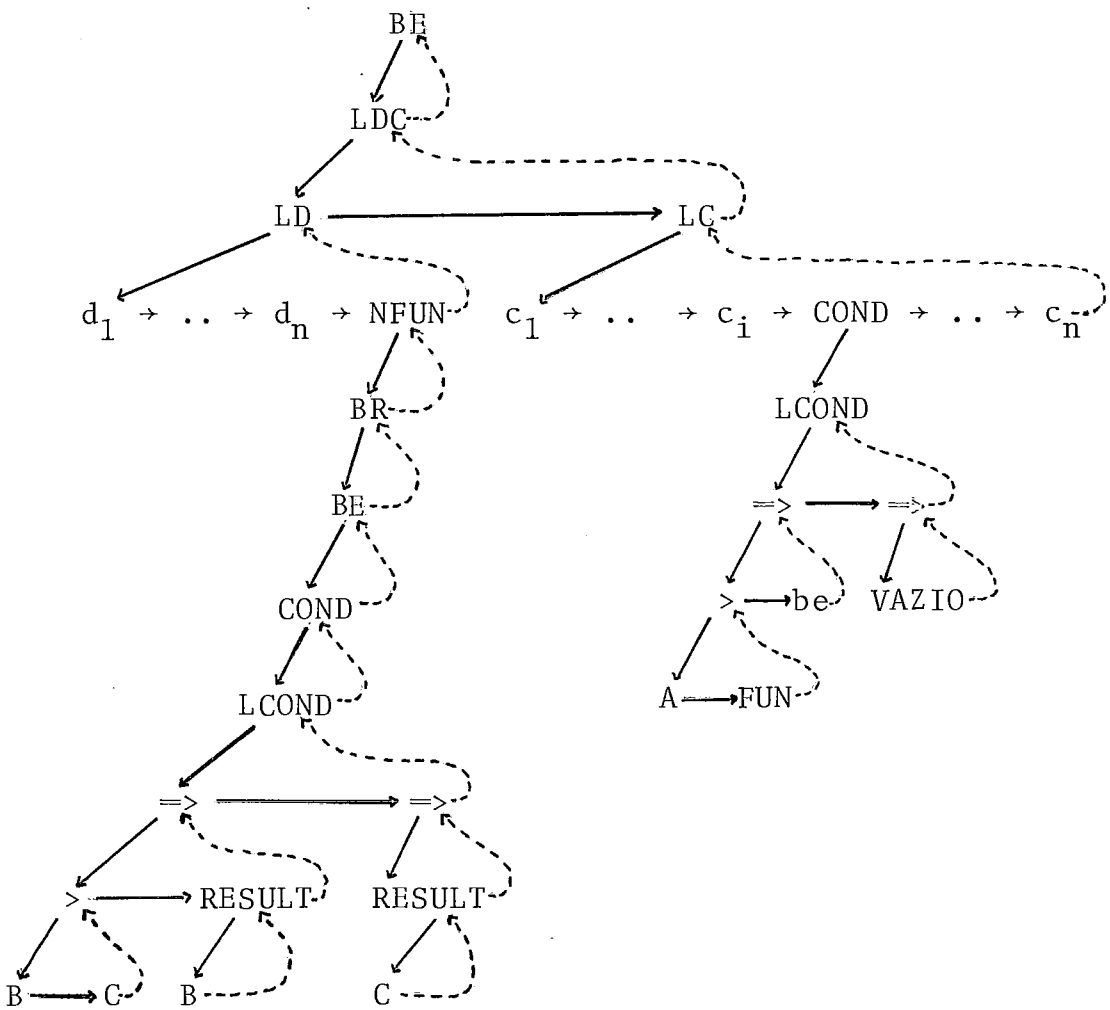
Por outro lado, se o bloco result estivesse envolvido em uma expressão relacional associada ao comando COND, para evitar que futuramente o mesmo seja avaliado desnecessariamente, em vez do procedimento acima descrito, é criado para ele uma função do tipo REAL (representada na árvore pelo nodo NFUN) a qual é incluída na árvore LDC (caso não exista é criada uma) exatamente após a última declaração existente, e em seu lugar

na expressão é criado um nodo FUN referente a chamada da função criada; esta situação é ilustrada no que segue:

Seja o seguinte segmento de programa:

```
BEGIN
  d1;
  :
  dn;
  c1;
  :
  ci;
  COND A > BEGIN
             COND B > C => RESULT B;
                        => RESULT C
             END
           END => BEGIN
                    :
                  END;
               =>
  END;
  :
  cn
END
```

a árvore correspondente é a mostrada na figura a seguir.



LIMITAÇÃO

A árvore sintática construída, não poderá possuir mais que 4096 nodos; caso este limite seja ultrapassado, será emitida uma mensagem notificando o usuário. Entretanto, da mesma forma que as limitações referentes ao tratamento de macros (seção 7, capítulo IV), o número máximo de nodos pode ser redefinido ; neste caso a variável a ser alterada (no fonte do compilador EXPAND II) é NMAXNODO.

V.2. GERAÇÃO DE ALGOL

O resultado do processo de tradução será um programa na linguagem ALGOL, o qual é gerado a partir da árvore sintática construída para o programa EXPAND II de entrada.

A linguagem ALGOL foi escolhida devido a semelhança estrutural das construções sintáticas (e consequentemente a uma semelhança semântica) entre EXPAND II e ela, permitindo desta forma uma tradução simples e em muitos casos direta.

O programa ALGOL gerado, excetuando-se alguns casos que serão vistos no final desta seção, será um programa sintáticamente correto, pois só será gerado quando o fonte EXPAND II também estiver correto; entretanto, nada pode ser afirmado sobre sua validade semântica uma vez que nenhuma análise é efetuada nesse sentido. Assim sendo, o programa gerado será um programa normal que servirá de entrada ao compilador ALGOL, o que equivale a dizer que o usuário além de conhecer a sintaxe da linguagem EXPAND II, deverá conhecer também a semântica da linguagem ALGOL |BURROUGHS 4| referente as construções sintáticas utilizadas.

O procedimento de geração de ALGOL é efetuado pela procedure GERAALGOL e consiste em percorrer a árvore sintática a partir da raiz, excecutando para cada nodo encontrado o procedimento associado ao seu código, até que o nodo raiz seja novamente encontrado, estabelecendo o final da geração e consequentemente o fim do processo de tradução.

O percurso na árvore sintática, não está compromissado com nenhum algoritmo padrão, sendo o mesmo definido em função de cada nodo encontrado; todavia, a idéia fundamental do referido percurso é mostrada através do algoritmo abaixo.

```
PROCEDURE GERAALGOL (RAIZ);
BEGIN
  DESCENDO := TRUE;
  ROTINA (COD[RAIZ],  DESCENDO);
  PROX := ESQ[RAIZ];
  WHILE PROX NEQ RAIZ
  DO BEGIN
       ROTINA (COD[PROX], DESCENDO);
       IF DESCENDO
       THEN IF ESQ[PROX] NEQ "λ"
            THEN PROX := ESQ[PROX]
            ELSE IF DIR[PROX] GT Ø
                 THEN PROX := DIR[PROX]
                 ELSE BEGIN
                        DESCENDO := FALSE;
                        PROX  := -DIR[PROX]
                      END
       ELSE IF DIR[PROX] GT Ø
            THEN BEGIN
                   DESCENDO := TRUE;
                   PROX := DIR[PROX]
                 END
            ELSE PROX := -DIR[PROX];
     END;
END OF PROCEDURE
```

Algoritmo V.1 - Idéia Geral do Percurso na Árvore

Como foi dito anteriormente, o procedimento a ser efetuado em cada caso depende do código do nodo encontrado, mas além disso depende também da direção da pesquisa sobre a árvore no momento em que o referido nodo é encontrado; esta direção poderá ser: para baixo (DESCENDO), para cima (SUBINDO) e para a direita (DESCENDO).

No que segue, não nos deteremos nos procedimentos propriamente ditos mas sim no resultado obtido quando da execução dos mesmos; assim sendo, descreveremos para cada ítem e para

cada construção sintática da linguagem EXPAND II, o resultado da tradução das mesmas para a linguagem ALGOL, juntamente com comentários, exemplos e esclarecimentos que se fizerem necessários em cada caso. Os exemplos utilizados envolverão a construção usada no programa EXPAND II, a árvore sintática correspondente e o resultado de sua tradução para o ALGOL; sendo que as árvores estarão de acordo com as convenções especificadas na seção anterior deste capítulo.

IDENTIFICADORES

Os identificadores simples criados pelo programador são simplesmente copiados para o programa gerado; esta operação é efetuada com o auxílio do campo INFO do nodo ID (cujo conteúdo é um ponteiro para a tabela de símbolos), o qual é usado para acessar a representação física do identificador.

Além dos identificadores criados pelo programador, o programa gerado poderá conter identificadores criados pelo próprio analisador, estes casos serão tratados na descrição da tradução das construções sintáticas nas quais os mesmos são criados.

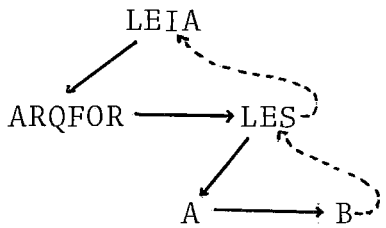
CONSTANTES

São copiadas literalmente da tabela de números, com o auxílio do campo INFO do nodo DIG.

SÍMBOLOS ESPECIAIS E PALAVRAS RESERVADAS

Os símbolos especiais e as palavras reservadas que constam do programa gerado, são criados explicitamente nos procedimentos associados aos nodos correspondentes às construções que os definem.

EXEMPLO: Considere a seguinte árvore:

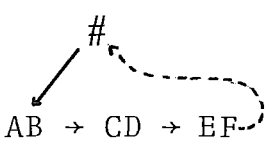


Neste caso, ao ser encontrado o nodo LEIA (descendo), é gerada a cadeia "READ(" e o caminhamento continua pelo LINK da esquerda; ao ser encontrado novamente o nodo LEIA (agora subindo) é gerado ")" ou ");" dependendo do próximo nodo. Desta forma são efetuados a maioria dos procedimentos associados a nodos que envolvam palavras reservadas e ou símbolos especiais.

LISTA DE IDENTIFICADORES (IDENTIFICADORES CONCATENADOS)

A tradução, neste caso, consiste em eliminar os símbolos de concatenação (#) utilizados, justapondo os identificadores simples envolvidos (os quais individualmente passam pelo processo normal de tradução descrito anteriormente).

EXEMPLO:

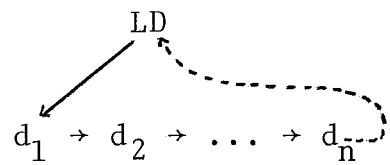
AB#CD#EF ==>  ==> ABCDEF

Cabe ressaltar aqui que o tamanho do identificador resultante, não poderá ultrapassar 64 caracteres; caso isto aconteça, serão considerados apenas os primeiros identificadores da lista, cuja concatenação (justaposição) não ultrapasse o limite estabelecido.

LISTAS

O processo de tradução efetuado sobre as diversas listas existentes na linguagem EXPAND II é bastante simples, e consiste na tradução de cada elemento que a compõem e na criação do símbolo separador a ela associado, o qual é inserido entre os elementos dela (exceção feita a lista de identificadores, descrita acima).

EXEMPLO:

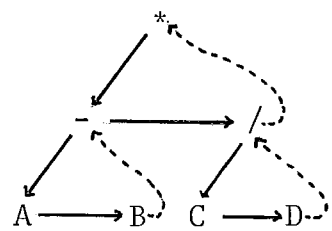


Neste caso, após ter sido traduzida a declaração d_1, é gerado um ";" que é o símbolo usado como separador de declarações; em seguida é traduzida a declaração d_2 e gerado um novo ";" e assim sucessivamente até o final da lista.

EXPRESSÕES ARITMÉTICAS

A tradução de expressões aritméticas segue o mesmo princípio da tradução de listas, entretanto é mais complexa uma vez que envolve operadores (e não separadores) os quais possuem uma prioridade associada; na tradução é utilizada uma pilha de operadores cujo elemento do topo define o operador a ser gerado após a tradução de cada operando. Quanto a geração de sub-expressões entre parênteses, a estrutura da árvore correspondente juntamente com a prioridade associada aos operadores envolvidos, definem a necessidade bem como o ponto em que os parênteses deverão ser introduzidos.

EXEMPLO:

(A - B) * (C / D) ==>  ==> (A - B) * C / D

BLOCO

Um bloco utilizado no programa EXPAND II, resultará também em um bloco no programa gerado; sendo que a tradução do mesmo, consiste na geração da palavra "BEGIN", na tradução das declarações e dos comandos nele contidos e na geração da palavra "END".

É necessário salientar que o número de blocos do programa gerado pode ser maior que o número de blocos existentes no programa EXPAND II; isto deve-se ao fato de que em algumas situações são criadas novas construções (pelo analisador) exigindo, por imposição sintática, que novos blocos (ou comandos compostos) sejam introduzidos. Esta situação será exemplificada no decorrer desta seção.

BLOCO RESULT

Um bloco result pode ser usado em duas situações distintas: como corpo de um procedimento com tipo ou como operando de uma expressão aritmética (note que apesar dessa situação não mais existir fisicamente ela continua existindo do ponto de vista lógico, uma vez que existe uma ligação entre o bloco result e o nodo TEMP correspondente). Em ambos os casos, inicialmente o processo de tradução é o mesmo, e consiste na tradução do bloco subordinado a ele e na criação de um comando de atribuição zerando a variável associada (a qual será o nome da função ou a temporária criada pelo analisador); comando este que será introduzido no programa gerado antes do primeiro comando

executável do bloco em questão.

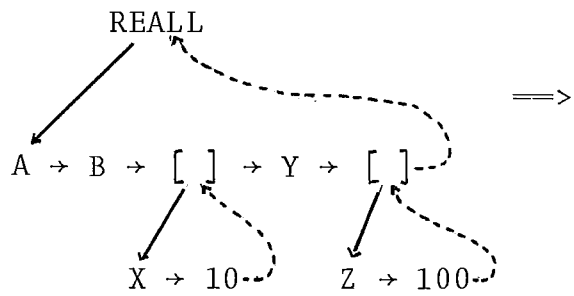
No primeiro caso, se o procedimento foi criado pelo analisador (durante a etapa de geração de árvores), será criado um nome para o referido procedimento, o qual será um identificador da forma "FUNXYZ N" onde N é uma constante composta de 4 dígitos geradas consecutivamente a partir de 1001; nome este que será utlizado nos comandos result internos ao bloco result em questão e em substituição ao nodo FUN correspondente (caracterizando uma chamada de função).

No segundo caso, é criada uma temporária associada ao bloco result, a qual será utilizada nos comandos result internos ao bloco em questão, e em substituição ao nodo TEMP correspondente ao referido bloco. As temporárias criadas para este fim, serão variáveis do tipo REAL e da forma "XYZTEMPØØ|I|",onde o subscrito I define o elemento do array de temporárias a ser utilizado em cada caso; quando existir, o array de temporárias será a primeira declaração do programa algol gerado,e sua dimensão dependerá da quantidade de blocos result usados no comando que contiver o maior número de blocos result, e não do número total de blocos result utilizados como operandos; isto equivale a dizer que blocos result associados a comandos distintos, poderão ter as mesmas temporárias associadas. O controle dos comandos de atribuição criados e das temporárias (tanto a criação como a utilização) é feito com o auxílio de uma pilha criada durante o processo de geração, a qual contém informações acerca dos blocos result envolvidos. No decorrer desta seção serão apresentados exemplos envolvendo estas situações.

DECLARAÇÃO INTEGER E REAL

Para cada declaração INTEGER ou REAL do programa EXPAND II, será gerada uma ou mais declarações (INTEGER ou REAL, dependendo do caso) no programa ALGOL gerado; sendo que será gerado mais de uma, se dentre as variáveis declaradas existir pelo menos uma que não seja uma variável simples (isto é, seja um array), neste caso para cada variável array declarada, será criada uma declaração de array no programa gerado.

EXEMPLO REAL A, B, X[10], Y, Z[100]; ==>



==>

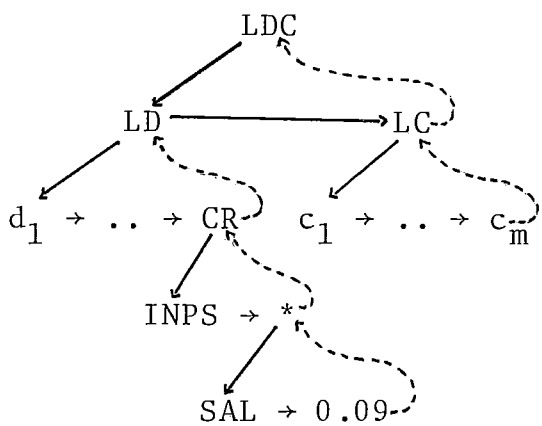
```
REAL A, B;
REAL ARRAY X[1 : 10];
REAL Y;
REAL ARRAY Z[1 : 100];
```

DECLARAÇÃO DE CONSTANTES

Esta declaração é traduzida como sendo uma declaração de variável INTEGER ou REAL (dependendo do tipo da constante) e um comando de atribuição, no qual a variável a esquerda do sinal de atribuição e a expressão aritmética à direita do mesmo são aquelas usadas na declaração da constante; o referido comando será gerado antes do primeiro comando executável do bloco em que se encontra a declaração de constante.

EXEMPLO:

d_1
⋮
REAL INPS : SAL * 0,09 ; ⟹
c_1
⋮
c_m



⟹

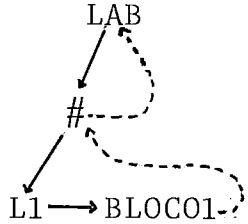
d_1
⋮
REAL INPS;
INPS := SAL * 0.09;
c_1
⋮
c_m

DECLARAÇÃO LABEL

Consiste na geração da palavra LABEL e na tradução do identificador associado.

EXEMPLO:

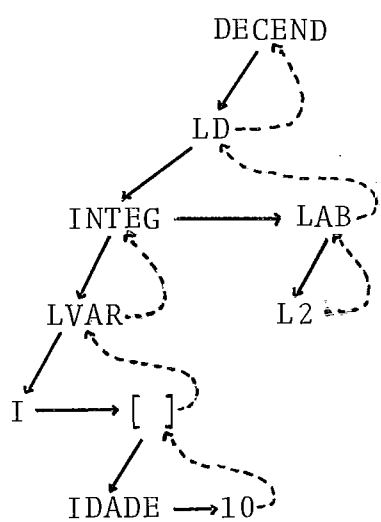
LABEL L1#BLOCO1 ==> LAB ==> LABEL L1BLOCO1



DECLARAÇÃO DECLARE

Apenas as declarações que a compõem são traduzidas.

EXEMPLO:
```
DECLARE
   INTEGER I,  IDADE[10];
   LABEL L2
END
```
==>

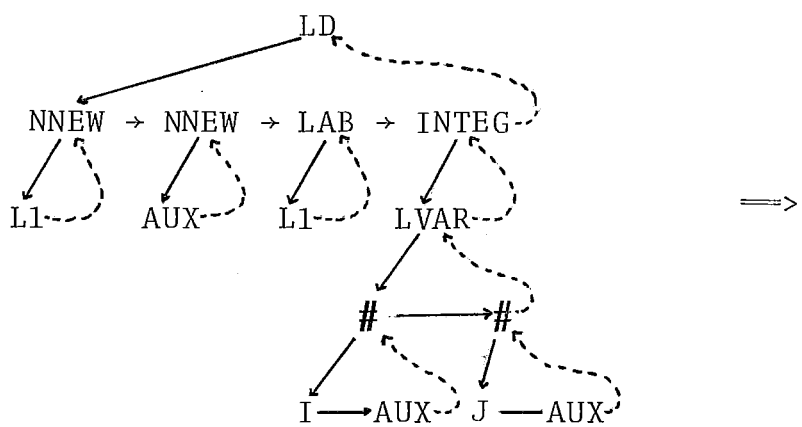



```
INTEGER I;
INTEGER ARRAY IDADE[1 : 10];
LABEL L2;
```

DECLARAÇÃO NEW

Não é traduzida para o algol; sua função é fazer com que o analisador crie um novo identificador da forma "XYZ N", onde N é uma constante formada por 5 dígitos, criada sequencialmente a partir de 00001, o qual substituirá todas as referências ao identificador declarado como NEW, no escopo da referida declaração.

EXEMPLO: NEW L1
NEW AUX; ==>
LABEL L1;
INTEGER I#AUX, J#AUX;



==>

LABEL XYZ00001;
INTEGER IXYZ00002, JXYZ00002;

DECLARAÇÃO MACRO E ARQMACRO

Não são traduzidas para o algol, uma vez que são utilizadas apenas no tratamento de macros.

DECLARAÇÃO FILE

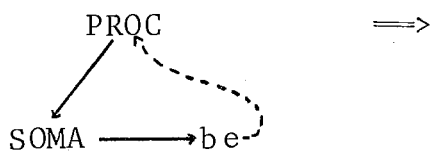
Sua tradução consiste na geração da palavra "FILE" e na cópia da especificação dos arquivos que estão sendo declarados; especificação esta que encontra-se na tabela de arquivo e formato, a qual é acessada através do conteúdo do campo INFO do nodo NFILE.

DECLARAÇÃO DE PROCEDIMENTOS

A tradução de um procedimento consiste na tradução do cabeçalho e do corpo do referido procedimento; sendo que a tradução do cabeçalho compreende a geração da(s) palavra(s) reservada(s) envolvida(s) e a tradução do identificador do procedimento em questão, enquanto que o corpo é traduzido de acordo com os procedimentos de tradução de BLOCOS ou de BLOCOS RESULT, dependendo se o procedimento é uma ROTINA ou uma FUNÇÃO.

EXEMPLOS:

1) PROCEDURE SOMA
   BEGIN
      ⋮
   END




```
PROCEDURE SOMA;
BEGIN
   ⋮
END
```

2) REAL PROCEDURE FUN1; ==>

BEGIN

d_1;

⋮

d_n;

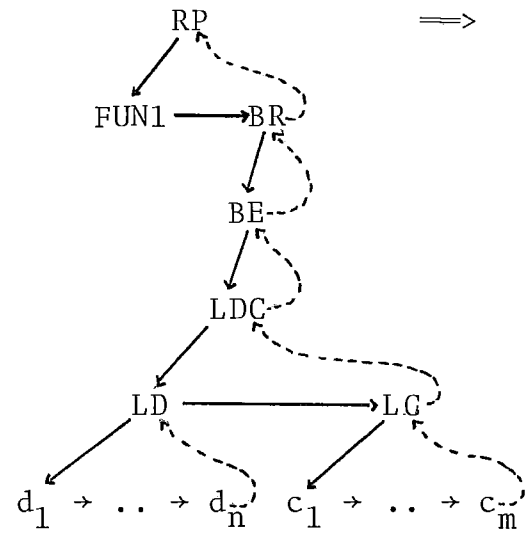
c_1;

⋮

c_m

END

==>



REAL PROCEDURE FUN1;

BEGIN

d_1;

⋮

d_n;

FUN1 := Ø;

c_1;

⋮

c_m

END

COMANDO DE ATRIBUIÇÃO

Sua tradução consiste na tradução da variável e da expressão aritmética que o compoẽm, bem como na geração do símbolo de atribuição (":=") entre a variável e a expressão traduzidas.

COMANDO RESULT

É traduzido para um comando de atribuição, no qual: a variável será a temporária criada para o bloco result que contém o comando result em questão ou o nome do procedimento ao

qual o bloco result está subordinado, dependendo do contexto em que o bloco result se encontra; e a expressão aritmética será aquela associada ao comando RESULT.

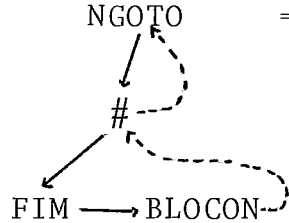
Por outro lado se o comando RESULT não pertencer a nenhum bloco result, apesar desta situação ser de erro, mesmo assim o comando de atribuição correspondente será gerado, só que a variável associada será o identificador "XYZERROTEMP" (criado pelo analisador), a qual por não ter sido declarada, ocasionará um erro no programa ALGOL gerado.

COMANDO GOTO

Consiste na geração da cadeia "GO TO" e na tradução do identificador associado, a qual será feita de acordo com os procedimentos normais de tradução de identificadores.

EXEMPLO:

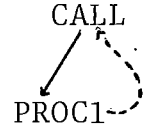
GOTO FIM#BLOCON ==> NGOTO ==> GO TO FIMBLOCON



COMANDO CALL

Neste comando, apenas o identificador associado será traduzido.

EXEMPLO: CALL PROC1 ==> CALL ==> PROC1



COMANDO VAZIO

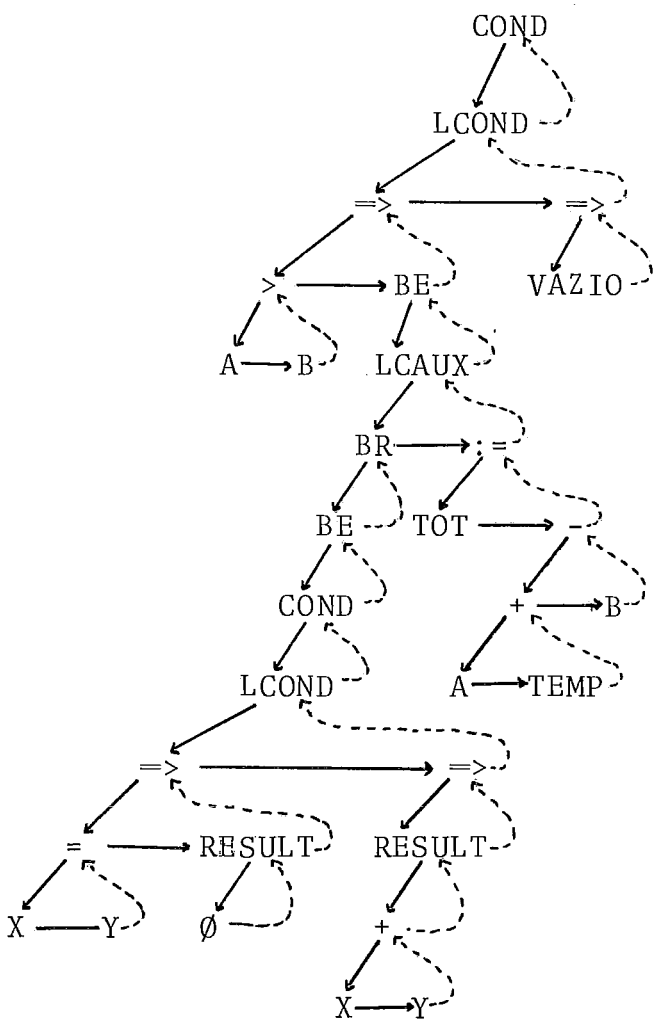
Todo comando vazio utilizado no programa EXPAND II será mantido no programa ALGOL gerado; entretanto nenhum procedimento de tradução é efetuado.

COMANDO COND

É traduzido para um comando IF-THEN-ELSE ou para um conjunto (aninhado) deles; dependendo se o comando COND em questão, contém uma ou mais condições.

EXEMPLOS:

```
1)   COND A > B => TOT := A + BEGIN
                                 COND X = Y => RESULT Ø;
                                            => RESULT X + Y
                                 END
                               END - B;
                 =>
     END
```




```
IF A > B
THEN BEGIN
       BEGIN
         XYZTEMPØØ[1] := Ø.,
         IF X = Y
         THEN XYZTEMPØØ[1] := Ø
         ELSE XYZTEMPØØ[1] := X + Y;
       END;
       TOT := A + XYZTEMPØØ [1] - B;
     END
ELSE ;
```

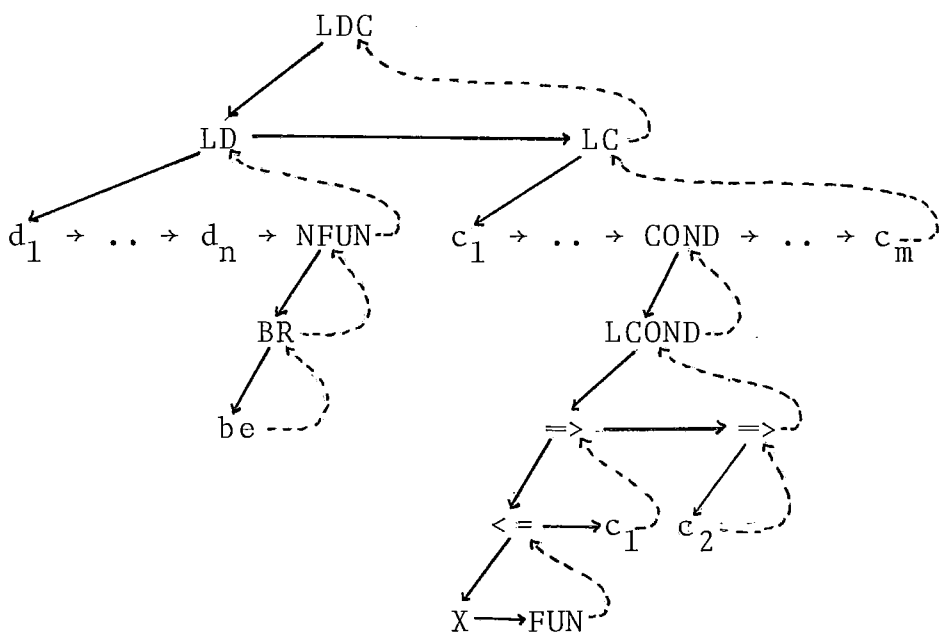
OBSERVAÇÃO: Note que o bloco associado a condição A > B foi criado por haver mais de um comando subordinado a esta condição.

2) d_1;
:
d_n;
c_1;
:

```
COND IND <= BEGIN
                COND TAB = 1 => RESULT  500;
                     TAB = 2 => RESULT 1000;
                             => RESULT 2000;
                END
            END => c1;
                => c2;
END;
```

:
c_m

==>



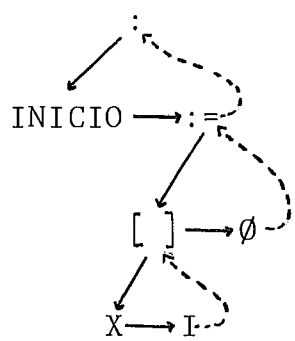
```
==>     d1;
        :
        dn;
        REAL PROCEDURE FUNXYZ1001;
        BEGIN
          FUNXYZ1001 := 0.;
          IF TAB = 1
          THEN FUNXYZ1001 := 500
          ELSE IF TAB = 2
               THEN FUNXYZ1001 := 1000
               ELSE FUNXYZ1001 := 2000
        END;
        c1;
        :
        IF X <= FUNXYZ1001
        THEN c1
        ELSE c2;
        :
        cm
```

COMANDO ROTULADO

Consiste na tradução do identificador usado como rótulo, na geração do símbolo ":" e na tradução do comando associado (a qual, obviamente, depende do comando em questão).

EXEMPLO:

INICIO : X[I] := Ø ==>



==>

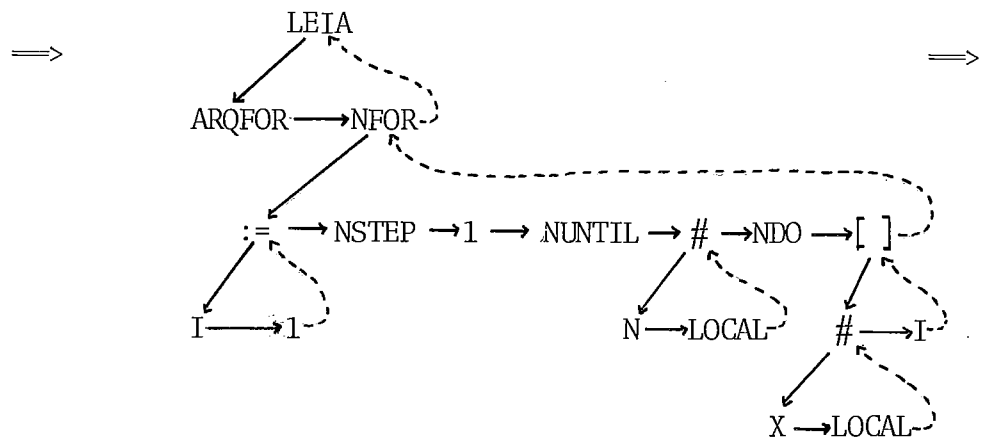
INICIO : X[I] := Ø

COMANDOS DE ENTRADA E SAÍDA (READ E WRITE)

A tradução consiste na geração de comandos de E/S equivalentes na linguagem ALGOL. A especificação de arquivo e formato dos mesmos, é copiada literalmente da tabela de arquivo e formato, cujo acesso é feito através do campo INFO do nodo do ARQFOR subordinado ao nodo que representa o comando (READ ou WRITE) em questão. Quanto a tradução da lista de expressões de entrada e saída (LEES), ela é efetuada de acordo com cada caso, ou seja, as expressões aritméticas, os identificadores e as constantes são traduzidas de acordo com os procedimentos apropriados (vistos anteriormente), enquanto que a tradução da cláusula FOR consiste na tradução das construções sintáticas envolvidas e na geração das palavras reservadas que a compõem, segundo a ordem pré-estabelecida.

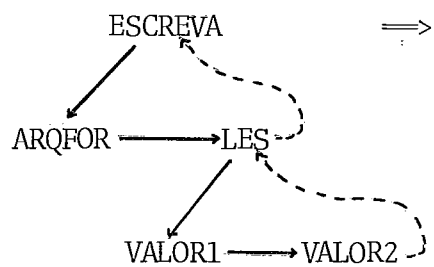
EXEMPLOS

1) READ (ENT, <10I5>, FOR I := 1 STEP 1 UNTIL N#LOCAL DO X#LOCAL[I]);



READ (ENT, <10I5>, FOR I := 1 STEP 1 UNTIL NLOCAL DO XLOCAL[I]);

2) WRITE (SAI, /, VALOR1, VALOR2);



WRITE (SAI, /, VALOR1, VALOR2);

CASOS EM QUE O PROGRAMA ALGOL GERADO PODERÁ CONTER ERROS SINTÁTICOS.

Sempre que o programa EXPAND II contiver declaração FILE, comando READ ou comando WRITE, nada poderá ser afirmado sobre a sintaxe do programa ALGOL gerado; isto porque a especificação de arquivo e formato associada às construções sintáticas acima mencionadas, não é analisada pelo compilador EXPAND II, o que equivale a dizer que tanto as especificações corretas como as incorretas são passadas literalmente para o programa ALGOL gerado.

CONSIDERAÇÕES GERAIS SOBRE A EDIÇÃO DOS PROGRAMAS GERADOS

Com o objetivo de tornar mais inteligível o programa ALGOL gerado, foi implementado um mini PRETTY-PRINTER associado ao processo de geração; suas principais características são:

- Faz identação até a coluna 50
- Alinha BEGIN-END e IF-THEN-ELSE
- Evita a divisão de itens sintáticos (tais como identificadores, constantes, etc ...)
- Muda de linha após a geração de cada declaração e de cada comando.

CAPÍTULO VI

OUTROS ASPECTOS DO ANALISADOR

VI.1. ANÁLISE LÉXICA

Esta análise é efetuada pela procedure SCANNER, a qual tem como função percorrer o programa fonte, agrupando os caracteres encontrados em ítens sintáticos também denominados tokens (os quais nada mais são que uma representação interna dos símbolos terminais e não-terminais da gramática (tabela VI.1). Além disso, devido às características particulares deste compilador (tais como o uso de macros sintáticas e o processo de tradução implementado), o SCANNER tem ainda como função efetuar alguns procedimentos especiais referentes a situações específicas que serão descritos nesta seção.

SÍMBOLOS NÃO-TERMINAIS		SÍMBOLOS TERMINAIS					
1	PROGR	20	.	39	DO	59	ID (identificador)
2	BLOCO	21	BEGIN	40	[	60	>
3	LISTA	22	END	41	]	61	<
4	COM	23	;	42	+	62	=
5	LDECL	24	:=	43	-	63	< >
6	VAR	25	GOTO	44	#	64	<=
7	ARIT	26	COND	45	=>	65	>=
8	IDEN	27	:	46	INTEGER	66	K (constante)
9	LCOND	28	RESULT	47	REAL	67	$ (EOF)
10	LEES	29	CALL	48	LABEL	68	MACRO
11	EES	30	READ	49	PROCEDURE	69	ARQMACRO
12	DECL	31	(	50	DECLARE	70	DEFINE
13	TERM	32	ARQFOR	51	FILE		
14	IDS	33	,	53	ENDMACRO		
15	EXPR	34	)	54	NEW		
16	LVAR	35	WRITE	55	*		
17	BLRES	36	FOR	56	/		
18	FACT	37	STEP	57	MOD		
19	CONST	38	UNTIL	58	DIV		

Tabela VI.1 - Representação Interna dos Símbolos da Gramática.

A seguir descreveremos os vários procedimentos (normais e especiais) efetuados pelo analisador léxico.

TRATAMENTO DE IDENTIFICADORES

Ao ser encontrado um identificador é verificado se seu tamanho está de acordo com o limite especificado; em seguida, é ativada a procedure PESQUISATABSIMB, cuja função é pesquisar e inserir os identificadores na tabela de símbolos (TABSIMB), para isto é usada uma função HASH |BARROS, 15| a qual especificará o índice (posição) correspondente ao identificador na TABSIMB, além de informar se o identificador em questão é ou não uma palavra reservada. Na TABSIMB cada identificador é representado por 5 atributos, a saber: tamanho do identificador, ponteiro para o poço de identificadores, duas informações relacionadas ao tratamento de macros (descritas anteriormente) e o token do identificador (sua representação interna).

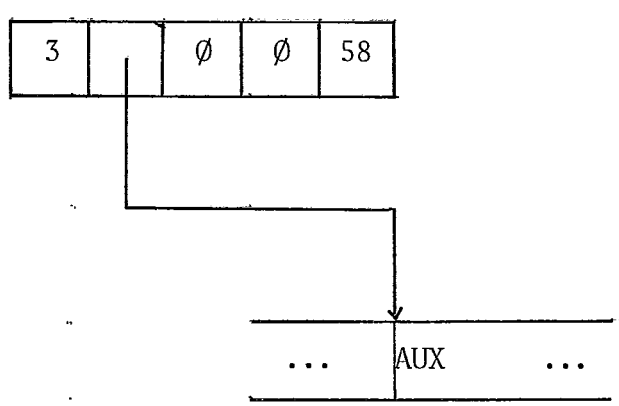


Figura VI.1 - Estrutura da Tabela de Símbolos

Além disso, são efetuados alguns procedimentos especiais para determinadas palavras reservadas, a saber:

BEGIN - Um novo nível é criado e são guardadas informações referentes a situação das tabelas do analisador sintático neste momento.

END - Volta-se ao nível anterior, fazendo com que as tabelas do analisador voltem a situação do nível, agora atual.

MACRO - Se a análise atual não for referente a definição de uma macro, as rotinas de tratamento de macros são ativadas.

ARQMACRO - É ativada a procedure PESQUISADIR, cuja função é possibilitar ao analisador o acesso ao DIRETÓRIO DE MACROS.

FILE - Armazenamento da especificação dos arquivos correspondente a declaração FILE em questão. Neste caso, além do token correspondente a palavra reservada FILE, será retornada a posição ocupada pela sua especificação na tabela de arquivo e formato (TABARQFOR). A figura VI.2 mostra a situação das estruturas envolvidas após a análise da declaração FILE ENT(KIND=READER),SAI(KIND=PRINTER);

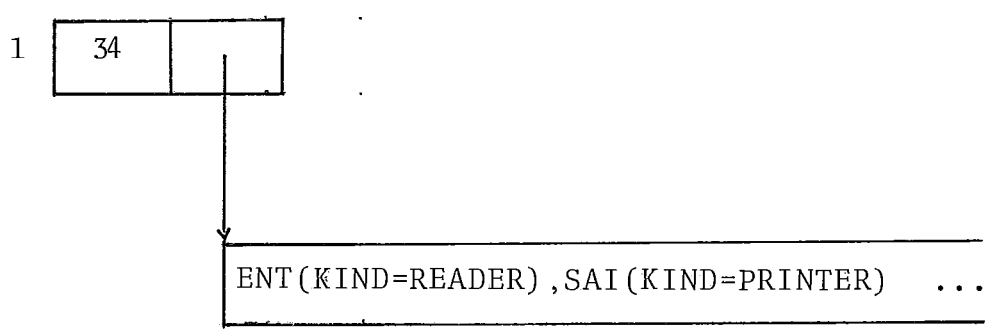


Figura VI.2 - Estrutura da Tabela de Arquivo e Formato

OBSERVAÇÃO: Os procedimentos associados às palavras reservadas BEGIN e END, fazem com que o analisador seja dinâmico e possibilitam o tratamento de erros na declaração de macros.

Cabe ainda salientar que:

- A tabela de símbolos é pré-inicializada com as palavras reservadas da linguagem e com suas respectivas representa

ções internas;

- As palavras reservadas MACRO, DEFINE e ARQMACRO tem função apenas na análise léxica e no tratamento de macros.

- Para cada identificador encontrado, exceto a palavra FILE, é retornado um par de elementos onde o primeiro é o token do identificador em questão e o segundo é a posição por ele ocupada na TABSIMB.

TRATAMENTO DE NÚMEROS

Consiste simplesmente no armazenamento do número, da forma como ele foi encontrado no fonte; aqui, o par retornado será composto pelo token correspondente a constante e pela posição por ela ocupada na tabela de números.

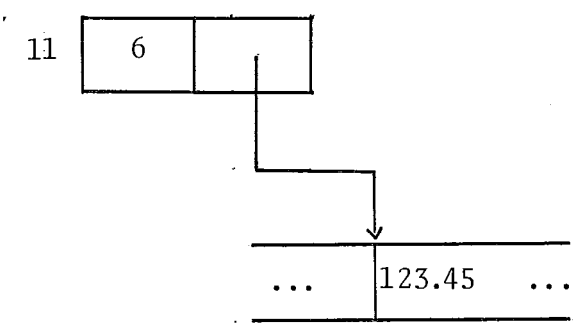


Figura VI.3 - Estrutura da Tabela de Números

TRATAMENTO DE SÍMBOLOS SIMPLES E DUPLOS

Para estas classes de símbolos, o tratamento consiste na obtenção do token correspondente ao símbolo encontrado, e o único valor retornado é o token obtido.

TRATAMENTO DE COMENTÁRIOS, CARACTER "∅" E CARACTER INVÁLIDO

Em todos estes casos, o controle da execução permanece com o analisador léxico, pois nenhum deles forma um item sintático; os procedimentos específicos são:

. COMENTÁRIOS - é forçada a leitura do próximo registro.

. CARACTER "Ø" - o fonte é percorrido até que um caracter ≠ de "Ø" seja encontrado.

. CARACTER INVÁLIDO - O usuário é notificado da presença do mesmo, através de uma mensagem de erro.

TRATAMENTO DE ARQFOR

Arqfor é um símbolo terminal criado pelo analisador para simular a presença da especificação de arquivo e formato referente aos comandos READ e WRITE. Este artifício foi usado para evitar que as referidas especificações fossem analisadas; a pesar disso, estas informações devem ser guardadas já que as mesmas serão necessárias na fase subsequente do processo de compilação. Assim sendo, sempre que for necessário criar um ARQFOR (isto ocorrerá quando os últimos dois símbolos analisados forem "READ" e "(" ou "WRITE" e "(") o fonte será percorrido até que o final da especificação de arquivo e formato seja encontrado e as informações encontradas serão armazenadas na tabela de arquivo e formato. Neste caso é retornado o token correspondente ao terminal ARQFOR e a posição ocupada pela referida especificação na TABARQFOR.

TRATAMENTO DA ENTRADA CRIADA PELO MACROEXPANSOR

Quando uma expansão de macro é efetuada, é criada uma entrada contendo os símbolos da definição da macro expandida, e os próximos símbolos requeridos pelo analisador sintático,serão exatamente os símbolos desta entrada. Para que isto seja possível, a leitura do fonte é inibida e cada vez que o SCANNER for ativado, deve ser retornado um símbolo dessa entrada.

Deve ser salientado, que os símbolos que compõem a definição de uma macro já foram analisados lexicamente e portanto, os símbolos da entrada criada pelo macroexpansor já estão codi-

ficados em tokens.

Quanto a análise propriamente dita, na maioria dos casos, consiste apenas em retornar as informações que encontram-se no topo da pilha de entrada (além do token, esta pilha conterá outras informações tais como posição de um identificador na TABSIMB, ordem de um parâmetro, etc...), entretanto duas situações especiais podem ocorrer: o elemento do topo é o terminal MACRO, neste caso as rotinas de tratamento de macros devem ser ativadas; ou, o elemento do topo é um não-terminal, neste caso além do token do não-terminal encontrado, é retornado o nodo raiz da sub-árvore correpondente ao parâmetro real utilizado.

ERROS LÉXICOS

Os possíveis erros léxicos detectados por este analisador são:

- Caracter inválido;
- Número de caracteres de um identificador maior que o permitido;
- Estouro de tabelas (TABSIMB, TABNUMEROS e TABARQFOR).

VI.2. ANÁLISE SINTÁTICA

Esta análise é efetuada pela procedure PARSER, cuja função é verificar se as estruturas sintáticas usadas pelo programador estão de acordo com as regras sintáticas da gramática.

Nesta implementação é usado o método de análise sintática SLR(1) (Simple/Left-to-right/Rightmost, com um símbolo de lookahead) |AHO & ULLAMANN, 2| com algumas modificações, atendendo as necessidades do método de macroexpansão implementado (estas modificações estão descritas no capítulo II).

A procedure PARSER é ativada em duas situações distintas: pelo programa principal, para comandar a análise sintática do programa fonte; e pelas rotinas de tratamento de macros, para verificar a conformidade sintática da definição de uma macro. No que segue, estas situações serão referenciadas respectivamente por análise do fonte e análise da definição.

ESTRUTURAS DO ANALISADOR

Com o objetivo de facilitar a extensão dinâmica do analisador, suas tabelas (geradas por |COSTA 10| a partir da gramática especificada no apêndice 1) estão armazenadas sob a forma de listas encadeadas; desta forma torna-se possível, de maneira simples e eficiente, incluir informações que permitam a análise sintática das macros introduzidas, e excluir estas mesmas informações no momento em que a macro, sob o ponto de vista lógico, deixar de existir.

As tabelas envolvidas no processo de análise sintática são:

- TABSLR - contém as ações de empilhamento SHIFT e GOTO;
- ESTADO - contém um ponteiro para TABSLR e o número da produção pela qual existe uma redução nesse estado (se existir);
- PRODUÇÃO - contém o não-terminal do lado esquerdo e o número de símbolos do lado direito de cada produção;
- TABFOLLOW - contém os símbolos seguidores válidos dos não-terminais;

. NAOTERMINAL - contém para cada não-terminal da gramática, a indicação do estado auxiliar para ele construído, e um ponteiro para TAB-FOLLOW.

Além dessas tabelas, o analisador usa uma pilha explícita denominada pilha sintática (cujo conteúdo, em cada momento, representa a situação da análise sintática) e outras três pilhas paralelas a pilha sintática, as quais apesar de serem mantidas sob o controle do analisador sintático, são utilizads apenas nos processos de macroexpansão e de tradução; estas pilhas são: pilha de tokens, pilha semântica e pilha de nodos.

FUNCIONAMENTO DO ANALISADOR

A análise sintática inicia-se com o estado inicial (estado Ø) na pilha sintática e desenvolve-se token a token até que o fim do programa fonte seja detectado (o qual é representado pelo símbolo "$"). A ação a ser efetuada a cada passo é determinada pelo estado que encontra-se no topo da pilha sintática (estado atual) e pelo token corrente na entrada (token atual); esta ação poderá ser: EMPILHA, REDUZ, ERRO ou ACEITA, as quais estão descritas a seguir juntamente com as situações em que poderão ocorrer:

AÇÃO EMPILHA - Será efetuada quando o estado atual possuir uma transição (SHIFT ou GOTO) com o token atual, e consiste no empilhamento do estado resultante desta transição e na ativação da procedure SCANNER ou da procedure PEGABETA (dependendo se a análise é do fonte ou da definição) para que um novo token torne-se disponível para a análise.

AÇÃO REDUZ - Será efetuada quando o estado atual possuir uma redução por uma produção cujo não-terminal do lado esquerdo contenha o token atual entre seus seguidores válidos (FOLLOW's) e consiste no seguinte:

Se a produção, através da qual a redução será feita, for da gramática base, um número de elementos igual ao tamanho do lado direito da referida produção será retirado da pilha sintática e o não-terminal correspondente ao lado esquerdo da produção será colocado na entrada; ainda neste caso se a análise que está sendo efetuada for do fonte e se até o momento nenhum erro foi detectado, então a procudure GERAARVORE é ativada para que seja construída a árvore sintática correspondente a construção reconhecida.

Se a produção corresponde a uma macro, então se a análise for do fonte e se nenhum erro ocorreu, a procedure MACRO-EXPANSOR é ativada para que seja realizada a expansão da macro; senão os estados correspondentes aos elementos da estrutura da macro reconhecida são retirados da pilha sintática e o não-terminal que especifica a categoria da referida macro é colocado na entrada.

AÇÃO ERRO - Ocorrerá quando a configuração do analisador não for a configuração de final de análise e o estado atual juntamente com o token atual não determinar nenhuma ação EMPILHA ou REDUZ. Neste caso se a análise que está sendo efetuada for do fonte, será ativada a procedure RECUPERAERRO, cuja função é modificar a configuração atual do analisador de forma tal que a análise possa ser continuada; senão, a análise é da definição, e neste caso é verificado se a configuração atual é a configuração de aceitação da definição, caso não seja, a referida análise é concluída anormalmente e a macro não poderá ser considerada.

AÇÃO ACEITA - Ocorrerá quando a configuração do analisador for $(\emptyset_{PROG}\ 1, \$)$, a qual é a configuração de fim de análise. Neste momento, se nenhum erro foi detectado, a procedure GERAALGOL será ativada para que o programa analisado seja traduzido (a partir da árvore sintática criada) para um programa equivalente na linguagem ALGOL.

VI.3. RECUPERAÇÃO DE ERROS

A recuperação de erros é efetuada pela procedure RECUPERAERRO e consiste simplesmente em permitir que a análise sintática seja continuada após a ocorrência de um erro.

Nesta implementação os erros podem ser detectados na fase de análise léxica (ERROS LÉXICOS), na fase de tratamento de macros (ERROS de MACROS) e durante a análise sintática (ERROS SINTÁTICOS); sendo que apenas os erros sintáticos nos interessam aqui.

ERROS SINTÁTICOS

Um erro sintático provém do uso incorreto das regras sintáticas da gramática e são detectados automaticamente pelo analisador, uma vez que este está preparado apenas para reconhecer construções sintáticas válidas. Ao detectar um erro, o analisador sintático fica sem ação, daí a necessidade de um recuperador de erros, pois a falta deste exigiria o término da análise no momento em que o primeiro erro sintático fosse detectado.

O método de recuperação de erros utilizado é o bastante difundido PANIC-MODE |AHO & ULLMANN 2|, o qual apesar de não ser perfeito, permite que a análise sempre seja concluída, satisfazendo com isto as necessidades desta implementação.

Devido as características especiais deste compilador, o método PANIC-MODE utilizado apresenta algumas variações com relação a definição, variações estas que poderão ser notadas no transcorrer desta seção.

DESCRIÇÃO DO MÉTODO DE RECUPERAÇÃO IMPLEMENTADO

Ao ser ativado, o primeiro passo do recuperador será notificar o usuário através de uma mensagem de erro, a qual dependerá do estado que encontra-se no topo da pilha sintática no momento da detecção do erro. Para os estados da gramática base existe uma tabela indicando a mensagem a ser emitida, enquanto

que para os estados criados em função das macros, será emitida uma mensagem padrão. As mensagens de erro emitidas pelo recuperador encontram-se no apêndice 6.

Em seguida, os elementos da entrada são desconsiderados até que um símbolo de sincronização seja encontrado; posteriormente são descartados sucessivos estados da pilha sintática até que seja encontrado um estado que, considerando o símbolo de sincronização encontrado, permita a continuidade da análise sintática. Os símbolos considerados de sincronização são: ";" , "END", "=>", "READ", "WRITE" e todas as palavras reservadas criadas em função das macros. O ";" e o "END" aparecem naturalmente por serem delimitadores de declarações e comandos; o símbolo "=>" é considerado de sincronização por ser um separador de expressões e comandos, no comando COND; "READ" e "WRITE" são considerados, para evitar que as especificações de arquivo e formato correspondentes, deixem de ser transparentes ao analisador; e finalmente, as palavras reservadas de macros são consideradas para evitar que os erros ocorridos numa referência a macro, gerem erros espúrios em demasia e se propaguem para além dos limites dessa referência.

Devido a importância do processo de macroexpansão neste trabalho, o presente recuperador possui alguns procedimentos especiais para o tratamento de erros ocorridos na utilização de macros, os quais são uma tentativa de restaurar ou sincronizar a macro envolvida; estes procedimentos serão ativados em uma das seguintes situações:

. Quando o símbolo de sincronização for uma palavra reservada de macro;

. Quando o estado encontrado na pilha sintática corresponder ao empilhamento de um elemento da estrutura da macro; ou,

. Quando as situações acima ocorrerem simultaneamente.

Na última situação especificada, se tanto o símbolo de sincronização como o estado encontrados estiverem relacionados a uma mesma macro, será efetuada a sincronização da macro, enquanto que em todos os outros casos o procedimento será de restauração.

CAPÍTULO VII

CONCLUSÕES E CONSIDERAÇÕES GERAIS

Um dos objetivos deste trabalho era testar a implementabilidade do esquema de extensão proposto em |RANGEL 1|, objetivo este plenamente alcançado, uma vez que a implementação foi possível e encontra-se disponível no computador BURROUGHS B6700; quanto a aceitabilidade do referido esquema (outro objetivo deste trabalho), nada pode ser dito ainda, já que somente após o uso exaustivo do mesmo será possível concluir pela sua aceitabilidade ou não.

Algumas considerações cabem aqui sobre os resultados da implementação efetuada:

1) O esquema implementado corresponde as espectativas previstas (a nível teórico) em |RANGEL 1|, isto é, a gramática base e seu analisador são extensíveis, desde que as restrições impostas inicialmente (a definição das macros deve ser derivável na gramática estendida, sem que seja feito uso da macro que está sendo declarada e a linguagem estendida deve ser SLR(1)) sejam respeitadas; possibilitando assim a análise sintática de referências a macros e a consequente expansão das mesmas, da maneira que havia sido previsto.

2) O tempo e o espaço requeridos para análise e expansão das macros, tendo em vista as vantagens oferecidas pelo processo, são aceitáveis; sendo que o espaço requerido é comparável aquele necessário para um compilador convencional da linguagem estendida. Por outro lado, o tempo total necessário para análise e tradução do programa EXPAND II e compilação e execução do programa ALGOL gerado, é excessivamente grande, justificando-se somente pela característica experimental desta implementação; contudo supõem-se que, se em vez da tradução fossem efetuadas a análise semântica e a geração de código, o tempo total seria aceitável.

3) Os mecanismos utilizados para declaração de macros, revelaram-se bastante eficientes, além do que são simples e maleáveis.

4) Dentre as restrições impostas para aceitação de uma macro, aquela que proíbe o uso de não-terminais (parâmetros) consecutivos na estrutura da macro poderia perfeitamente ser desconsiderada, aumentando consideravelmente o universo de macros aceitáveis, mas exigindo em troca esforços adicionais de implementação e cuidados especiais na especificação das macros.

5) O diretório de macros é outro ponto positivo desta implementação, pois apesar de ser simples, mostrou-se bastante eficiente nos testes efetuados, justificando plenamente a razão da sua existência.

6) Outro fator importante a ser considerado é a gramática base (EXPAND II) utilizada nesta implementação; esta gramática (definida também a nível experimental) apresenta pontos altamente positivos (como é o caso do comando COND, das declarações NEW e DECLARE e dos identificadores concatenados) sob o aspecto de extensibilidade, além de apresentar uma série de opções e recursos interessantes (estrutura de blocos por exemplo) e possuir um tamanho considerado ideal (67 produções) para a aplicação em questão. Por outro lado, a gramática considerada oferece algumas opções que apesar de interessantes (caso das declarações de constantes, do uso de blocos result como operando de expressões aritméticas, dos operadores de multiplicação "MOD" e "DIV" e do operador unário "+"), poderiam deixar de existir sem comprometer as potencialidades da linguagem gerada (pelo menos dentro de seus objetivos primordiais), e ressente-se de algumas opções (tais como array's bi-dimensionais, variáveis simples do tipo boolean e os operadores lógicos NOT, AND e OR) que certamente seriam mais úteis que as citadas anteriormente e não comprometeriam o tamanho final da gramática. Note que a declaração de array's bi-dimensionais (matrizes) através de macros é perfeitamente possível, entretanto, com base nos diversos testes realizados concluiu-se que para este caso específico, o meca-

nismo de macroexpansão mostrou-se pouco eficiente; justificando por isso, a introdução de array's bi-dimensionais na gramática base.

7) Dentre os aspectos de extensão não cobertos de forma simples pelo esquema implementado, destaca-se a possibilidade de introdução de novos tipos e estruturas de dados e a manipulação de listas através de macros sintáticas; o que certamente aumentaria em muito a flexibilidade e as potencialidades da linguagem estendida resultante.

8) As macros sintáticas, se bem aproveitadas, constituem-se em uma ferramenta bastante poderosa. Entretanto, é necessário que o usuário ao definir suas macros, tenha sempre em mente o resultado de sua expansão na linguagem base, para evitar que macros aparentemente simples e interessantes resultem em segmentos ineficientes e até mesmo inaceitáveis do ponto de vista lógico, transformando-as assim em uma ferramenta perigosa.

Enfim, sob meu ponto de vista, linguagens extensíveis através de macros sintáticas poderiam ser bem mais utilizadas (e consequentemente mais desenvolvidas), se não fosse a crescente pré-disposição em adaptar-se aos esquemas e estruturas introduzidas pelas linguagens mais difundidas. De qualquer forma, acredito que as linguagens extensíveis constituem campo e ferramenta importantes na pesquisa de novas facilidades em linguagens de programação.

BIBLIOGRAFIA

|1| RANGEL, J.L. e SCHNEIDER, S.M., - "Para uma Gramática Extensível, um Analisador Extensível?", Relatório Técnico, COPPE/UFRJ, Rio de Janeiro, 1981.

|2| AHO, A.V. e ULLMANN, J.D. - "Principles of Compiler Design", Califórnia, Addison-Wesley, 1978.

|3| AHO, A.V. e ULLMANN, J.D. - "The Theory of Parsing, Translation and Compiling", Volume I: Parsing, Volume II: Compiling, Prentice Hall Inc, 1972, 1973, Englewood Cliffs, N.J.

|4| BURROUGHS-B6000/7000, "Algol Language Reference Manual", 1977.

|5| BURROUGHS - "An Algol Programmer's Guide to the Use of the B6700", 1974.

|6| BURROUGHS-B6700/7700, "Extended Algol Language Reference Manual", 1971.

|7| SEGRE, L.M. - "Linguagem Algol B6000/B7000", Rio de Janeiro, COPPE/UFRJ, 1981.

|8| CHEATHAM, T.E. - "The Introduction of Definitional Facilities Into Higher Level Programming Languages",Proc. AFIPS, 1966; Fall Joint Computes Conference, USA, 29: 623-637, 1966.

|9| LEAVENWORTH, B.M. - "Syntax Macros and Extended Translation", CACM, USA, 9(11): 790-793, 1966.

|10| COSTA, R. - "Um Gerador de Tabelas de Análise Sintática para Gramáticas de Tipo SLR(1), LALR(1) e LR(1)", Rio de

Janeiro, Tese de M.Sc., COPPE/UFRJ, 1981.

|11| Proceedings of the International Symposium on Extensible Languages (September 6-8, 1971), SIGPLAN NOTICES, Vol. 6, NUM. 12, December, 1971.

|12| COLE, A.J. - "Macro Processors", Cambridge, Cambridge University Press, 1976.

|13| KULLOCK, P.C. - "EXPAND, Uma Linguagem Extensível Através de Macros Sintáticas: Análise Léxica e Tratamento das Macros", Rio de Janeiro, Tese de M.Sc., COPPE/UFRJ, 1981.

|14| ZAKIMI, M.B. - "EXPAND, Uma Linguagem Extensível Através de Macros Sintáticas: Compilador da Linguagem Base", Rio de Janeiro, Tese de M.Sc., COPPE/UFRJ, 1980.

|15| BARROS, L. - "Notas de Aulas do Curso Laboratório de Compiladores I", COPPE/UFRJ, 1981.

|16| BURROUGHS B6700 - "Input/Output Subsystem Manual", 1974.

|17| BAUER, F.L. & EICKEL, J.; ed. - "Compiler Construction an Advanced Course", Berlim, Hedelberg, Springer-Verlag, 1974.

|18| GRIES, D. - "Compiler Construction for Digital Computers", John Wiley & Sans, Wiley, 1971.

|19| KNUTH, D.E. - "The Art of Computer Programming , Volume 1: Fundamental Algorithm", Reading, Mass., Addison-Wesley, 1968.

APENDICE 1

GRAMATICA BASE EXPAND II

```
NUMERO                          PRODUCAO

   1   :    PROGR => BLOCO ','

   2   :    BLOCO => 'BEGIN' LISTA 'END'

   3   :    LISTA => LISTA ';' COM
   4   :           |  LDECL ';' COM
   5   :           |  COM

   6   :    COM   => VAR ':=' ARIT
   7   :           |  'GOTO' IDEN
   8   :           |  'COND' LCOND 'END'
   9   :           |  BLOCO
  10   :           |  IDEN ':' COM
  11   :           |  'RESULT' ARIT
  12   :           |  'CALL' IDEN
  13   :           |  'READ' '(' 'ARQFOR' ',' LEES ')'
  14   :           |  'WRITE' '(' 'ARQFOR' ',' LEES ')'
  15   :           |  'WRITE' '(' 'ARQFOR' ')'
  16   :           |

  17   :    LDECL => LDECL ',' DECL
  18   :           |  DECL

  19   :    VAR   => IDEN
  20   :           |  IDEN '(' ARIT ')'

  21   :    ARIT  => ARIT '+' TERM
  22   :           |  ARIT '-' TERM
  23   :           |  TERM

  24   :    IDEN  => IDEN '#' IOS
  25   :           |  IOS

  26   :    LCOND => EXPR '=>' COM ';' LCOND
  27   :           |  '=>' COM

  28   :    LEES  => LEES ',' EES
  29   :           |  EES

  30   :    EES   => ARIT
  31   :           |  'FOR' IDEN ':=' ARIT 'STEP' ARIT 'UNTIL'
                                              ARIT 'DO' EES

  32   :    DECL  => 'INTEGER' LVAR
  33   :           |  'REAL' LVAR
```

```
34    :             |   'INTEGER' IDEN ':' ARIT
35    :             |   'REAL' IDEN ':' ARIT
36    :             |   'LABEL' IDEN
37    :             |   'PROCEDURE' IDEN ';' BLOCO
38    :             |   'INTEGER' 'PROCEDURE' IDEN ';' BLOCO
39    :             |   'REAL' 'PROCEDURE' IDEN ';' BLOCO
40    :             |   'DECLARE' LDECL 'END'
41    :             |   'FILE'
42    :             |   'ARQMACRO'
43    :             |   'ENDMACRO'
44    :             |   'NEW' IDS

45    :     TERM    => TERM '*' FACT
46    :             |   TERM '/' FACT
47    :             |   TERM 'MOD' FACT
48    :             |   TERM 'DIV' FACT
49    :             |   FACT

50    :     IDS     => 'ID'

51    :     EXPR    => ARIT '>' ARIT
52    :             |   ARIT '<' ARIT
53    :             |   ARIT '=' ARIT
54    :             |   ARIT '<>' ARIT
55    :             |   ARIT '<=' ARIT
56    :             |   ARIT '>=' ARIT
57    :             |   ARIT

58    :     LVAR    => LVAR ',' VAR
59    :             |   VAR

60    :     BLRES   => BLOCO

61    :     FACT    => '(' EXPR ')'
62    :             |   VAR
63    :             |   BLRES
64    :             |   CONST
65    :             |   '-' FACT
66    :             |   '+' FACT

67    :     CONST   => 'K'
```

APENDICE 2

PALAVRAS RESERVADAS

1. PALAVRAS RESERVADAS EXPAND II

PROGR	LVAR	DO
BLOCO	BLRES	INTEGER
LISTA	FACT	REAL
COM	CONST	LABEL
LDECL	BEGIN	PROCEDURE
VAR	END	DECLARE
ARIT	GOTO	FILE
IDEN	COND	ENDMACRO
LCOND	RESULT	NEW
LEFS	CALL	MOD
EES	READ	DIV
DECL	ARQFOR	MACRO
TERM	WRITE	ARQMACRO
IDEN	FOR	DEFINE
IDS	STEP	
EXPR	UNTIL	

2. IDENTIFICADORES CRIADOS PELO COMPILADOR

- FUNXYZ'N' , ONDE 'N' E UMA CONSTANTE (COMPOSTA POR QUATRO DIGITOS) CRIADA SEQUENCIALMENTE A PARTIR DE '1001'.

- XYZTEMP00 , IDENTIFICADOR DO ARRAY DE TEMPORARIAS

- XYZERROTEMP , IDENTIFICADOR CRIADO QUANDO DA UTILIZACAO DE UM COMANDO 'RESULT' FORA DE UM BLOCO 'RESULT'.

- XYZ'M' , ONDE 'M' E UMA CONSTANTE (COMPOSTA POR CINCO DIGITOS) CRIADA SEQUENCIALMENTE A PARTIR DE '00001'.

3. PALAVRAS RESERVADAS ALGOL

ALPHA	EVENT	PROPERTY
ARRAY	FALSE	QUEUE
ASCII	FILE	REAL
BCL	FOR	REFERENCE
BEGIN	GO	STEP
BOOLEAN	HEX	SWITCH
COMMENT	IF	TASK
CONTINUE	INTEGER	THEN
DIRECT	LABEL	TRANSLATETABLE
DISKHEADER	LIST	TRUE
DO	LONG	TRUTHSET
DOUBLE	MESSAGE	UNTIL
EBCDIC	OWN	VALUE
ELSE	POINTER	WHILE
END	PROCEDURE	ZIP

NOTA : AS PALAVRAS LISTADAS ACIMA , SAO AS PALAVRAS RESERVADAS TIPO 1 ; ALEM DAS PALAVRAS RESERVADAS TIPO 1, A LINGUAGEM ALGOL POSSUI PALAVRAS RESERVADAS TIPO 2 (AS QUAIS QUANDO DECLARADAS COMO IDENTIFICADORES , DEIXAM DE SER RESERVADAS NO ESCOPO DA REFERIDA DECLARACAO) E PALAVRAS RESERVADAS TIPO 3 (AS QUAIS PODEM SER DECLARADAS COMO IDENTIFICADORES , MAS, QUANDO USADAS NO PROGRAMA DE ACORDO COM A ESPECIFICACAO SINTATICA DA LINGUAGEM ALGOL, TEM SIGNIFICADO RESERVADO). A LISTA DAS PALAVRAS RESERVADAS TIPO 2 E TIPO 3 PODEM SER ENCONTRADAS EM | BURROUGHS 4 |.

APENDICE 3

LISTAGEM DO COMPILADOR

```
%    UFRJ - UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
%
%    COPPE - COORDENACAO DOS PROGRAMAS DE POS-GRADUACAO EM ENGENHARIA
%
%    PROGRAMA DE ENGENHARIA DE SISTEMAS E COMPUTACAO
%
%    TESE DE MESTRADO
%
%    TITULO - EXPAND II : UM TRADUTOR PARA UMA LINGUAGEM EXTENSIVEL
%                         ATRAVES DE MACROS SINTATICAS
%    AUTOR       - OLINTO JOSE VARELA FURTADO
%    ORIENTADOR  - PROF. JOSE LUCAS MOURAO RANGEL NETTO
%    LINGUAGEM   - ALGOL
%    COMPUTADOR  - BURROUGHS B6700 - NCE/UFRJ
%    VERSAO      - 1
%    DATA        - AGOSTO / 1984
%
BEGIN
  %
  %  DIMENSAO MAXIMA DAS TABELAS DO ANALISADOR
  %
  %  VARIAVEIS A SEREM AUMENTADAS EM CASO DE ESTOURO DAS TABELAS
  %
  DEFINE NMAXMACRO   = 200#,
         NMAXPAR     = 600#,
         NMAXELEM    = 4096#,
         NMAXFOL     = 256#,
         NMAXEST     = 512#,
         NMAXSHIFT   = 2048#,
         NMAXCONST   = 256#,
         NMAXIDEN    = 351#,
         NMAXARQFOR  = 256#,
         NMAXNODOS   = 4096#,
         MAXTOPO     = 200#,
         NMAXNIV     = 64#,
         MAXTAMCONST= 512#,
         MAXTAMAF    = 4096#,
         MAXTAMIDEN = 3600#,
         MAXTAMLISTA= 64#,
         NMAXCOL     = 256#;
  BEGIN
    FILE ENT(KIND=PACK,FILETYPE=7,FAMILYNAME="USERPACK."),
         SAI(KIND=PACK,PROTECTION=SAVE),
         ARQGER(KIND=PACK,PROTECTION=SAVE,FILEKIND=ALGOLSYMBOL,
                          BLOCKSIZE=800,MAXRECSIZE=80,UNITS=CHARACTERS),
         ARQMACRO(KIND=PACK,TITLE="DIRETORIO/MACROS.",FILETYPE=7,
```

```
                        FAMILYNAME="USERPACK.");
  %
  %     ***    DECLARACOES GLOBAIS    ***
  %
  INTEGER INDICE,POSICAOTAB,% POSICAO DA TABELA DE SIMBOLOS
          NCAR,             % NUM. DE CARACTERES DE UM TOKEN
          NCOLS,            % NUM. DE COLUNAS A SEREM ANALIS.
          CODID,            % CODIGO DE UM IDENTIFICADOR
          CODEXEC,          % CODIGO DE EXECUCAO
          TOPOANTIGO1,TOPOANTIGOANT,    % AUXILIARES
          ULTIMOSIMBOLO,    % ULTIMO ELEMENTO DA EST. DE UMA MACRO
          AUXPONT,          % AUXILIAR
          NELEMENTOS,       % NUM. DE ELEMENTOS DA ESTRUTURA
          TOPENT,           % TOPO DA PILHA DE ENT. DO MACROEXP.
          PTVM,             % INDICE DO VETOR VETMACRO
          INDENT;           % INDICE AUX. DA PILHA DE ENTRADA
BOOLEAN FIMPROG,LISTAFONTE,TRADUZFONTE,
        DCLMACRO,LEMACRO,ESTOURO,
        CONTINUA,PALRES,LECAR,JATINHA;
POINTER PISS,PSS,PTAUX,
        PTID,PINICIOID,PTPOCO,
        PTI,PTPONTO,PTF;
INTEGER ARRAY TOKEN[1:3];
REAL    ARRAY CARTAO[1:14],
              VETMACRO[1: NMAXELEM ],
              TABSIMB[0: NMAXIDEN ,1:3],
              POCOID[0: MAXTAMIDEN ],
              DESCPAR[1: NMAXPAR ],DESCMACRO1[1:NMAXMACRO ],
              DESCMACRO[1: NMAXMACRO];
ALPHA   ARRAY SAIDASCAN[1:10];
%
%  ESTRUTURAS DE DADOS E VAR. UTILIZADAS NO TRAT. DE ARQFOR. E FILE.
%
BOOLEAN NAOABRIU,QUERARQFOR;
INTEGER INDAF,TAM,PTAF;
EBCDIC  ARRAY POCOARQFOR[1: MAXTAMAF];
REAL    ARRAY TABARQFOR[1: NMAXARQFOR];
DEFINE  TAMARQFOR[X1]  = TABARQFOR[X1].[47:24]#,
        PONTARQFOR[X1] = TABARQFOR[X1].[23:24]#;
%
% VARIAVEIS E ESTRUTURAS UTILIZADAS NO TRAT. DE NUMEROS
%
INTEGER INDTABNUM;
POINTER PTNI,PTNF;      % PONTEIROS PARA TABNUMEROS
REAL ARRAY TABNUMEROS[1:NMAXCONST],
           POCONUMEROS[0:MAXTAMCONST];
%
%  DECLARACAO DE VARIAVEIS USADAS NO PROC. DE EXPANSAO
%
INTEGER NPARDEF,        % NUMERO DE PARAMETROS DA DEFINICAO DA MACRO
        NPAREST,        % NUMERO DE PARAMETROS DA ESTRUTURA
        PTO,            % PONTEIRO PARA DESCRITOR DA MACRO
        NUMPRODRED,     % NUM. DA PODUCAO A SER USADA NA REDUCAO
        PTTESTA,INDMACRO,     % AUXILIARES
        TAMBETA,        % CONTEM O TAMANHO DA ESTRUTURA DA MACRO
        PDP,            % PONTEIRO PARA DESCRITOR DE PARAMETRO
        COL;            % COLUNA QUE UM NOVO SIMBOLO USADO NA MACRO
```

```
                        % PASSARA A TER NA TABSLR( SERA SEU TOKEN)
BOOLEAN ERROMACRO,      % INDICA OCORRENCIA DE ERRO EM UMA MACRO
        TEM,ESTA,JATEM,         % AUXILIARES
        RED,SHIF,       % CONTROLAM EXISTENCIA DE ACOES SHIFT E REDUCE
        DEUERRO,        % INDICA SE OCORREU ESTOURO NAS TAB. DO ANAL.
        ANTENT,         % INDICA SE O SIMBOLO ANTERIOR ERA NAO-TERM.
        NAOESLR;        % INDICA SE A G-BASE APOS A INCLUSAO DA MACRO
                        % CONTINUA SENDO SLR.
%
%  ESTRUTURA DE DADOS E VARIAVEIS USADAS NO MECANISMO DE EXTENSAO
%
INTEGER ESTH,I,J,AUXANT,CONT,AUXFOLBETA, % AUXILIARES
        INDUE,          % INDICE P/ ULTIMO ESTADO(TAMEST E PONTEST)
        AUX1,AUX2,AUX3,AUX4,AUX5,       % AUXILIARES
        INDEST,         % INDICE DE POCOEST
        PONTIA,         % PONT. P/ ITEM AUXILIAR (ITEMCOD E ITEMPROX)
        AUXFOL,         % CONTEM UM SIMB. FOLLOW DE CAT
        ULTEST,         % ULTIMO ESTADO CRIADO INICIALMENTE
        ULTIMO,         % ULTIMO ESTADO CRIADO
        COMECOSHIFT,    % PONT. PARA A PRIM. ACAO SHIFT DE UM ESTADO
        AUXPONTANT,     % AUXILIAR
        NDE,            % CONTROLA NINHOS DE MACRO-ENDMACRO
        NOMEMACRO,      % CONTEM TOKEN ATRIBUIDO AO NOME DE UMA MACRO
        ESTAUX1,        % ESTADO AUXILIAR A SER ANALISADO
        N,              % TAMANHO DE UMA LISTA DE ESTADOS TEMPORARIOS
        ULTCOLUNA,      % ULTIMO TOKEN CRIADO
        PROD,           % N. DA PRODUCAO PELA QUAL EXISTE REDUCAO
        CONTSHIFT;      % N. DE ACOES SHIFT CRIADAS P/ UM DET. ESTADO
VALUE ARRAY CATFOL(67,67,4(22),24,22,27,22,33,33,22,34,22,34,
                                                  33,22,33,33);
REAL ARRAY UNEEST[1:NMAXEST],   % USADO P/ CONTROLAR O CJTO EST. DE
           POCOEST[1:MAXTAMLISTA],   % ESTADOS CUJA UNIAO FOR, UM NOVO
           MACROEXTERNA[1:MAXTOPO],  % INF. SOBRE A MACRO EXTERNA
           ITEM[1001:1200]; % CJTO DE ITENS QUE FORMARA UM ESTADO.
INTEGER ARRAY MATUNIAO[1:NMAXCOL,1:MAXTAMLISTA]; %MATRIZ AUX. USADA NA
                                        %  FORMACAO DOS NOVOS ESTADOS.
DEFINE TAMEST [X1] = UNEEST[X1].[47:24]#,
       PONTEST[X1] = UNEEST[X1].[23:24]#,
       CODEST [X1] = POCOEST[X1].[47:24]#,
       LINKEST[X1] = POCOEST[X1].[23:24]#,
       PILHAMACRO[X]=MACROEXTERNA[X].[47:16]#,
       NODOMACRO  [X]=MACROEXTERNA[X].[15:16]#,
       STACKMACRO [X]=MACROEXTERNA[X].[31:16]#,
       ITEMCOD[X1] = ITEM[X1].[47:24]#,
       ITEMPROX[X1] = ITEM[X1].[23:24]#;
%
%  ESTRUTURAS USADAS NO CONTROLE DE NIVEIS
%
REAL ARRAY VETINDSHIF   [0:NMAXNIV],
           VETINDESTADO[0:NMAXNIV],
           VETINDFOL    [0:NMAXNIV],
           VETCOL       [0:NMAXNIV],
           VETPTVM      [0:NMAXNIV],
           VETPTD       [0:NMAXNIV],
           VETPDP       [0:NMAXNIV],
           VETINDRED    [0:NMAXNIV];
%
```

```
%  DECLARACAO DAS VARIAVEIS USADAS NO TRATAMENTO DE ARQMACRO
%
INTEGER INIDIR,FIMDIR,IMACRO,FMACRO,NCOLSAUX;
BOOLEAN LEARQ,ERROMACROARQ,PESQUIDIR;
POINTER PTIAUX,PTFAUX,PTMACRO;
REAL ARRAY NMACRO[1:10],CARTAOAUX[1:14];
%
%  DECLARACAO DE VARIAVEIS E ESTRUTURAS USADAS NA ANALISE SINTATICA
%                E NA GERACAO DE ARVORES
%
INTEGER TOPOANTIGO,     % TOPO P/ RETOMADA DA ANALISE SINTATICA
                        % APOS A ANALISE DA DEF. DE UMA MACRO.
        TOPO,           % TOPO DE PILHA , STACK E PILHA1
        PROX,           % PROXIMO SIMBOLO C/ O QUAL HA UMA TRANSICAO
        AUX,IND,XX,     %  AUXILIARES.
        LESQ,           % LADO ESQUERDO DE UMA PROD. (AUXILIAR)
        NDIR,           % NUM. DE ELEM. NO LADO DIREITO DA PRODUCAO
        GUARDA1,GUARDA2,        % AUXILIARES
        CAT,            % CATEGORIA SINTATICA DE UMA MACRO
        SHIFTMAX,       % INDICE DA ULTIMA ACAO SHIFT VALIDA (TABSLR)
        REDMAX,         % ULTIMA PRODUCAO VALIDA
        PBASE,          % NUM. DA ULTIMA PRODUCAO DA GRAMATICA BASE
        TOPOAR,         % TOPO DA PILHA USADA NA PESQUISA DE FOLLOWS
        CONTERRO,CONTREG,CONTALGOL,     % CONTADORES ESTATISTICOS
        INDSHIFT,       % INDICE DA TABELA DE SHIFTS (TABSLR)
        INDESTADO,      % INDICE DA TABELA ESTADO
        NIVEL,NIV,NIVELD,  % NIVEL DO BLOCO ATUAL
        NIVELANTERIOR, % NIVEL EM QUE ESTA SENDO DECL. A MACRO
        NUMBEGIN,       % USADO PARA CONTOLE DE NIVEIS
        INDFOL,         % INDICE DA TABFOLLOW
        INDRED;         % INDICE DA TABELA REDUCAO.
BOOLEAN ACEITA,         % USADA PARA SINALIZAR PARADA DO PARSER
        ERROSINTATICO, %CONTROLA A EXECUCAO DA GERAARVORE
        SHIFT,          % INDICA SE HOUVE SUCESSO COM SHIFT
        REDUZ,          % INDICA SE HOUVE SUCESSO COM REDUCE
        ACHOUF,         % SINALIZA EXISTENCIA DE UMA ACAO REDUCE EM
        PROCURA,        % UM DETERMINADO ESTADO.(ACHOUF)
        NAODER,         % INDICA SE UMA MACRO E DERIVAVEL NA GRAM. BASE
        TESTANDO;       % INDICA SE A ANALISE E DO FONTE OU DA DEFINIC.
INTEGER ORDEM,          % ORDEM DO PARAMETRO NA ESTRUTURA DA MACRO
        CONTBETA,       % TAMANHO DA DEFINICAO DA MACRO
        CONTME,         % CONTADOR CONTROLADOR DE MACRO-ENDMACRO
        NUMFUN,NIVBEG, % GERACAO DE FUNCOES
        NUMRES,         % GERAARVORE - NUMERO DE BLRES
        TOPONC,MAIORTOPONC,TOPOCOND,TNC,   % USADOS NA GERAC.DE CONST
        PTBETA,PONTMACRO,  % AUXILIARES
        NUMELEM,        % NUMERO DE ELEMENTOS DA ESTRUTURA DA MACRO
        P,              % INDICE DA ARVORE GERADA (CONTROLE DE NODOS)
        AUXIF,AUXINV,   % AUXILIARES
        PONTPAR;        % AUX-PONTEIRO PARA DESCRITOR DE PARAMETRO
REAL ARRAY TABSLR[0:NMAXSHIFT],     % CONTEM AS ACOES DE SHIFT E GOTO
           NAOTERMINAL[0:19],  % INFORMACOES SOBRE CADA NAO-TERMINAL
           TABFOLLOW[1:NMAXFOL],   % FOLLOW DOS NAO-TERMINAIS
           TABEST[0:150],     % CODERRO DOS ESTADOS
           REDUCAO[0:NMAXMACRO],      % INFORMACOES SOBRE CADA PRODUCAO
           ESTADO[0:NMAXEST];       % PONT. PARA AS ACOES SHIFT E RED.
INTEGER ARRAY PILHA[1:MAXTOPO],     % PILHA SINTATICA
```

```
                STACK[1:MAXTOPO],     % PILHA SEMANTICA
                PILHA1[1:MAXTOPO],    % PILHA DE TOKENS
                PTNODO[1:MAXTOPO],    % CONTEM PONTEIROS PARA OS NODOS
                ARVORE[0:NMAXNODOS,1:2], % CONTEM OS NODOS GERADOS
                LISTANC[1:64],          % CONTEM OS NODOS RESULT
                QUANTRES[1:64],PONTRES[1:64],PILHANC[1:100],
                QUANTCOND[0:20],PONTCOND[0:20],
                PILHAAUXR[1:60],      % AUXILIAR
                GUARDATOK[1:2],       % AUXILIAR
                ENTSINT[1:MAXTOPO],  % PILHA SINT. USADA NA MACROEXPANSA
                ENTSEM [1:MAXTOPO];  % PILHA SEMANT. USADA NA MACROEXPAN
%
% CODIGO DOS NODOS ( DE ACORDO COM O NUMERO DA PRODUCAO )
%
DEFINE PONTO = 1#,    NREAD = 13#,     LCON = 25#,  FI = 38#, MM = 54#,
       BE = 2#,       NWRITE = 14#,    FALS = 27#,  FR = 39#, MEI = 55#,
       LC = 3#,       WLIT = 15#,     LES = 28#,   NFILE=41#, MAI = 56#,
       LOC = 4#,      VAZIO = 16#,    NFOR = 31#,   NNEW= 44#, LV = 58#,
       ATRIB = 6#,    VEZES = 45#,    INTEG = 32#, NMOD= 47#, MEU = 65#,
       NGOTO = 7#,     BRKT = 20#,     REALL = 33#, NDIV= 48#, MAU =66#,
       DFCEND = 40#, MAIS = 21#,     CONSTI= 34#, ID = 50#, DIG = 67#,
       ROT = 10#,     MENOS = 22#,    CONSTR= 35#, NMA = 51#, JV = 24#,
       RES = 11#,     DIVIS = 46#,    LAB = 36#,   NME = 52#, LD = 17#,
       CALL = 12#,    VERD = 26#,     PROC = 37#,  NIG = 53#, CE = 8#,
       ARQFOR = 5#;
DEFINE DIR[X] = ARVORE[X,2]#,              % LINK DA DIREITA DE NODO
       ESQ[X] = ARVORE[X,1].[47:13]#,      % PONT P/ FILHO MAIS A ESQ.
       INV[X] = ARVORE[X,1].[34:13]#,      % CONTEM O APONTADOR
       CODN[X] = ARVORE[X,1].[21:08]#,     % CONTEM O CODIGO DO NODO
       INFO[X] = ARVORE[X,1].[13:14]#,     % CONTEM INFORMACOES DO NODO
       %
       NDTOP  = PTNODO[ TOPO ]#,
       NDTOP1 = PTNODO[ TOPO - 1 ]#,
       NDTOP2 = PTNODO[ TOPO - 2 ]#,
       NDTOP3 = PTNODO[ TOPO - 3 ]#,
       NMAXNODO = 4096#;                   % N. MAXIMO DE NODOS
DEFINE SIMBENT[X]=TABSLR[X].[47:16]#,          % ENTRADA
       ESTPROD[X]=TABSLR[X].[31:16]#,          % ESTADO SEGUINTE
       LINK[X]   =TABSLR[X].[15:16]#,          % PROXIMA ENTRADA
       %
       ESTADONT[X]=NAOTERMINAL[X].[47:16]#,    % ESTADO AUXILIAR
       FIRSTNT[X] =NAOTERMINAL[X].[31:16]#,    % PONT. P/ FIRST
       FOLLOWNT[X]=NAOTERMINAL[X].[15:16]#,    % PONT. P/ FOLLOW
       %
       LFOLLOW[X]=TABFOLLOW[X].[47:24]#,       % CONTEM UM TERMINAL
       LINKFOL[X]=TABFOLLOW[X].[23:24]#,       % PROXIMO TERMINAL
       %
       ESTXNT[X,Y]=TABEST[X].[Y:1]#,        % CONTEM 1 SE X ACEITA Y
       CODERRO[X]=TABEST[X].[47:8]#,        % COD ERRO DO ESTADO X
       %
       LESQUERDO[X]=REDUCAO[X].[47:24]#,       % N-TERM. DO LADO ESQ
       LDIREITO [X]=REDUCAO[X].[23:24]#,       % TAMANHO DO LADO DIR
       %
       PTSHIFT  [X]=ESTADO[X].[47:24]#,        % PONT.P/ TABSLR
       PTREDUCAO[X]=ESTADO[X].[23:24]#;        % NUM DA PROD USADA
%
% DEFINE'S DAS ESTRUTURAS USADAS EM: MACROS,PARAMETROS TABSIMB,TABNUM
```

```
%
DEFINE TAMANHOID[Z1]=TABSIMB[Z1,1].[47:08]#,
       PTPOCOID [Z1]=TABSIMB[Z1,1].[39:16]#,
       PARXMACRO[Z1]=TABSIMB[Z1,1].[23:04]#,
       PONTPXM  [Z1]=TABSIMB[Z1,1].[19:20]#,
       % PALAVRA 2 = TOKEN
       % PALAVRA 3 = INDEFINIDA
       MACROCAT [Y1]=DESCMACRO[Y1].[47:08]#,
       MACROTA  [Y1]=DESCMACRO[Y1].[39:08]#,
       MACROALFA[Y1]=DESCMACRO[Y1].[31:12]#,
       MACROTB  [Y1]=DESCMACRO[Y1].[19:08]#,
       MACROBETA[Y1]=DESCMACRO[Y1].[11:12]#,
       MACROEXT[Y1] =DESCMACRO1[Y1].[47:24]#,
       MACROINT[Y1] =DESCMACRO1[Y1].[23:12]#,
       MACRONOME[Y1]=DESCMACRO1[Y1].[11:12]#,
       % DESCRITORES DOS PARAMETROS
       PARMACRO[X1] = DESCPAR[X1].[47:16]#,
       PARCATEG[X1] = DESCPAR[X1].[31:08]#,
       PARINDMACRO[X1] = DESCPAR[X1].[23:08]#,
       PARORDEM[X1] = DESCPAR[X1].[15:08]#,
       PARPTM   [X1] = DESCPAR[X1].[07:08]#,
       TAMANUM[X2]  = TABNUMEROS[X2].[47:24]#,
       PONTNUM[X2]  = TABNUMEROS[X2].[23:24]#,
       % CAMPOS DO VETMACRO
       C1 = [47:16]#,
       C2 = [31:16]#;
%
%  VARIAVEIS COM CODIGOS FIXOS ( DEFINICAO )
%
DEFINE CODDOLLAR = 67#,
       CODFILE   = 51#,
       CODMACRO  = 68#,
       CODENDMACRO=53#,
       CODNT     = 00#,
       CODIDEN   = 59#,
       CODRED    = 5555#,
       CODARQFOR = 32#,
       CODBEGIN  = 21#,
       CODEND    = 22#,
       CODARQMACRO=69#,
       CODENDARQ = 52#,
       CODDEFINE = 70#,
       CODDIG    = 66#;
%
%  SIMOLOS DUPLOS ( DEFINICAO )
%
DEFINE DP = ":" #,
       IG = "=" #,
       ME = "<" #,
       MA = ">" #,
       PT = "." #;
%
```

```
%  ************************************************************
%  *                                                          *
%  *  DECLARACAO DAS PROCEDURE ( FORWARD )                    *
%  *                                                          *
%  ************************************************************
%
PROCEDURE SCANNER; FORWARD;
%
PROCEDURE PEGABETA; FORWARD;
%
PROCEDURE PARSER; FORWARD;
%
PROCEDURE VARREMACRO; FORWARD;
%
PROCEDURE VERFOLLOW; FORWARD;
%
% *** DECLARACAO E DEFINICAO DAS PROCEDURES ***
%
```

```
%*****************************************************************
%
PROCEDURE MSGERRO(ERRO);
%
%          FUNCAO = EMITIR MENSAGENS DE ERRO
%*****************************************************************
INTEGER ERRO;
BEGIN
  CONTERRO:= * + 1;
  CASE ERRO OF
  BEGIN
    1   : WRITE(SAI,</,T*,"* TAMANHO DO IDENTIFICADOR NAO DEVE EXCE",
                            "DER 64 CARACTERES",/>,OFFSET(PTI)+12);
    3   : WRITE(SAI,</,T*,"* CARATER NAO PERTENCENTE A LINGUAGEM  ",/>
                            ,OFFSET(PTI)+12);
    4   : WRITE(SAI,</,T12,"*** CATEGORIA DA MACRO NAO ESPECIFICADA  "
                            ,/>);
    5   : WRITE(SAI,</,T12,"*** NOME DA MACRO DEVE SER UM IDENTIFICA",
                            "DOR ",/>);
    6   : WRITE(SAI,</,T12,"*** NAO E PERMITIDA A OCORRENCIA DE PARA",
                            "METROS CONSECUTIVOS",/>);
    7   : WRITE(SAI,</,T12,"*** OS PARAMETROS DA MACRO DEVEM SER DIS",
                            "TINTOS ENTRE SI",/>);
    8   :WRITE(SAI,</,T12,"*** IDENTIFICADOR DE PARAMETRO INVAL.",/>);
    9   : WRITE(SAI,</,T12,"*** NUMERO DE PARAMETROS DA ESTRUTURA NA",
                            "O COINCIDE C/ O N. DA DEFINICAO",/>);
    10  :WRITE(SAI,</,T12,"*** USO INDEVIDO DE IDEN. DE PARAMETO",/>);
    11  : WRITE(SAI,</,T12,"*** SINTAXE DA DEFINICAO DA MACRO ESTA I",
                            "NCORRETA",/>);
    12  : WRITE(SAI,</,T12,"*** CONFLITO SHIFT-REDUCE",/>);
    13  : WRITE(SAI,</,T12,"*** CONFLITO REDUCE-REDUCE",/>);
    14  : WRITE(SAI,</,T12,73("*"),/>);
          WRITE(SAI,</,T12,"* A PARTIR DESTE PONTO O PROGRAMA FOI IG",
                            "NORADO POR HAVER MACRO ERRADA",/>);
          WRITE(SAI,</,T12,73("*"),/>);
    15  : WRITE(SAI,</,T12,"*** A ANALISE DO BLOCO ACIMA NAO FOI CON",
                            "CLUIDA POR HAVER MACRO ERRADA ",/>);
    16  : WRITE(SAI,</,T12,"*** MACRO NAO EXISTENTE NO DIRETORIO",/>);
    17  : WRITE(SAI,</,T12,"*** ESPERAVA-SE , OU ; NA DECL. ARGMA.">);
    18  : WRITE(SAI,</,T12,"*** DECLARACAO DE MACRO INCOMPLETA">);
    19  : WRITE(SAI,</,T12,"*** AREA DE CODIGO CHEIA (NODOS)  ">);
    100 : IF PILHA[TOPO] EQL 0 THEN ACEITA:=TRUE;
          WRITE(SAI,</,T12,"*** ESPERAVA-SE UM BEGIN">);
    101 : WRITE(SAI,</,T12,"*** FINAL DE PROGRAMA ESPERADO">);
          ACEITA:=TRUE;
    102 : WRITE(SAI,</,T12,"*** ESPERAVA-SE . DE FIM DE PROGRAMA">);
          ACEITA:=TRUE;
    103 : WRITE(SAI,</,T12,"*** ESPERAVA-SE DECLARACAO OU COMANDO">);
    104 : WRITE(SAI,</,T12,"*** FALTOU DELIMITADOR ( ; OU END )">);
    105 : WRITE(SAI,</,T12,"*** ESPERAVA-SE   ;  ">);
    106 : WRITE(SAI,</,T12,"*** FALTOU SINAL DE ATRIBUICAO (:=)">);
    107 : WRITE(SAI,</,T12,"*** ESPERAVA-SE UM IDENTIFICADOR">);
    108 : WRITE(SAI,</,T12,"*** EXPRESSAO INVALIDA">);
    109 : WRITE(SAI,</,T12,"*** SEGUIDOR DE IDENTIFICADOR INVALIDO">);
    110 : WRITE(SAI,</,T12,"*** ESPERAVA-SE UM ( ">);
    111 : WRITE(SAI,</,T12,"*** ESPERAVA-SE IDENT. OU PROCEDURE">);
    112 : WRITE(SAI,</,T12,"*** DECLARACAO INVALIDA">);
```

```
113 : WRITE(SAI,</,T12,"*** COMANDO INVALIDO">);
114 : WRITE(SAI,</,T12,"*** ESPERAVA-SE END ">);
115 : WRITE(SAI,</,T12,"*** ESPERAVA-SE =>  ">);
116 : WRITE(SAI,</,T12,"*** SEGUIDOR DE EXPRESSAO INVALIDO">);
117 : WRITE(SAI,</,T12,"*** FALTANDO ESPECIFICACAO DE ARQUIVO",
          " E FORMATO ">);
118 : WRITE(SAI,</,T12,"*** ESPERAVA-SE ; OU #">);
119 : WRITE(SAI,</,T12,"*** ESPERAVA-SE )">);
120 : WRITE(SAI,</,T12,"*** FALTANDO )">);
121 : WRITE(SAI,</,T12,"*** ESPERAVA-SE , ">);
122 : WRITE(SAI,</,T12,"*** ESPERAVA-SE := OU # ">);
123 : WRITE(SAI,</,T12,"*** ESPERAVA-SE STEP ">);
124 : WRITE(SAI,</,T12,"*** ESPERAVA-SE UNTIL">);
125 : WRITE(SAI,</,T12,"*** ESPERAVA-SE DO ">);
126 : WRITE(SAI,</,T12,"*** ESPERAVA-SE # ">);
127 : WRITE(SAI,</,T12,"*** ESPERAVA-SE OP. DE MULTIPLICACAO">);
128 : WRITE(SAI,</,T12,"*** ESPERAVA-SE UM ) OU UMA ,">);
129 : WRITE(SAI,</,T12,"*** DELIMITADOR(, ; OU END) ESPERADO">);
130 : WRITE(SAI,</,T12,"*** ERRO NA UTILIZACAO DE UMA MACRO">);
131 : WRITE(SAI,</,T12,"*** FINAL DE PROGRAMA INESPERADO">);
132 : WRITE(SAI,</,T12,"*** SEGUIDOR DE BLOCO INVALIDO ",>);
133 : WRITE(SAI,</,T12,"*** SEGUIDOR DE VARIAVEL INVALIDO">);
134 : WRITE(SAI,</,T12,"*** SEGUIDOR DE CONSTANTE INVALIDO">);
135 : WRITE(SAI,</,T12,"*** SEGUIDOR DE BLOCO RESULT INVALIDO">);
136 : WRITE(SAI,</,T12,"*** ESPERAVA-SE  +  -  ;  OU  END ">);
137 : WRITE(SAI,</,T12,"*** ESPERAVA-SE  #  ;  OU  END ">);
138 : WRITE(SAI,</,T12,"*** ESPERAVA-SE  END ">);
139 : WRITE(SAI,</,T12,"*** ESPERAVA-SE  +  -  ; OU )">);
200 : WRITE(SAI,</,T12,"*** NUMERO DE MACROS MAIOR QUE O PERMITI",
                       "DO - AUMENTAR NMAXMACRO ",/>);
201 : WRITE(SAI,</,T12,"*** NUMERO DE PARAMETROS MAIOR QUE O PER",
                       "MITIDO - AUMENTAR NMAXPAR ",/>);
202 : WRITE(SAI,</,T12,"*** NUMERO DE ELEMENTOS MAIOR QUE O PERM",
                       "ITIDO - AUMENTAR NMAXELEM ",/>);
203 : WRITE(SAI,</,T12,"*** NUMERO DE FOLLOWS MAIOR QUE O PERMIT",
                       "IDO - AUMENTAR NMAXFOL ",/>);
204 : WRITE(SAI,</,T12,"*** NUMERO DE ESTADOS MAIOR QUE O PERMIT",
                       "IDO - AUMENTAR NMAXEST ",/>);
205 : WRITE(SAI,</,T12,"*** NUMERO DE SHIFTS  MAIOR QUE O PERMIT",
                       "IDO - AUMENTAR NMAXSHIFT ",/>);
206 : WRITE(SAI,</,T12,"*** NUMERO DE CONSTANTES MAIOR QUE O PER",
                       "MITIDO - AUMENTAR NMAXCONST ",/>);
207 : WRITE(SAI,</,T12,"*** NUMERO DE IDENTIFICADORES MAIOR QUE ",
                       "O PERMITIDO - AUMENTAR NMAXIDEN ",/>);
208 : WRITE(SAI,</,T12,"*** NUMERO DE ESPECIFICACAO DE ARQUIVO-F",
      "ORMATO MAIOR QUE O PERMITIDO - AUMENTAR NMAXARQFOR ",/>);
209 : WRITE(SAI,</,T12,"*** NUMERO DE NODOS MAIOR QUE O PERMITID",
                       "O - AUMENTAR NMAXNODO ",/>);
210 : WRITE(SAI,</,T12,"*** ESTOURO DA PILHA SINTATICA - AUMENTA",
                       "R MAXTOPO ",/>);
211 : WRITE(SAI,</,T12,"*** NUMERO DE NIVEIS MAIOR QUE O PERMITI",
                       "DO - AUMENTAR NMAXNIV ",/>);
212 : WRITE(SAI,</,T12,"*** ESTOURO DO POCO DE NUMEROS - AUMENTA",
                       "R MAXTAMCONST ",/>);
213 : WRITE(SAI,</,T12,"*** ESTOURO DO POCO DE ESPECIFICACOES DE",
                       " ARQUIVO-FORMATO - AUMENTAR MAXTAMAF",/>);
214 : WRITE(SAI,</,T12,"*** ESTOURO DO POCO DE IDENTIFICADORES -",
```

```
                                " = AUMENTAR MAXTAMIDEN",/>);
    215 : WRITE(SAI,</,T12,"*** NUMERO DE ESTADOS DE UMA LISTA EXCED",
                                "EU O MAXIMO = AUMENTAR MAXTAMLISTA ",/>);
    216 : WRITE(SAI,</,T12,"*** NUMERO DE PALAVRAS RESERVADAS EXCEDE",
                                "U O MAXIMO = AUMENTAR NMAXCOL ",/>);
   ELSE : BEGIN
          END
  END OF CASE;
END; %OF PROCEDURE MSGERRO
%
```

```
%*****************************************************************
%
PROCEDURE MACROERROR;
%
%          FUNCAO = IGNORAR O BLOCO ONDE ESTA SITUADA A MACRO
%                   DEFINIDA ERRONEAMENTE.
%*****************************************************************
BEGIN
  IF LEARQ
  THEN BEGIN
         ERROMACROARQ:=TRUE;LEARQ:=FALSE;
         FOR I:=1 STEP 1 UNTIL 14 DO CARTAO[I]:=CARTAOAUX[I];
         PTI:=PTIAUX;PTF:=PTFAUX;NCOLS:=NCOLSAUX;
       END;
  QUERARQFOR:=FALSE;
  LECAR:=ERROMACRO:=ERROSINTATICO:=ERROMACROARQ:=TRUE;
  IF NIVELANTERIOR LEQ 0
  THEN BEGIN
         FIMPROG:=ACEITA:=TRUE;
         MSGERRO(14);
       END
  ELSE BEGIN
         WHILE NIVEL > NIVELANTERIOR AND TOKEN[1] NEQ 67
         DO SCANNER;
         CONTINUA:=FALSE;
         TOKEN[1]:=2;    % CODIGO DE BLOCO
         MSGERRO(15);
         % DESEMPILHAR TUDO OQUE ESTIVER NA PILHA SINTATICA ATE
         % ENCONTRAR O BEGIN CORRESPONDENTE AO ULTIMO END ENCONTRADO
         WHILE PILHA1[TOPO] NEQ CODBEGIN AND TOPO > 1
         DO TOPO:=TOPO - 1;
       END;
  TOPO:=TOPO-1;
END;
%
```

```
%*****************************************************************
%
PROCEDURE EXPANDTAB;
%
%          FUNCAO = EXPANDIR AS TABELAS DO ANALISADOR , PARA INCLUIR
%                   O EFEITO DA MACRO MAIS RECENTEMENTE DECLARADA
%*****************************************************************
BEGIN
  INTEGER I1,J1;
  PROCEDURE REDFOL;
  %  FUNCAO = EXPLICITAR AS REDUCOES DE UM ESTADO
  BEGIN
    WHILE AUX2 < REDMAX
    DO BEGIN
         IF LFOLLOW[AUX2] < 20
         THEN BEGIN
                TOPOAR:= * + 1;
                PILHAAUXR[TOPOAR]:=AUX2;
                AUX2:=FOLLOWNT[LFOLLOW[AUX2]];
                REDFOL;
                TOPOAR:= * - 1;
                IF TOPOAR NEQ 0
                THEN BEGIN
                       AUX2:=LINKFOL[PILHAAUXR[TOPOAR]];
                       PILHAAUXR[TOPOAR]:=AUX2;
                     END;
              END
         ELSE BEGIN
                MATUNIAO[LFOLLOW[AUX2],J]:=2000+NUMPRODRED;
                AUX2:=LINKFOL[AUX2];
                IF TOPOAR > 0
                THEN PILHAAUXR[TOPOAR]:=AUX2;
              END;
       END;
  END;
  PROCEDURE INCLUINALISTA;
  %  FUNCAO = INCLUIR UM NOVO ESTADO EM UMA LISTA DE ESTADOS TEMP,
  %           VERIFICANDO ANTES SE JA NAO ESTAVA
  BEGIN
    AUX5:=PONTEST[ESTH];
    ESTA:=FALSE;
    WHILE AUX5 < 9999 AND NOT ESTA
    DO IF CODEST[AUX5] EQL MATUNIAO[I,J]
       THEN ESTA:=TRUE
       ELSE BEGIN
              AUXANT:=AUX5;
              AUX5:=LINKEST[AUX5];
            END;
       IF NOT ESTA
       THEN BEGIN
              INDEST:=* + 1;
              IF INDEST GEQ MAXTAMLISTA
              THEN BEGIN
                     MSGERRO(215); ESTOURO:=TRUE;
                     J:=N;          I:= COL;
                   END;
              LINKEST[AUXANT]:=INDEST;
```

```
            CODEST[INDEST]:=MATUNIAO[I,J];
            LINKEST[INDEST]:=9999;
            TAMEST[ESTM] := TAMEST[ESTM] + 1;
          END;
END;  % PROCEDURE INCLUINALISTA
%
%  VERIFICA SE A =>+ B
%
NAOEDER:=FALSE;
TOPOANTIGO:=TOPO;
TESTANDO:=TRUE;
PILHA[TOPO:=*+1]:=ESTADONT[CAT];        % EMPILHA ESTADO AUXILIAR
PTTESTA:= MACROBETA[PTD] + 1;           % APONTA P/PRIM.SIMB.DE BETA
TAMBETA:= MACROTB[PTD];                 % GUARDA TAMANHO DA DEFINICAO
AUXFOLBETA:=CATFOL[ CAT ];              % COLOCA UM FOLLOW DE A EM B
CONT:=0;
TOKEN[1]:=TOKEN[2]:=0;
PEGABETA;
PARSER;
TOPO:=TOPOANTIGO;
TESTANDO:=ACEITA:=FALSE;
GUARDATOK[1]:=0;
IF NAOEDER
THEN BEGIN
       MSGERRO(11);
       NAOESLR:=TRUE;
     END
ELSE BEGIN
       %
       %  CRIACAO DE ESTADOS NOVOS ( FASE 3 DO ALG. DE EXTENSAO )
       %
       %  MONTA ITENS E CONJUNTO DE ESTADOS P1,P2,....,PJ
       %
       PONTIA:=1001;           % PONTEIRO PARA ITEM AUXILIAR
       INDUE:=INDESTADO+1;     % INDICE P/ ULTIMO ESTADO (174)
       INDEST:=1;
       PROX:=MACROALFA[PTD];  % PRIM. SIMBOLO DE ALFA
       ULTIMO:=PROX+MACROTA[PTD];
       PROX:=PROX + 1;
       WHILE PROX < ULTIMO
       DO BEGIN
            ITEMCOD[PONTIA]:=VETMACRO[PROX].C1;
            ITEMPROX[PONTIA]:=PONTIA + 1;
            TAMEST[INDUE]:=1;
            PONTEST[INDUE]:=INDEST;
            CODEST[INDEST]:=PONTIA;
            INDEST:= * + 1;
            IF VETMACRO[PROX].C1 < 20      % NAO-TERMINAL
            THEN BEGIN
                   LINKEST[INDEST - 1]:=INDEST;
                   TAMEST[INDUE] := 2;
                   CODEST[INDEST]:=ESTADONT[VETMACRO[PROX].C1];
                   INDEST:= * + 1;
                 END;
            INDUE:= * + 1;
            IF INDUE GEQ NMAXEST
            THEN BEGIN
```

```
                MSGERRO(204); ESTOURO:=TRUE;
                ULTIMO:=PROX;
              END;
        LINKEST[INDEST -1]:=9999;
        PONTIA:= * + 1;
        PROX:=PROX + 1;
     END;
ITEMCOD[PONTIA]:=CODRED;
ITEMPROX[PONTIA]:=CAT;
TAMEST[INDUE]:=1;
PONTEST[INDUE]:=INDEST;
CODEST[INDEST]:=PONTIA;
LINKEST[INDEST]:=9999;
INDEST:= * + 1;
%
%  ATUALIZACAO DO FOLLOW PARA INCLUIR O EFEITO DE A=>ALFA
%
%        *** INCLUSAO DE FOLLOW(A) EM FOLLOW(XK) ***
%
IF ULTIMOSIMBOLO < 20 AND ULTIMOSIMBOLO NEQ CAT
THEN BEGIN
        AUXPONT:=FOLLOWNT[ULTIMOSIMBOLO];
        ACHOUF:=FALSE;
        WHILE AUXPONT < REDMAX AND NOT ACHOUF
        DO BEGIN
             AUXANT:=AUXPONT;
             AUXPONT:=LINKFOL[AUXPONT];
           END;
        IF NOT ACHOUF
        THEN BEGIN
               INDFOL:=*+1;
               IF INDFOL GEQ NMAXFOL
               THEN BEGIN
                      MSGERRO(203); ESTOURO:=TRUE;
                    END;
               LINKFOL[AUXANT]:=INDFOL;
               LFOLLOW[INDFOL]:=CAT;
               LINKFOL[INDFOL]:=REDMAX;
             END;
      END;
%
%       INCLUSAO DE XI + 1 EM FOLLOW DE XI
%
AUXPONT:=MACROALFA[PTO];
NELEMENTOS:=MACROTA[PTO] + AUXPONT - 2;
FOR I := AUXPONT STEP 1 UNTIL NELEMENTOS
DO BEGIN
      IF VETMACRO[I].C1 < 20
      THEN BEGIN
             AUXFOL:=VETMACRO[I + 1].C1;
             AUX := FOLLOWNT[VETMACRO[I].C1];
             ACHOUF:=FALSE;  TOPOAR:=0;
             VERFOLLOW;
             IF NOT ACHOUF AND NOT ESTOURO
             THEN BEGIN
                    INDFOL:=*+1;
                    IF INDFOL GEQ NMAXFOL
```

```
                    THEN BEGIN
                           MSGERRO(203);
                           ESTOURO:=TRUE;
                         END;
                    LINKFOL[AUXPONTANT]:=INDFOL;
                    LFOLLOW[INDFOL]:=AUXFOL;
                    LINKFOL[INDFOL]:=REDMAX;
                  END;
          END;
   END;
%
%  MONTA MATUNIAO
%
ESTAUX1:=INDESTADO + 1;
WHILE ESTAUX1 LEQ INDUE
DO BEGIN
     J:=1;
     AUX:=PONTEST[ESTAUX1];
     WHILE AUX NEQ 9999
     DO BEGIN
          IF CODEST[AUX] > 1000
          THEN BEGIN
                 IF ITEMCOD[CODEST[AUX]] EQL 5555
                 THEN BEGIN                % REDUZ MACRO
                        AUX2:=FOLLOWNT[CAT];
                        NUMPRODRED:=INDRED + 1;
                        TOPOAR:=0;
                        REDFOL;
                      END
                 ELSE MATUNIAO[ITEMCOD[CODEST[AUX]],J]:=
                                    ITEMPROX[CODEST[AUX]];
               END
          ELSE BEGIN
                 IF PTSHIFT[CODEST[AUX]] EQL 0
                 THEN AUX1:=SHIFTMAX
                 ELSE AUX1:=PTSHIFT[CODEST[AUX]];
                 WHILE AUX1 < SHIFTMAX DO
                 BEGIN
                   MATUNIAO[SIMBENT[AUX1],J]:=ESTPROD[AUX1];
                   AUX1:=LINK[AUX1];
                 END;
                 %  VERIFICA REDUCAO
                 IF PTREDUCAO[CODEST[AUX]] NEQ 0
                 THEN BEGIN
                        NUMPRODRED:=PTREDUCAO[CODEST[AUX]];
                        AUX2:=FOLLOWNT[LESQUERDO
                                       [NUMPRODRED]];
                        TOPOAR:=0;
                        REDFOL;
                      END;
               END;
          J:=J+1;
          AUX:=LINKEST[AUX];
        END;
     %
     %  UNIAO DOS ESTADOS JA MONTANDO OS NOVOS
     %
```

```
N:=TAMEST[ESTAUX1];
PROD:=0;
FOR I:=1 STEP 1 UNTIL COL
DO BEGIN
      JATEM:=RED:=SHIF:=FALSE;
      FOR J:=1 STEP 1 UNTIL N
      DO IF MATUNIAO[I,J] NEQ 0
         THEN IF MATUNIAO[I,J] > 2000
              THEN BEGIN
                   IF SHIF
                   THEN BEGIN
                          MSGERRO(12);
                          MACROERROR;
                        END
                   ELSE IF NOT RED
                        THEN BEGIN
                               RED:=TRUE;
                               PROD:=MATUNIAO[I,J]
                                     - 2000;
                             END
                        ELSE IF MATUNIAO[I,J] = 2000
                                NEQ PROD
                             THEN BEGIN
                                    MSGERRO(13);
                                    MACROERROR;
                                    J:=N;  I:=COL;
                                  END
                   END
              ELSE IF RED
                   THEN BEGIN
                          MSGERRO(12); MACROERROR;
                          J:=N;  I:=COL;
                        END
                   ELSE IF NOT SHIF
                        THEN BEGIN
                             SHIF:=TRUE;
                             CONTSHIFT:=1;
                             IF MATUNIAO[I,J] > 1000
                             THEN BEGIN
                                  JATEM:=TRUE;
                                  ESTH:=ESTAUX1+1;
                                  END
                             ELSE ESTH:=MATUNIAO[I,J];
                             END
                        ELSE IF MATUNIAO[I,J] NEQ ESTH
                             THEN IF JATEM
                                  THEN INCLUINALISTA
                                  ELSE BEGIN
                                       % CRIA LISTA
                                       JATEM:=TRUE;
                                       INDUE:=* + 1;
                                       TAMEST[INDUE]:=2;
                                       INDEST:= * + 1;
                                       PONTEST[INDUE]:=
                                                 INDEST;
                                       CODEST[INDEST]:=
                                                 ESTH;
```

```
                                                    LINKEST[INDEST]
                                                      := INDEST + 1;
                                                    INDEST:=* + 1;
                                                    CODEST[INDEST]:=
                                                       MATUNIAO[I,J];
                                                    LINKEST[INDEST]:=
                                                            9999;
                                                    ESTH:=INDUE;
                                                    END;
        %
        %  MONTA SHIFT COM SIMBOLO "I" SE NECESSARIO.
        %
        IF SHIF
        THEN BEGIN
                IF PTSHIFT[ESTAUX1] EQL 0
                THEN BEGIN
                       INDSHIFT:=* + 1;
                       IF INDSHIFT GEQ NMAXSHIFT
                       THEN ESTOURO:=TRUE;
                       PTSHIFT[ESTAUX1]:=INDSHIFT;
                       COMECOSHIFT:=INDSHIFT;
                     END
                ELSE BEGIN
                       AUX5:=COMECOSHIFT;
                       WHILE AUX5 < SHIFTMAX
                       DO BEGIN
                            AUXANT:=AUX5;
                            AUX5:=LINK[AUX5];
                          END;
                       INDSHIFT:=* + 1;
                       IF INDSHIFT GEQ NMAXSHIFT
                       THEN ESTOURO:=TRUE;
                       LINK[AUXANT]:=INDSHIFT;
                     END;
                IF ESTOURO
                THEN BEGIN
                       MSGERRO(205); I:=COL;
                     END;
                SIMBENT[INDSHIFT]:=I;
                ESTPROD[INDSHIFT]:=ESTH;
                LINK   [INDSHIFT]:=SHIFTMAX;
             END;
    END;
%
%  ATUALIZA REDUCAO NO ESTADO ESTAUX1 (SE NECESS.)
%
IF PROD NEQ 0
THEN BEGIN
       PTREDUCAO[ESTAUX1] := PROD;
       IF PROD > INDRED
       THEN BEGIN
              INDRED:=* + 1;
              IF INDRED GEQ NMAXMACRO
              THEN BEGIN
                     MSGERRO(200);ESTOURO:=TRUE;
                   END;
              LESQUERDO[INDRED]:=CAT;
```

```
                        LDIREITO [INDRED]:=MACROTA[PTD];
                      END;
                END;
            IF ESTOURO
            THEN BEGIN
                   ESTAUX1:=INDUE;ACEITA:=FIMPROG:=TRUE;
                 END;
            ESTAUX1:= * + 1;
            FOR I1:=1 STEP 1 UNTIL COL
            DO FOR J1:=1 STEP 1 UNTIL N
               DO MATUNIAO[I1,J1]:=0;
          END;
        %
        %  INCLUSAO DE SHIFT COM O SIMBOLO INICIAL
        %
        IF NOT ESTOURO AND NOT ERROMACRO
        THEN FOR I:=0 STEP 1 UNTIL INDUE
             DO BEGIN
                  IF PTSHIFT[I] NEQ 0
                  THEN BEGIN
                         AUX:=PTSHIFT[I];
                         TEM:=FALSE;
                         WHILE NOT TEM AND AUX < SHIFTMAX
                         DO IF SIMBENT[AUX] EQL CAT
                            THEN TEM:=TRUE
                            ELSE AUX:=LINK[AUX];
                         IF TEM
                         THEN BEGIN
                                % INSERE SIMBOLO INICIAL DA MACRO
                                % NO FINAL DA LISTA DE SHIFT
                                WHILE AUX < SHIFTMAX
                                DO BEGIN
                                     AUXANT:=AUX;
                                     AUX:=LINK[AUX];
                                   END;
                                INDSHIFT:= * + 1;
                                IF INDSHIFT GEQ NMAXSHIFT
                                THEN BEGIN
                                       MSGERRO(205);I:=INDESTADO;
                                       ESTOURO:=TRUE;
                                     END;
                                LINK[AUXANT]:=INDSHIFT;
                                SIMBENT[INDSHIFT]:=NOMEMACRO;
                                ESTPROD[INDSHIFT]:=INDESTADO + 1;
                                LINK[INDSHIFT]:=SHIFTMAX;
                              END;
                       END;
                END;
          INDESTADO:=INDUE;
          VETINDSHIF[NIVEL]:=INDSHIFT;
          VETINDESTADO[NIVEL]:=INDESTADO;
          VETINDFOL[NIVEL]:=INDFOL;
          VETINDRED[NIVEL]:=INDRED;
        END;
END;     %PROCEDURE EXPANDTAB
%
```

```
% ***************************************************************************
%
PROCEDURE DEFMACRO;
%
%          FUNCAO = ANALISAR A DEFINICAO DA MACRO QUE ESTA SENDO
%                   DECLARADA.
%***************************************************************************
BEGIN
  VARREMACRO;
  IF NOT ERROMACRO THEN
  BEGIN
    CONTME:=1;
    CONTBETA:=0;
    NPARDEF:=0;
    WHILE CONTME NEQ 0 AND NOT ERROMACRO
    DO BEGIN
         SCANNER;
         PTVM:= * + 1;
         IF PTVM GEQ NMAXELEM
         THEN BEGIN
                MSGERRO(202); MACROERROR;
              END;
         CONTBETA:= * +1;
         IF TOKEN[1] EQL CODMACRO
         THEN BEGIN
                CONTME:= * + 1; MACROINT[PTD]:=1;
                VETMACRO[PTVM].C1:=CODMACRO;
              END
         ELSE IF TOKEN[1] EQL CODENDMACRO
              THEN BEGIN
                     CONTME:=* - 1;
                     IF CONTME NEQ 0 THEN
                     VETMACRO[PTVM].C1:=CODENDMACRO;
                   END
              ELSE IF TOKEN[1] EQL CODIDEN
                   THEN IF PARXMACRO[INDICE] EQL 2 AND CONTME = 1
                        THEN BEGIN
                               AUX:=PONTPXM[INDICE];
                               IF PARMACRO[AUX] EQL MACRONOME[PTD]
                               THEN BEGIN
                                      ORDEM:=MACROTA[PTD]+PARORDEM[AUX];
                                      VETMACRO[PTVM].C1:=PARCATEG[AUX];
                                      VETMACRO[PTVM].C2:=ORDEM;
                                      NPARDEF:=* + 1;
                                    END
                               ELSE IF PARMACRO[AUX] = MACROEXT[PTD]
                                    THEN BEGIN
                                           ORDEM:=PARORDEM[AUX];
                                           VETMACRO[PTVM].C1:=
                                                      PARCATEG[AUX];
                                           VETMACRO[PTVM].C2:=5000
                                                  + NODOMACRO[ORDEM];
                                         END
                                    ELSE BEGIN
                                           MSGERRO(10);MACROERROR;
                                         END;
                             END
```

```
                        ELSE BEGIN
                               VETMACRO[PTVM].C1:=CODIDEN;
                               VETMACRO[PTVM].C2:=INDICE;
                             END
                    ELSE IF TOKEN[1] EQL CODDIG
                         THEN BEGIN
                               VETMACRO[PTVM].C1:=TOKEN[1];
                               VETMACRO[PTVM].C2:=TOKEN[2];
                             END
                         ELSE VETMACRO[PTVM].C1:=TOKEN[1];
      END;    %WHILE TOKEN[1]....
    MACROTB[PTD]:=CONTBETA - 1;
    IF NPARDEF < NPAREST AND NOT ERROMACRO
    THEN BEGIN
           MSGERRO(9);
           MACROERROR;
         END;
    IF NOT ERROMACRO
    THEN BEGIN
           VETPTVM[NIVEL]:=PTVM;
           NAOESLR:=FALSE;
           EXPANDTAB;
           IF NAOESLR
           THEN MACROERROR;
         END;
  END;
END;  % PROCEDURE DEFMACRO
%
```

```
%*****************************************************************
%
PROCEDURE VARREMACRO;
%
%          FUNCAO = ANALISAR A ESTRUTURA DA MACRO QUE ESTA SENDO
%                   DECLARADA.
%*****************************************************************
BEGIN
  SCANNER;
  ANTENT:=FALSE;
  IF TOKEN[1] GEQ 20
  THEN BEGIN
         MSGERRO(4);
         MACROERROR;
       END
  ELSE BEGIN
         CAT:=TOKEN[1];
         SCANNER;
         IF TOKEN[1] NEQ CODIDEN
         THEN BEGIN
                MSGERRO(5);
                MACROERROR;
              END
         ELSE BEGIN
                PTD:=* + 1;      COL:=* + 1;
                IF PTD GEQ NMAXMACRO
                THEN BEGIN
                       MSGERRO(200); MACROERROR;
                     END;
                IF COL GEQ NMAXCOL AND NOT ERROMACRO
                THEN BEGIN
                       MSGERRO(216); MACROERROR;
                     END;
                IF NOT LECAR
                THEN BEGIN
                       MACROEXT[PTD]:=MACRONOME[PONTMACRO];
                       MACROINT[PONTMACRO]:=1;
                     END;
                MACRONOME[PTD]:=COL;
                IF PARXMACRO[INDICE] EQL 2
                THEN IF PARMACRO[PONTPXM[INDICE]] EQL MACROEXT[PTD]
                     THEN BEGIN
                            ORDEM:=PARORDEM[PONTPXM[INDICE]];
                            INDICE:=STACKMACRO[ORDEM];
                          END;
                TABSIMB[INDICE,2]:=COL;
                PONTPXM[INDICE]:=PTD;
                PARXMACRO[INDICE]:=1;
                INDMACRO:=INDICE;
                NOMEMACRO:=COL;
                MACROCAT[PTD]:=CAT;
                MACROALFA[PTD]:=PTVM;  NUMELEM:=1;
                VETMACRO[PTVM].C1:=COL;VETMACRO[PTVM].C2:=INDICE;
                SCANNER;
                NPAREST:=0;
                WHILE TOKEN[1] NEQ CODDEFINE AND NOT ERROMACRO
                DO BEGIN
```

```
            NUMELEM:=* + 1;
            PTVM:= * + 1;
            IF PTVM GEQ NMAXELEM
            THEN BEGIN
                   MSGERRO(202); MACROERROR;
                 END;
            IF TOKEN[1] < 20 THEN
            BEGIN
              IF ANTENT
              THEN BEGIN
                     MSGERRO(6);
                     MACROERROR;
                   END
              ELSE BEGIN
                     ANTENT:=TRUE;
                     VETMACRO[PTVM].C1:=TOKEN[1];
                     SCANNER;
                     PONTPAR:=0;
                     IF TOKEN[1] NEQ CODIDEN
                     THEN BEGIN
                            MSGERRO(8); MACROERROR;
                          END
                     ELSE IF PARXMACRO[INDICE] = 2 THEN
                          BEGIN
                            XX:=PONTPXM[INDICE];
                            PONTPAR:=XX;
                            IF PARMACRO[XX] EQL NOMEMACRO
                            AND PARINDMACRO[XX] EQL INDMACRO
                            THEN BEGIN
                                   MSGERRO(7);MACROERROR;
                                 END;
                          END;
              IF NOT ERROMACRO
              THEN BEGIN
                     IF PONTPAR EQL 0
                     THEN BEGIN
                            PDP:= * + 1;
                            IF PDP GEQ NMAXPAR
                            THEN BEGIN
                                   MSGERRO(201);MACROERROR;
                                 END;
                            PONTPAR:= PDP;
                          END;
                     PARXMACRO[INDICE]:=2;
                     PONTPXM[INDICE]:=PONTPAR;
                     PARMACRO[PONTPAR]:=NOMEMACRO;
                     PARINDMACRO[PONTPAR]:=INDMACRO;
                     PARCATEG[PONTPAR]:=VETMACRO[PTVM].C1;
                     PARORDEM[PONTPAR]:=NUMELEM;
                     PARPTM[PONTPAR]:=PTD;
                     NPAREST:= * + 1;
                   END
              END
          END
    ELSE BEGIN
           ANTENT:=FALSE;
           IF TOKEN[1] EQL CODIDEN
```

```
                    THEN BEGIN
                           COL:=COL + 1;PONTPXM[INDICE]:=PTD;
                           IF COL > NMAXCOL
                           THEN BEGIN
                                  MSGERRO(216);MACROERROR;
                                END;
                           PARXMACRO[INDICE]:=3;
                           TABSIMB[INDICE,2]:=COL;
                           VETMACRO[PTVM].C1:=COL;
                           VETMACRO[PTVM].C2:=INDICE;
                         END
                    ELSE VETMACRO[PTVM].C1:=TOKEN[1];
                  END;
            IF NOT ERROMACRO
            THEN SCANNER;
          END;    % WHILE
        IF NOT ERROMACRO
        THEN BEGIN
               MACROTA[PTD]:=NUMELEM;
               VETCOL[NIVEL]:=COL;
               VETPTD[NIVEL]:=PTD;
               VETPOP[NIVEL]:=POP;
               ULTIMOSIMBOLO:=VETMACRO[PTVM].C1;
               MACROBETA[PTD]:=PTVM + 1;
             END;
      END    % IF
   END;    % IF
 END;    % PROCEDURE VARREMACRO
%
```

```
%*****************************************************************************
%
PROCEDURE PESQUISATABSIMB;
%
%          FUNCAO = CALCULAR HASH , VERIFICAR SE UM ID ESTA NA TABSIMB
%                   E SE EH PALAVRA RESERVADA
%*****************************************************************************
BEGIN
  INTEGER CONT,INCREMENTO,AAA,POS;
  REAL ADDR1,INCR1,VALOR;
  DOUBLE MEIO;
  BOOLEAN PROCURA,EIGUAL;
  DEFINE A1 = SAIDASCAN[1].[47:44]#,
         A2 = SAIDASCAN[2].[45:44]#,
         A3 = SAIDASCAN[3].[43:44]#,
         A4 = SAIDASCAN[4].[47:44]#,
         A5 = SAIDASCAN[5].[45:44]#,
         A6 = SAIDASCAN[6].[43:44]#,
         A7 = SAIDASCAN[7].[47:44]#,
         A8 = SAIDASCAN[8].[45:44]#,
         A9 = SAIDASCAN[9].[43:44]#,
         A10 = SAIDASCAN[10].[47:44]#;
  VALOR.[43:44] := REAL(NOT(BOOLEAN(A1) EQV BOOLEAN(A2) EQV
                            BOOLEAN(A3) EQV BOOLEAN(A4) EQV
                            BOOLEAN(A5) EQV BOOLEAN(A6) EQV
                            BOOLEAN(A7) EQV BOOLEAN(A8) EQV
                            BOOLEAN(A9) EQV BOOLEAN(A10)));
  MEIO := VALOR.[42:22] * VALOR.[21:22];
  ADDR1.[9:10] := MEIO.[35:10];
  INCR1.[5:6] := MEIO.[28:6];
  INDICE := INTEGER( ADDR1 MOD NMAXIDEN );
  INCREMENTO := INTEGER((INCR1 MOD 347) + 1);
  CONT := 1;
  PROCURA:=TRUE;
  PALRES:=ESTOURO:=FALSE;
  WHILE PROCURA AND CONT < NMAXIDEN
  DO BEGIN
       IF INDICE > NMAXIDEN
       THEN INDICE:=(INDICE MOD NMAXIDEN);
       IF TABSIMB[INDICE,2] EQL 0
       THEN BEGIN
              PROCURA:=FALSE;
              TAMANHOID[INDICE]:=NCAR;
              POS:=DELTA(PINICIOID,PTID);
              IF POS+NCAR > MAXTAMIDEN
              THEN BEGIN
                     MSGERRO(214);ESTOURO:=TRUE;
                   END
              ELSE BEGIN
                     PTPOCOID[INDICE]:=POS;
                     REPLACE PTID:PTID BY PISS FOR NCAR;
                     JATINHA:=FALSE;
                     CODID:=CODIDEN;
                     TABSIMB[INDICE,2]:=CODIDEN;
                   END;
            END
       ELSE BEGIN
```

```
                IF TAMANHOID[INDICE] EQL NCAR
                THEN BEGIN
                        AAA:=PTPOCOID[INDICE] DIV 6;
                        PTAUX:=POINTER(POCOID[AAA])+(PTPOCOID[INDICE]
                                                        MOD 6 );
                        IF PTAUX EQL PISS FOR NCAR
                        THEN BEGIN
                                IF TABSIMB[INDICE,2] NEQ CODIDEN
                                THEN BEGIN
                                        PALRES:=TRUE;
                                        CODID:=TABSIMB[INDICE,2];
                                     END;
                                PROCURA:=FALSE;
                                JATINHA:=TRUE;
                             END
                        ELSE BEGIN
                                CONT:= * + 1;
                                INDICE:=* + INCREMENTO;
                             END;
                     END
                ELSE BEGIN
                        CONT:=* + 1;
                        INDICE:=* + INCREMENTO;
                    END;
            END;
        END;  % DO WHILE
        IF CONT > NMAXIDEN
        THEN ESTOURO := TRUE;
END;  % PROCEDURE PESQUISATABSIMB.
%
```

```
%*****************************************************************
%
PROCEDURE PEGABETA;
%
%          FUNCAO = DEVOLVER,UM A UM,OS SIMBOLOS DA DEFINICAO
%                   DE UMA MACRO
%*****************************************************************
BEGIN
  CONT:=CONT + 1;
  IF CONT EQL TAMBETA + 1
  THEN TOKEN[1]:=AUXFOLBETA
  ELSE IF CONT > TAMBETA
       THEN BEGIN
              NAOEDER:=TRUE;
              ACEITA:=TRUE;
            END
       ELSE BEGIN
              PTTESTA:= * + 1;
              IF VETMACRO[PTTESTA].C1 EQL CODMACRO
              THEN BEGIN
                     MDE:=0;
                     WHILE VETMACRO[PTTESTA].C1 NEQ CODENDMACRO
                           OR MDE NEQ 1
                     DO BEGIN
                          IF VETMACRO[PTTESTA].C1 EQL CODMACRO
                          THEN MDE:=MDE + 1
                          ELSE IF VETMACRO[PTTESTA].C1 EQL CODENDMACRO
                               THEN MDE:=MDE - 1;
                          CONT:=CONT + 1;
                          PTTESTA:= * + 1;
                        END;
                     TOKEN[1]:=CODENDMACRO;
                   END
              ELSE TOKEN[1]:=VETMACRO[PTTESTA].C1;
            END;
END;  % PROCEDURE PEGABETA
%
```

```
%*****************************************************************
PROCEDURE PESQUISAODIRETORIO;
%
%          FUNCAO = VERIFICAR SE UMA DETERMINADA MACRO ENCONTRA-SE
%                   NO DIRETORIO DE MACROS E  TORNA-LA  DISPONIVEL
%                   AO ANALISADOR.
%*****************************************************************
BEGIN
  DEFINE ERRODCLARQ(X)=BEGIN
                         MSGERRO(X); ERROSINTATICO:=TRUE;
                         WHILE TOKEN[1] NEQ 33 AND TOKEN[1] NEQ 23
                         AND TOKEN[1] NEQ 67 DO SCANNER;
                         IF TOKEN[1] EQL 23 OR TOKEN[1] EQL 67
                         THEN ERROMACROARQ:=TRUE
                         ELSE SCANNER;
                       END#;
  ERROMACROARQ:=FALSE;SCANNER;
  WHILE TOKEN[1] NEQ 23 AND NOT ERROMACROARQ
  DO IF TOKEN[1] NEQ CODIDEN
     THEN ERRODCLARQ(5)
     ELSE BEGIN
            READ(ARQMACRO[0],<2I5>,INIDIR,FIMDIR);
            PROCURA:=TRUE;
            WHILE INIDIR LEQ FIMDIR AND PROCURA
            DO BEGIN
                 PTMACRO:=POINTER(NMACRO[1]);
                 READ(ARQMACRO[INIDIR],<2I5,A60>,IMACRO,FMACRO,
                                                  PTMACRO);
                 IF PTMACRO EQL PISS FOR 60
                 THEN PROCURA:=FALSE
                 ELSE INIDIR:= * + 1;
               END;
            IF PROCURA
            THEN ERRODCLARQ(16)
            ELSE BEGIN
                   FOR I:= 1 STEP 1 UNTIL 14
                   DO CARTAOAUX[I]:=CARTAO[I];
                   PTIAUX:=PTI; PTFAUX:=PTF; LEARQ:=TRUE;
                   NCOLSAUX:=NCOLS; IMACRO:= * - 1;NCOLS:=-1;
                   SCANNER;
                   IF NOT ERROMACROARQ
                   THEN BEGIN
                          FOR I:=1 STEP 1 UNTIL 14
                          DO CARTAO[I]:=CARTAOAUX[I];
                          LEARQ:=FALSE;NCOLS:=NCOLSAUX;
                          PTI:=PTIAUX; PTF:=PTFAUX;
                          SCANNER;
                          IF TOKEN[1] NEQ 33 AND TOKEN[1] NEQ 23
                          THEN ERRODCLARQ(17)
                          ELSE IF TOKEN[1] EQL 33 THEN SCANNER;
                        END;
                 END;
          END;
END OF PROCEDURE;
%
```

```
%*************************************************************************
%
PROCEDURE SCANNER;
%
%          FUNCAO = PERCORRER O FONTE AGRUPANDO OS CARACTERES
%                   ENCONTRADOS EM TOKENS.
%*************************************************************************
BEGIN
  INTEGER I,INDICET;
  BOOLEAN SIMBOLODUPLO;
  TRUTHSET DIGITOS("0123456789.@");
  VALUE ARRAY VETTOK(0,63(0),
                     0,                  % BRANCO
                     9(0),               %
                     40,                 % [
                     20,                 % .
                     61,                 % <
                     31,                 % (
                     42,                 % +
                     2(0),               %
                     9(0),               %
                     41,                 % ]
                     67,                 % $
                     55,                 % *
                     34,                 % )
                     23,                 % ;
                     0,                  %
                     43,                 % -
                     56,                 % /
                     9(0),               %
                     33,                 % ,
                     0,                  % %
                     0,                  % =
                     60,                 % >
                     11(0),              %
                     27,                 % :
                     44,                 % #
                     2(0),               %
                     62,                 % =
                     0,                  %
                     24,                 % := ( POS. 128)
                     64,                 % <=
                     63,                 % <>
                     65,                 % >=
                     45,                 % => ( POS. 132)
                     66,                 % VALOR NUMERICO
                     59);                % IDENTIFICADOR
  VALUE ARRAY CLASSE(0,63(0),            %
                     6,                  % BRANCO
                     9(0),               %
                     3,                  % [
                     3,                  % .
                     4,                  % <
                     3,                  % (
                     3,                  % +
                     11(0),              %
                     3,                  % ]
```

```
                        3,                  % $
                        3,                  % *
                        3,                  % )
                        3,                  % ;
                        0,                  %
                        3,                  % -
                        3,                  % /
                        9(0),               %
                        3,                  % ,
                        5,                  % %
                        0,                  % _
                        4,                  % >
                        11(0),              %
                        4,                  % :
                        3,                  % #
                        2(0),               %
                        4,                  % =
                        66(0),              %
                        9(2),               % A-I
                        7(0),               %
                        9(2),               % J-R
                        8(0),               %
                        8(2),               % S-Z
                        6(0),               %
                        10(1),              % 0-9
                        7(0));              %
PROCEDURE LECARTAO;
BEGIN
  IF LEARQ
  THEN BEGIN
         IMACRO:= * + 1;
         READ(ARQMACRO[IMACRO],<A80>,PTF:=POINTER(CARTAO[1]));
         CONTREG:=* + 1;
         IF LISTAFONTE
         THEN WRITE(SAI,<"ARQMACRO    ",A80>,PTF);
         CARTAO[13]:="       ";NCOLS:=72;
       END
  ELSE IF NOT READ(ENT,<A80>,PTF:=POINTER(CARTAO[1]))
       THEN BEGIN
              CONTREG:= * + 1;
              IF LISTAFONTE
              THEN WRITE(SAI,<T12,A80>,PTF);
              CARTAO[13]:="       ";NCOLS:=72;
            END
       ELSE BEGIN
              TOKEN[1]:=COODOLLAR;CONTINUA:=FALSE;FIMPROG:=TRUE;
              IF DCLMACRO
              THEN BEGIN
                     CONTME:=0; ERROMACRO:=TRUE;
                     MSGERRO(18);MSGERRO(131);ACEITA:=TRUE;
                   END;
            END;
END OF PROCEDURE;
DEFINE TESTA = BEGIN
                 PTF:=PTF + 1;
                 NCOLS:=NCOLS - 1;
               END#;
```

```
DEFINE MONTAARQFOR = BEGIN
                        REPLACE POCOARQFOR[PTAF] BY PTF FOR 1;
                        PTAF := PTAF + 1;
                        IF PTAF GEQ MAXTAMAF
                        THEN BEGIN
                               MSGERRO(213);
                               FIMPROG:=ACEITA:=ESTOURO:=TRUE;
                             END;
                        TAM := TAM + 1;
                        IF FIMPROG
                        THEN ESTOURO:=TRUE
                        ELSE BEGIN
                               PTF := PTF + 1;
                               NCOLS := NCOLS - 1;
                               IF NCOLS LEQ 0
                               THEN LECARTAO;
                             END;
                     END#;
DEFINE MAXNCAR = 64#;
%
%  VERIFICA SE ESPERA-SE ARQUIVO-FORMATO
%
IF QUERARQFOR
THEN BEGIN
       PONTARQFOR[INDAF]:=PTAF;
       TAM:=0;
       NAOABRIU:=TRUE;
       WHILE(NOT (PTF EQL "/" AND NAOABRIU) AND PTF NEQ ">"
                          AND PTF NEQ ";") AND NOT ESTOURO
       DO BEGIN
            IF PTF EQL "<"
            THEN NAOABRIU := FALSE
            ELSE IF PTF EQL """
                 THEN BEGIN
                        MONTAARQFOR;
                        WHILE PTF NEQ """  AND NOT ESTOURO
                        DO MONTAARQFOR;
                      END;
            IF NOT ESTOURO THEN MONTAARQFOR;
          END;
       IF NOT ESTOURO
       THEN IF PTF EQL "/" OR PTF EQL ">"
            THEN MONTAARQFOR;
       TAMARQFOR[INDAF]:=TAM;
       TOKEN[1]:=CODARQFOR;
       TOKEN[2]:=INDAF;
       INDAF:=INDAF + 1;
       IF INDAF > NMAXARQFOR
       THEN BEGIN
              MSGERRO(208); FIMPROG:=ACEITA:=TRUE;
            END;
       QUERARQFOR:=FALSE;
     END
ELSE IF LECAR THEN
     BEGIN
       CONTINUA:=TRUE;
       WHILE CONTINUA DO
```

```
BEGIN
  IF NCOLS LEQ 0
  THEN LECARTAO;
  IF NOT FIMPROG THEN
  BEGIN
    PTI:=PTF;
    PSS:=POINTER(SAIDASCAN[1]);
    TOKEN[2]:=TOKEN[3]:=0;
    CONTINUA:=FALSE;
    CASE CLASSE[REAL(PTI,1)] OF
    BEGIN
    1:% TRATAMENTO DE NUMEROS (DIGITOS)
      PTI:=PTF;
      WHILE (PTF IN DIGITOS) AND NCOLS GEQ 0
      DO BEGIN
           TESTA;
           IF PTF = "+" OR PTF = "-"
           THEN IF (PTF - 1) = "@"
                THEN TESTA;
         END;
      NCAR:=DELTA(PTI,PTF);
      INDTABNUM:= * + 1;
      IF INDTABNUM GEQ NMAXCONST
      THEN BEGIN
             MSGERRO(206); ESTOURO:=TRUE;
           END;
      TAMANUM[INDTABNUM]:=NCAR;
      PONTNUM[INDTABNUM]:=DELTA(PTNI,PTNF);
      IF DELTA(PTNI,PTNF)+NCAR > MAXTAMCONST
      THEN BEGIN
             MSGERRO(212);ESTOURO:=TRUE;
           END
      ELSE REPLACE PTNF:PTNF BY PTI FOR NCAR;
      IF ESTOURO
      THEN BEGIN
             IF DCLMACRO THEN MACROERROR;
             FIMPROG:=ACEITA:=TRUE;
           END;
      TOKEN[1]:= CODDIG ; TOKEN[2]:=INDTABNUM;
      %
    2:% TRATAMENTO DE IDENTIFICADORES
      %
      FOR I := 1 STEP 1 UNTIL 10
      DO SAIDASCAN[I] := "       ";
      PSS:=PISS:=POINTER(SAIDASCAN[1]);
      PTI:=PTF;
      SCAN PTF:PTF FOR NCOLS:NCOLS WHILE IN ALPHA;
      NCAR := DELTA(PTI,PTF);
      IF NCAR > MAXNCAR
      THEN BEGIN
             MSGERRO(1);
             NCAR:=MAXNCAR;
           END;
      REPLACE PSS:PSS BY PTI FOR NCAR;
      CODID:=59;
      PESQUISATABSIMB;
      IF ESTOURO
```

```
THEN BEGIN
       IF NOT ACEITA THEN MSGERRO(207);
       IF DCLMACRO THEN MACROERROR;
       FIMPROG:=ACEITA:=TRUE;
     END
ELSE BEGIN
     TOKEN[1]:=CODID;TOKEN[2]:=INDICE;
     IF PALRES
     THEN IF CODID EQL CODBEGIN OR
             CODID = 26 OR CODID = 50
          THEN BEGIN
                 NIVEL:= * + 1;
                 IF NIVEL GEQ NMAXNIV
                 THEN BEGIN
                        MSGERRO(211);
                        IF DCLMACRO
                        THEN MACROERROR
                        ELSE ACEITA:=TRUE;
                      END;
               END
          ELSE IF CODID = CODEND
               THEN NIVEL:=NIVEL - 1
               ELSE IF NOT ERROMACRO
                    THEN IF CODID = CODMACRO
                         THEN BEGIN
                              IF PESQUIDIR AND
                                           NOT LEARQ
                              THEN LEMACRO := FALSE
                              ELSE LEMACRO := TRUE;
                              IF NOT DCLMACRO AND
                                 LEMACRO
                              THEN BEGIN
                                   NIVELANTERIOR:=
                                             NIVEL-1;
                                   DCLMACRO:=TRUE;
                                   DEFMACRO;
                                   DCLMACRO:=FALSE;
                                   IF ERROMACRO
                                   THEN ERROMACRO:=
                                           FALSE
                                   ELSE TOKEN[1] :=
                                           CODENDMACRO;
                                   SHIFT:=TRUE;
                                   END;
                              END
                         ELSE IF CODID = CODFILE THEN
                              BEGIN
                              PONTARQFOR[INDAF]:=PTAF;
                              TAM:=0;
                              WHILE PTF NEQ ";" AND NOT
                              ESTOURO  DO MONTAARQFOR;
                              TAMARQFOR[INDAF]:=TAM;
                              TOKEN[2]:=INDAF;
                              INDAF:=INDAF + 1;
                              IF INDAF > NMAXARQFOR
                              THEN BEGIN
                                     MSGERRO(208);
```

```
                                        ACEITA:=TRUE;
                                      END;
                                 END
                            % ELSE DO IF CODID = CODFILE
                            ELSE IF CODID EQL CODARQMACRO
                                 THEN IF NOT PESQUIDIR
                                      THEN BEGIN
                                           PESQUIDIR:=TRUE;
                                        PESQUISADIRETORIO;
                                         PESQUIDIR:=FALSE;
                                       IF NOT ERROMACROARQ
                                      THEN CONTINUA:=TRUE;
                                           END;
          END;
  %
3:%  TRATAMENTO DE SIMBOLOS SIMPLES
  %
  TOKEN[1]:=VETTOK[REAL(PTI,1)];
  PTF:=PTF + 1;
  NCOLS := NCOLS - 1;
  %
4:%  TRATAMENTO DE SIMBOLOS DUPLOS
  %  : = < > .  := => <> <= >= ..
  SIMBOLODUPLO:=TRUE;
  INDICET := REAL(PTI,1);
  CASE INDICET
  OF BEGIN
       DP : IF PTF = ":="
            THEN INDICET := 128
            ELSE SIMBOLODUPLO := FALSE;
       IG : IF PTF EQL "=>"
            THEN INDICET := 132
            ELSE SIMBOLODUPLO := FALSE;
       ME : IF PTF EQL "<="
            THEN INDICET := 129
            ELSE IF PTF EQL "<>"
                 THEN INDICET := 130
                 ELSE SIMBOLODUPLO := FALSE;
       MA : IF PTF EQL ">="
            THEN INDICET := 131
            ELSE SIMBOLODUPLO := FALSE;
    ELSE : SIMBOLODUPLO := FALSE;
    END OF CASE;
  TOKEN[1]:=VETTOK[INDICET];
  PTF:=* + 1;
  NCOLS:= * - 1;
  IF SIMBOLODUPLO
  THEN BEGIN
         PTF:=* + 1;
         NCOLS:=* - 1;
       END;
  %
 5:% TRATAMENTO DE COMENTARIOS
  %
  NCOLS:= -1;
  CONTINUA:=TRUE;
  %
```

```
            6:% TRATAMENTO DO CARATER "BRANCO"
              %
              SCAN PTF:PTF FOR NCOLS:NCOLS WHILE EQL " ";
              CONTINUA:=TRUE;
              %
          ELSE:% TRAT.DE CARACTERES NAO PERTENCENTES A LING.
              BEGIN
                IF NOT ERROMACRO
                THEN MSGERRO(3);
                PTF:=PTF + 1;
                NCOLS:=NCOLS - 1;
                CONTINUA:=TRUE;
              END;
            END; %  OF CASE CLASSE
          END; % IF FIMPROG
        END; % WHILE CONTINUA
      END
    ELSE BEGIN     % LE MEMORIA
          TOKEN[2]:=TOKEN[3]:=0;
          IF INDENT EQL 0
          THEN BEGIN
                 LECAR:=TRUE;
                 TOKEN[1]:=GUARDA1;
                 TOKEN[2]:=GUARDA2;
               END
          ELSE BEGIN
                 TOKEN[1]:=ENTSINT[INDENT];
                 IF ENTSINT[INDENT] EQL CODMACRO
                 THEN IF NOT DCLMACRO
                      THEN BEGIN
                             DCLMACRO:=TRUE; INDENT:=* - 1;
                             JATINHA:=FALSE; DEFMACRO;
                             IF NOT ERROMACRO
                             THEN BEGIN
                                    TOKEN[1]:=CODENDMACRO;
                                    INDENT:=* + 1;
                                    SHIFT:=TRUE;
                                  END;
                             DCLMACRO:=FALSE;
                           END
                      ELSE BEGIN
                           END
                 ELSE IF ENTSINT[INDENT] EQL CODIDEN OR
                         ENTSINT[INDENT] EQL CODDIG
                       THEN BEGIN
                              INDICE:=TOKEN[2]:=ENTSEM[INDENT];
                            END
                      ELSE IF ENTSINT[INDENT] < 20   % CODNT
                           THEN PTNODO[TOPO+1]:=ENTSEM[INDENT]
                           ELSE BEGIN
                                  TOKEN[1]:=ENTSINT[INDENT];
                                  INDICE:=TOKEN[2]:=ENTSEM[INDENT];
                                END;
                 INDENT:=* -1;
               END;
        END;
END; % PROCEDURE SCANNER
```

```
%*****************************************************************
%
PROCEDURE MACROEXPANSOR;
%
%          FUNCAO = EXPANDIR MACROS, ISTO E , RESTIRAR A ESTRUTU-
%                   RA DA MACRO DA PILHA SINTATICA E  CRIAR  UMA
%                   ENTRADA COM A DEFINICAO DA REFERIDA MACRO.
%*****************************************************************
BEGIN
  PONTMACRO:=PTREDUCAO[PILHA[TOPO]] - PBASE;
  PTBETA:=MACROBETA[PONTMACRO];
  TAMBETA:=MACROTB[PONTMACRO] - 1;
  FOR I:=PTBETA+TAMBETA STEP -1 UNTIL PTBETA
  DO BEGIN
       INDENT:=* + 1;
        IF INDENT GEQ MAXTOPO
        THEN BEGIN
               MSGERRO(210); I:=PTBETA - 1;
               FIMPROG:=ACEITA:=TRUE;
             END;
       ENTSINT[INDENT]:=VETMACRO[I].C1;
       IF VETMACRO[I].C1 < 20
       THEN BEGIN
              P:=P+1;
              IF VETMACRO[I].C2 > 5000
              THEN AUX:=VETMACRO[I].C2 - 5000
              ELSE AUX:=PTNODO[TOPO - VETMACRO[I].C2];
              IF CODN[AUX] EQL 50
              THEN BEGIN
                     CODN[P]:=50;INFO[P]:=INFO[AUX];
                   END
              ELSE BEGIN
                     CODN[P]:=88;INFO[P]:=AUX;
                   END;
              ENTSEM[INDENT]:=P;
            END
       ELSE ENTSEM [INDENT]:=VETMACRO[I].C2;
     END;  % FOR
  IF MACROINT[PONTMACRO] NEQ 0
  THEN FOR I:=1 STEP 1 UNTIL NDIR
       DO BEGIN
            AUX:=TOPO-NDIR+I;
            PILHA1MACRO[I]:=PILHA1[AUX];
            STACKMACRO [I]:=STACK [AUX];
            NODOMACRO  [I]:=PTNODO[AUX];
          END;
  TOPO:=TOPO-NDIR;
  LECAR:=FALSE;
END;  % PROCEDURE MACROEXPANSOR
%
%    PROCEDURE USADA RECURSIVAMENTE PARA VERIFICAR SE UM DADO
%    SIMBOLO EH FOLLOW DE UM DETERMINADO NAO-TERMINAL
PROCEDURE VERFOLLOW;
BEGIN
  AUXPONTANT:=PILHAAUXR[TOPOAR:=*+1]:=AUX;
  WHILE AUX < REDMAX AND NOT ACHOUF
  DO BEGIN
```

```
        IF LFOLLOW[AUX] < 20
        THEN BEGIN
                AUX:=FOLLOWNT[LFOLLOW[AUX]];
                VERFOLLOW;
                TOPOAR := * - 1;
                AUXPONTANT:=PILHAAUXR[TOPOAR];
                AUX:=LINKFOL[PILHAAUXR[TOPOAR]];
                IF AUX NEQ REDMAX
                THEN AUXPONTANT:=PILHAAUXR[TOPOAR]:=AUX;
             END
        ELSE IF LFOLLOW[AUX] EQL AUXFOL
             THEN ACHOUF:=TRUE
             ELSE BEGIN
                    AUX:=LINKFOL[AUX];
                    IF AUX NEQ REDMAX
                    THEN AUXPONTANT:=PILHAAUXR[TOPOAR]:=AUX;
                  END;
       END;
END; % PROCEDURE VERFOLLOW
%
%     FUNCAO = CRIAR UM NOVO NODO PARA A ARVORE SINTATICA
%
PROCEDURE CRIANO;
BEGIN
  P:=P + 1;
  IF P > NMAXNODO
  THEN BEGIN
         MSGERRO(19);
         ERROSINTATICO:=TRUE;
       END
  ELSE BEGIN
         ARVORE[P,1] := 0;
         ARVORE[P,2] := 0;
       END;
END;
%
%    FUNCAO = VERIFICAR NECESSIDADE DE TRANSFORMACAO DE BLOCO RESULT
%             EM COMANDO COMPOSTOE EFETUAR OS PROCEDIMENTOS NECESSA-
%             RIOS PARA TAL
PROCEDURE VERIFICARESULT;
BEGIN
  FOR I:=PONTRES[NIV] STEP 1 UNTIL TOPONC
  DO BEGIN
       CRIANO;    CODN[P]:=60;
       IF AUX1 EQL AUX
       THEN ESQ[AUX1]:=P
       ELSE DIR[AUX1]:=P;
       ESQ[P]:=ESQ[PILHANC[I]];
       INFO[P]:=I;
       INFO[PILHANC[I]]:=I;
       DIR[ESQ[PILHANC[I]]]:= - P;
       ESQ[PILHANC[I]]:=0;
       CODN[PILHANC[I]]:=68;
       AUX1:=P;
     END;
  IF TOPONC > MAIORTOPONC THEN MAIORTOPONC:=TOPONC;
  TOPONC:= * - QUANTRES[NIV]; QUANTRES[NIV]:=0;
```

```
END;
%
%     FUNCAO =  TRANSFORMACAO DE BLOCOS RESULT EM FUNCOES C/ TIPO
%
PROCEDURE INCLUIFUNCAO;
BEGIN
  PROCEDURE CRIANOFUN;
  BEGIN
    FOR I:=PONTCOND[NIVBEG] STEP 1 UNTIL TNC
    DO BEGIN
         NUMFUN:=* + 1; CRIANO; CODN[P]:=99;
         INFO[P]:=NUMFUN;        AUX2:=P;
         IF AUX1 EQL AUX
         THEN ESQ[AUX1]:=P
         ELSE DIR[AUX1]:=P;
         CRIANO; CODN[P]:=60; ESQ[AUX2]:=P;
         DIR[P]:= - AUX2;  INFO[P]:=NUMFUN;
         ESQ[P]:=ESQ[PILHANC[I]];    CODN[PILHANC[I]]:=98;
         INFO[PILHANC[I]]:=NUMFUN;  DIR[ESQ[PILHANC[I]]]:=-P;
         ESQ[PILHANC[I]]:=0;         AUX1:=AUX2;
       END;
    TNC:= * - QUANTCOND[NIVBEG];
  END;  %OF PROCEDURE CRIANOFUN
  IF CODN[AUX4] EQL LC
  THEN BEGIN
         CRIANO;CODN[P]:=LDC;ESQ[AUX5]:=P;AUX3:=P;
         CRIANO;CODN[P]:=LD; ESQ[AUX3]:=P;AUX:=AUX1:=P;
         CRIANOFUN;
         DIR[AUX]:=AUX4;DIR[AUX4]:= - AUX3;DIR[AUX3]:=-AUX5;
       END
  ELSE BEGIN
         AUX:=ESQ[AUX4];   AUX1:=ESQ[AUX];
         WHILE DIR[AUX1] > 0 DO AUX1:=DIR[AUX1];
         CRIANOFUN;
       END;
  DIR[AUX1]:= - AUX;   INV[AUX]:=AUX1;
END; %OF PROCEDURE INCLUIFUNCAO
```

```
%*****************************************************************
%
PROCEDURE GERAARVORE;
%
%          FUNCAO = CONTRUIR A ARVOPE SINTATICA DO PROGRAMA DE ENTRA-
%                   DA DE ACORDO COM CADA PRODUCAO RECONHECIDA.
%*****************************************************************
BEGIN
  CASE NUMPRODRED
  OF BEGIN
       1 : % PROGR => BLOCO . ( CODNODO = PONTO )
           CRIANO;
           ESQ[P]:=NDTOP1;
           CODN[P]:=PONTO;
           DIR[NDTOP1]:= -P;
           PTNODO[TOPO-1]:= P;
       2 : % BLOCO => BEGIN LISTA END ( CODNODO = BE )
           CRIANO;
           ESQ[P] := NDTOP1;
           CODN[P] := BE;
           DIR[NDTOP1] := - P;
           PTNODO[TOPO - 2] := P;
           IF QUANTCOND[NIVBEG] > 0
           THEN BEGIN
                  AUX5:=P;
                  AUX4:=ESQ[P];INCLUIFUNCAO;QUANTCOND[NIVBEG]:=0;
                END;
           NIVBEG:= * - 1;
       3 : % LISTA => LISTA ; COM    (CODNODO = LC )
           IF CODN[NDTOP2] EQL LDC
           THEN AUX:=DIR[ESQ[NDTOP2]]
           ELSE AUX:=NDTOP2;
           IF CODN[NDTOP] EQL 69
           THEN BEGIN
                 DIR[INV[AUX]]:=ESQ[NDTOP];
                 DIR[INV[NDTOP]]:= - AUX;
                 INV[AUX]:=INV[NDTOP];
                END
           ELSE BEGIN
                 DIR[INV[AUX]]:=NDTOP; DIR[NDTOP]:= - AUX;
                 INV[AUX]:=NDTOP;
                END;
       4 : % LISTA => LDECL ; COM     ( CODNODO = LDC )
           CRIANO;           CODN[P]:=LDC;
           ESQ[P]:=NDTOP2;  AUX:=P;
           IF CODN[NDTOP] EQL 69
           THEN BEGIN
                  CODN[NDTOP]:=LC; AUX1:=NDTOP;
                END
           ELSE BEGIN
                  CRIANO; CODN[P]:=LC;
                  ESQ[P]:=NDTOP; AUX1:=P;
                  INV[P]:=NDTOP; DIR[NDTOP]:= - P;
                END;
           DIR[NDTOP2]:=AUX1; DIR[AUX1]:= - AUX;
           INV[AUX]:= AUX1;   PTNODO[TOPO - 2]:=AUX;
       5 : % LISTA => COM
```

```
      IF CODN[NDTOP] EQL 69
      THEN CODN[NDTOP]:=LC
      ELSE BEGIN
             CRIANO;     CODN[P]:=LC;
             DIR[NDTOP]:= - P; ESQ[P]:=NDTOP;
             INV[P]:=NDTOP;     PTNODO[TOPO]:=P;
           END;
 0 : % S' => PROGR
49 : % TERM => FACT
 9 : % COM => BLOCO
19 : % VAR => IDEN
23 : % ARIT => TERM
25 : % IDEN => IDS
29 : % LEES => EES
30 : % EES => ARIT
42 : % DECL => ENDARQ
57 : % EXPR => ARIT
59 : % LVAR => VAR
62 : % FACT => VAR
63 : % FACT => BLRES
64 : % FACT => CONST
      BEGIN
        % ESTAS PRODUCOES NAO GERAM NOVOS NODOS NEM ALTERAM
        % OS JA CONSTRUIDOS.
      END;
 6 : % COM => VAR := ARIT     ( CODNODO = ATRIB )
      CRIANO;
      CODN[P]:=NUMPRODRED;
      ESQ[P]:=NOTOP2;
      DIR[NDTOP2]:=NDTOP;
      INV[P] := NDTOP;
      DIR[NDTOP]:= - P;
      PTNODO[TOPO - 2] := P;
      IF QUANTRES[NIV] NEQ 0
      THEN BEGIN
             CRIANO;  CODN[P]:=69;
             AUX:=AUX1:=P; VERIFICARESULT;
             DIR[AUX1]:=NDTOP2; DIR[NDTOP2]:= - AUX;
             INV[AUX]:=NDTOP2;  PTNODO[TOPO-2]:=AUX;
           END;
 7 : % COM => GOTO IDEN         ( CODNODO = GOTO )
11 : % COM => RESULT ARIT       ( CODNODO = RES  )
12 : % COM => CALL IDEN         ( CODNODO = CALL )
32 : % DECL => INTEGER LVAR     ( CODNODO = INTEG )
33 : % DECL => REAL LVAR        ( CODNODO = REALL )
36 : % DECL => LABEL IDEN       ( CODNODO = LAB   )
44 : % DECL => NEW IDS          ( CODNODO = NEW   )
      CRIANO;
      CODN[P]:= NUMPRODRED;
       ESQ[P]:=NDTOP;
       INV[P]:=NDTOP;
      DIR[NDTOP]:= - P;
      PTNODO[TOPO - 1] := P;
      IF NUMPRODRED EQL 11 AND QUANTRES[NIV] NEQ 0
      THEN BEGIN
             CRIANO;  CODN[P]:=69;
             AUX:=AUX1:=P; VERIFICARESULT;
```

```
            DIR[AUX1]:=NDTOP1; DIR[NDTOP1]:= - AUX;
            INV[AUX]:=NDTOP1;  PTNODO[TOPO-1]:=AUX;
          END;
 8 : % COM => COND LCOND END   ( CODNODO = CE     )
     CRIANO;
     CODN[P]:=CE;
     ESQ[P]:=NDTOP1;
     INV[P]:=NDTOP1;
     DIR[NDTOP1]:= - P;
     PTNODO[TOPO - 2]:= P;
10 : % COM => IDEN : COM      ( CODNODO = ROT    )
     CRIANO;
     CODN[P]:=NUMPRODRED;
     ESQ[P]:=NDTOP2;
     DIR[NDTOP2]:=NDTOP;
     INV[P] := NDTOP;
     DIR[NDTOP]:= - P;
     PTNODO[TOPO - 2] := P;
13 : % COM => READ ( ARQFOR , LEES )     (CODNODO=LEIA)
14 : % COM => WRITE ( ARQFOR , LEES )    (CODNODO=ESCREVA)
     CRIANO;
     CODN[P]:=ARQFOR;
     DIR[P]:=NDTOP1;
     PTNODO[TOPO - 3]:=P;
     INFO[P]:=STACK[TOPO - 3];
     CRIANO;
     ESQ[P]:=PTNODO[TOPO - 3];
     CODN[P]:=NUMPRODRED;
     DIR[NDTOP1]:= - P;
     INV[P]:=NDTOP1;
     PTNODO[TOPO - 5]:=P;
     IF QUANTRES[NIV] NEQ 0
     THEN BEGIN
            CRIANO;  CODN[P]:=69;
            AUX:=AUX1:=P; VERIFICARESULT;
            DIR[AUX1]:=PTNODO[TOPO-5];
            DIR[PTNODO[TOPO-5]]:= - AUX;
            INV[AUX]:=PTNODO[TOPO-5];
            PTNODO[TOPO-5]:=AUX;
          END;
15 : % COM => WRITE ( ARQFOR )           (CODNODO=WLIT)
     CRIANO;
     CODN[P]:=WLIT;
     INFO[P]:=STACK[TOPO - 1];
     PTNODO[TOPO - 3]:=P;
16 : % COM => E (EPSOLON)                (CODNODO=VAZIO)
     CRIANO;
     CODN[P]:=VAZIO;
     PTNODO[TOPO + 1]:=P;
17 : % LDECL => LDECL ; DECL             (CODNODO=LD)
20 : % LEES  => LEES  ; EES              (CODNODO=LES)
     IF CODN[NDTOP2] EQL NUMPRODRED
     THEN BEGIN
            AUXINV:=INV[NDTOP2];
             DIR[AUXINV]:=NDTOP;
             INV[NDTOP2] := NDTOP;
            DIR[NDTOP]:= - NDTOP2;
```

```
          END
      ELSE BEGIN
             CRIANO;
             ESQ[P]:=NDTOP2;
             DIR[NDTOP2]:=NDTOP;
             INV[P] := NDTOP;
             CODN[P]:=NUMPRODRED;
             DIR[NDTOP]:= - P;
             PTNODO[TOPO - 2]:= P;
           END;
18 : % LDECL => DECL
     CRIANO; ESQ[P]:=NDTOP;
     INV[P]:=NDTOP;  CODN[P]:=LD;
     DIR[NDTOP]:= - P; PTNODO[TOPO]:= P;
20 : % VAR => IDEN [ ARIT ]                    (CODNODO=BRKT)
     CRIANO;
     CODN[P]:=BRKT;
     ESQ[P]:=NDTOP3;
      DIR[NDTOP3]:=NDTOP1;
      INV[P]:=NDTOP1;
     DIR[NDTOP1]:= - P;
     PTNODO[TOPO - 3]:= P;
21 : % ARIT => ARIT + TERMO                    (CODNODO=MAIS)
22 : % ARIT => ARIT - TERMO                    (CODNODO=MENOS)
24 : % IDEN => IDEN # IDS                      (CODNODO=JV)
45 : % TERM => TERM * FACT                     (CODNODO=VEZES)
46 : % TERM => TERM / FACT                     (CODNODO=DIVIS)
47 : % TERM => TERM MOD FACT                   (CODNODO=MOD)
48 : % TERM => TERM DIV FACT                   (CODNODO=DIV)
58 : % LVAR => LVAR , VAR                      (CODNODO=LV)
     IF CODN[NDTOP2] EQL NUMPRODRED
     THEN BEGIN
             AUXINV:=INV[NDTOP2];
             DIR[AUXINV]:=NDTOP;
             INV[NDTOP2]:=NDTOP;
             DIR[NDTOP]:= - NDTOP2;
          END
     ELSE BEGIN
             CRIANO;
             CODN[P]:=NUMPRODRED;
             ESQ[P]:=NDTOP2;
             DIR[NDTOP2]:=NDTOP;
             INV[P]:=NDTOP;
             DIR[NDTOP]:= - P;
             PTNODO[TOPO - 2]:= P;
          END;
26 : % LCOND => EXPR => COM ; LCOND            (CODNODO=VERD)
     %                                         (CODNODO=LCOM)
     IF CODN[NDTOP2] EQL 69
     THEN BEGIN
             CRIANO; CODN[P]:=BE;
             ESQ[P]:=NDTOP2; DIR[NDTOP2]:= -P;
             CODN[NDTOP2]:=LC; PTNODO[TOPO - 2]:=P;
          END;
     CRIANO;
     CODN[P]:=VERD;
     ESQ[P]:=PTNODO[TOPO-4];
```

```
      DIR[PTNODO[TOPO-4]]:=NDTOP2;
      INV[P]:=NDTOP2;
      IF CODN[NDTOP] EQL FALS
      THEN BEGIN
             DIR[NDTOP2]:= - P;
             DIR[P]:=NDTOP;
             AUX:=P;
             CRIANO;
             CODN[P]:=LCON;
             ESQ[P]:=AUX;
             INV[P]:=NDTOP;
             DIR[NDTOP]:= - P;
             PTNODO[TOPO - 4]:= P;
           END
      ELSE BEGIN
             DIR[NDTOP2]:= - P;
             DIR[P]:=ESQ[NDTOP];
             ESQ[NDTOP]:= P;
             PTNODO[TOPO - 4]:=PTNODO[TOPO];
           END;
27 : % LCOND => => COM          ( CODNODO = FALS )
      IF CODN[NDTOP] EQL 69
      THEN BEGIN
             CRIANO; CODN[P]:=BE;
             ESQ[P]:=NDTOP; DIR[NDTOP]:= -P;
             CODN[NDTOP]:=LC; PTNODO[TOPO]:=P;
           END;
      CRIANO;
      CODN[P]:= NUMPRODRED;
      ESQ[P]:=NDTOP;
      INV[P]:=NDTOP;
      DIR[NDTOP]:= - P;
      PTNODO[TOPO - 1] := P;
31 : % EES => FOR IDEN := ARIT STEP ARIT UNTIL ARIT DO EES
      %                                          (CODNODO=FOR)
      CRIANO; CODN[P]:=NFOR; AUX:=P;
      ESQ[P]:=PTNODO[TOPO - 8];
      DIR[PTNODO[TOPO - 8]]:=PTNODO[TOPO - 6];
      CRIANO; CODN[P]:=9; % NODO STEP
      DIR[PTNODO[TOPO - 6]] := P;
      DIR[P]:=PTNODO[TOPO - 4];
      CRIANO;  CODN[P]:=18;   % NODO UNTIL
      DIR[PTNODO[TOPO - 4]]:= P;
      DIR[P]:=NDTOP2;
      CRIANO;  CODN[P]:=19;   % NODO DO
      DIR[NDTOP2]:=P;
      DIR[P]:=NDTOP; INV[AUX]:=NDTOP;
      DIR[NDTOP]:= - AUX;
      PTNODO[TOPO - 9]:=AUX;
34 : % DECL => INTEGER IDEN : ARIT          (CODNODO=CONSTI)
35 : % DECL => REAL IDEN : ARIT             (CODNODO=CONSTR)
37 : % DECL => PROCEDURE IDEN ; BLOCO       (CODNODO=PROC)
38 : % DECL => INTEGER PROCEDURE IDEN ; BLOCO  (CODNODO=FI)
39 : % DECL => REAL PROCEDURE IDEN ; BLOCO     (CODNODO=FR)
      CRIANO;
      CODN[P]:=NUMPRODRED;
      ESQ[P]:=NDTOP2;
```

```
      DIR[NDTOP2]:=NDTOP;
      INV[P]:=NDTOP;
      DIR[NDTOP]:= - P;
      IF NUMPRODRED EQL 38 OR NUMPRODRED EQL 39
      THEN BEGIN
             PTNODO[TOPO - 4]:= P;
             INFO[NDTOP]:= NDTOP2 + 100;
              QUANTRES[NIV]:= * - 1; TOPONC:=*-1;
           END
      ELSE PTNODO[TOPO - 3]:= P;
40 ; % DECL => DECLARE LDECL END                    (CODNODO=DECEND)
      CRIANO;
      CODN[P]:=DECEND;
      ESQ[P]:=NDTOP1;
      DIR[NDTOP1]:= - P;
      PTNODO[TOPO - 2]:= P;
41 ; % DECL => FILE                                 (CODNODO=FILE)
50 ; % IDS  => ID                                   (CODNODO=ID)
67 ; % CONST => K                                   (CODNODO=DIG)
      CRIANO;
      CODN[P]:=NUMPRODRED;
      INFO[P]:=STACK[TOPO];
      PTNODO[TOPO]:=P;
 43 ; % DECL => ENDMACRO
       CRIANO;
       CODN[P] := NUMPRODRED;
        PTNODO[TOPO] := P;
51 ; % EXPR => ARIT > ARIT                         (CODNODO=MA)
52 ; % EXPR => ARIT < ARIT                         (CODNODO=ME)
53 ; % EXPR => ARIT = ARIT                         (CODNODO=IG)
54 ; % EXPR => ARIT <> ARIT                        (CODNODO=MM)
55 ; % EXPR => ARIT <= ARIT                        (CODNODO=MEI)
56 ; % EXPR => ARIT >= ARIT                        (CODNODO=MAI)
      CRIANO;
      CODN[P]:=NUMPRODRED;
      ESQ[P]:=NDTOP2;
      DIR[NDTOP2]:=NDTOP;
     INV[P]:=NDTOP;
      DIR[NDTOP]:= - P;
      PTNODO[TOPO - 2]:= P;
      IF QUANTRES[NIV] > 0
      THEN BEGIN
             IF QUANTCOND[NIVBEG] = 0
             THEN PONTCOND[NIVBEG]:=TNC + 1;
             QUANTCOND[NIVBEG]:= * + QUANTRES[NIV];
             FOR I:=PONTRES[NIV] STEP 1 UNTIL TOPONC
             DO PILHANC[TNC:=*+1]:=PILHANC[I];
             TOPONC:= * - QUANTRES[NIV];QUANTRES[NIV]:=0;
           END;
60 ; % BLRES => BLOCO
      CRIANO;          CODN[P]:= 60;
      DIR[NDTOP]:= -P;ESQ[P]:=NDTOP;
      PTNODO[TOPO]:=P;
      IF QUANTRES[NIV] EQL 0
      THEN PONTRES[NIV]:=TOPONC+1;
      QUANTRES[NIV]:=* + 1;PILHANC[TOPONC:=* + 1]:=P;
61 ; % FACT => ( EXPR )
```

```
            PTNODO[TOPO - 2]:=PTNODO[TOPO - 1];
        65 : % FACT => - FACT                          (CODNODO=MEU)
        66 : % FACT => + FACT                          (CODNODO=MAU)
            CRIANO;
            CODN[P]:=NUMPRODRED;
            ESQ[P]:=NDTOP;
            DIR[NDTOP]:= - P;
            PTNODO[TOPO - 1]:= P;
            STACK[TOPO - 1]:=STACK[TOPO];
    ELSE :  BEGIN
              WRITE(SAI,<"PRODUCAO INVALIDA ***  ">);
            END;
        END OF CASE;
END OF PROCEDURE;
```

```
%*****************************************************************
%
PROCEDURE GERAALGOL;
%
%          FUNCAO = PERCORRER A ARVORE SINTATICA CRIADA GERANDO UM
%                   PROGRAMA NA LINGUAGEM ALGOL , EQUIVALENTE AO
%                   PROGRAMA EXPAND II DE ENTRADA.
%*****************************************************************
BEGIN
  INTEGER DISPO,         % INDICE DE LISTANC
          NPRODGER,      % CODIGO DO NODO A SER GERADK
          NIVELPN,       % TOPO DE PILHANIVEL
          PTPOS,         % POS. INICIAL DO ELEMENTO A SER TRANSFERIDO
          TOPOPR,        % TOPO DE PILHAGP
          TOPORES,       % TOPO DE PILHARES
          TOPOAUX,       % TOPO DE PILHAAUX
          NAUX,NEWCONST,INDNEW,INDBEG,INDITE, % AUXILIARES
          DIF;           % NUM DE POS. DISPONIVEIS
  REAL    ARRAY BUF[0:14],     % USADO NA GERACAO DO ALGOL
                BUFCAR[0:11], % BUFER AUXILIAR
                POCOPR[0:7]; % CONTEM OS OPERADORES
  BOOLEAN NAOACABOUGER,  % CONTROLA GERACAO DO OBJETO
          DECLARANDO,    % INFORMA SE ESTA EM UMA DECLARACAO
          EOPRIMEIRO,    % INDICA SE O IDEN ATUAL E O PRIM. DA DECL
          DECLINT,       % INDICA SE ESTA EM UMA DECLARACAO DE INTE.
          PRIMCOND;      % CONTROLA LISTA DE CONDICOES
  REAL ARRAY PILHANIVEL[1:50], % INFO. DE CONST. A SEREM GERADAS
             LISTANC[1:200],   % NUM. DO NODO RAIZ DE UMA INICIAL.
             PILHAOP[0:50];     % INFO SOBRE OS OP A SEREM GER.
  POINTER PB,PC,PCI,PTP,PBI;    % PONTEIROS USADOS NA TRANSFERENCIA
  INTEGER ARRAY PILHARES[1:50],PILHAAUX[0:50], % AUXILIARES
                PILHABEG[0:64],PILHAITE[0:64],
                CONSTNEW[1:500],VARNEW[1:500],PILHANEW[0:50];
  DEFINE PTATRIB[X] = PILHANIVEL[X].[47:24]#,
         NUMCONST[X] = PILHANIVEL[X].[23:24]#,
         PONTOPR[X] = PILHAOP[X].[47:24]#,
         TAMOPR[X] = PILHAOP[X].[23:08]#,
         PILHAPRI[X]= PILHAOP[X].[15:08]#,
         PILHAPAR[X] =PILHAOP[X].[07:08]#;
  DEFINE GRAVABUF = CONTALGOL:= * + 1;
                    WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
                    REPLACE PBI BY " " FOR 80;
                    PB:=POINTER(BUF) + PILHABEG[INDBEG];
                    DIF:=72 - PILHABEG[INDBEG]#;
  DEFINE TESTATHEN = BEGIN
                       IF CODN[ABS(DIR[PROX])] EQL VERD
                       THEN BEGIN
                              IF PB=1 EQL "," THEN PB:=*-1;
                              CONTALGOL:= * + 1;
                              WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
                              REPLACE PBI BY " " FOR 80;
                              PB:=POINTER(BUF)+PILHAITE[INDITE];
                              DIF:=72 - PILHAITE[INDITE];
                              REPLACE PCI BY "THEN ";GERACAR(5);
                            END;
                     END#;
  DEFINE TESTAPV = BEGIN
```

```
                    PROX:=DIR[ABS(PROX)];
                    IF CODN[ABS(PROX)] NEQ VERD AND
                       CODN[ABS(PROX)] NEQ FALS AND
                       (PROX > 0 OR CODN[ABS(PROX)] NEQ ROT)
                    THEN BEGIN
                           REPLACE PCI BY ";"; GERACAR(1);
                           CONTALGOL:= * + 1;
                           WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
                           REPLACE PBI BY " " FOR 80;
                           PB:=POINTER(BUF)+PILHABEG[INDBEG];
                           DIF:= 72 - PILHABEG[INDBEG];
                         END;
                  END#;
DEFINE GERAOP = BEGIN
                    NAUX:=TAMOPR[TOPOPR];
                    PTP:=POINTER(POCOPR[0])+PONTOPR[TOPOPR];
                    REPLACE PCI BY PTP FOR NAUX;
                    GERACAR(NAUX);
                  END#;
PROCEDURE GERACAR(N);
INTEGER N;
BEGIN
  IF N > DIF THEN BEGIN
                    CONTALGOL:= * + 1;
                    WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
                    PB:=POINTER(BUF)+PILHABEG[INDBEG];
                    DIF:=72-PILHABEG[INDBEG];
                    REPLACE PBI BY " " FOR 80;
                  END;
  REPLACE PB:PB BY PCI FOR N ; DIF := * - N;
END;
PROCEDURE GERAARQ;
BEGIN
  NAUX:=TAMARQFOR[INFO[PROX]];
  PTPOS:=PONTARQFOR[INFO[PROX]];
  WHILE NAUX > 0 DO
    IF NAUX < DIF
    THEN BEGIN
           REPLACE PCI BY POCOARQFOR[PTPOS] FOR NAUX;
           GERACAR(NAUX); NAUX := -1;
         END
    ELSE BEGIN
           REPLACE PCI BY POCOARQFOR[PTPOS] FOR DIF;
           NAUX:=* - DIF; PTPOS:=* + DIF; GERACAR(DIF);
           CONTALGOL:=*+1;      DIF:=72;
           WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);PB:=POINTER(BUF);
           REPLACE PBI BY " " FOR 80;
         END;
END;
PROCEDURE GERAIDEN;
BEGIN
  DEFINE TRANSFEREID = PROCURA:=TRUE;
                       IF PILHANEW[NIVELPN] > 0
                       THEN BEGIN
                              I:=PILHANEW[NIVELPN];
                              WHILE I > 0 AND PROCURA
                              DO IF VARNEW[I] = INFO[PROX]
```

```
                                    THEN PROCURA:=FALSE
                                    ELSE I := I - 1;
                                END;
                            IF PROCURA
                            THEN BEGIN
                                   NAUX:=TAMANHOID[INFO[PROX]];
                                   PTP:=POINTER(POCOID[0]) +
                                        PTPOCOID[INFO[PROX]];
                                   N:=N+NAUX;
                                   IF N < 64
                                   THEN BEGIN
                                          AUXS:=N;
                                          REPLACE PC:PC BY PTP FOR NAUX;
                                        END;
                                END
                            ELSE BEGIN
                                   REPLACE PC:PC BY "XYZ";NAUX:=8;
                                   REPLACE PC:PC BY CONSTNEW[I]
                                   FOR 5 DIGITS;N:=N+8;AUXS:=N;
                                END#;
  PC:=POINTER(BUFCAR);N:=0;
  IF CODN[PROX] EQL 50
  THEN BEGIN
         TRANSFEREID;
         IF NAUX > (72 - PILHABEG[INDBEG])
         THEN BEGIN
                GRAVABUF;PB:=POINTER(BUF)+(70 - NAUX);DIF:=NAUX+1;
              END;
         GERACAR(NAUX);
       END
  ELSE BEGIN
         PROX:=ESQ[PROX];
         WHILE PROX > 0
         DO BEGIN
              IF CODN[PROX] EQL 88
              THEN BEGIN
                     AUX4:=PROX; PROX:=INFO[PROX];
                     DIR[INV[PROX]]:=DIR[AUX4];  PROX:=ESQ[PROX];
                   END;
              TRANSFEREID; PROX:=DIR[PROX];
            END;
         PROX:=ABS(PROX);
         IF AUXS > (72 - PILHABEG[INDBEG])
         THEN BEGIN
                GRAVABUF;PB:=POINTER(BUF)+(70 -AUXS);DIF:=AUXS+1;
              END;
         GERACAR(AUXS);
       END;
END; % OF PROCEDURE GERAIDEN
FILL POCOPR[*] WITH " ,:=+-","*/ MOD","  DIV "," STEP ",
                    " UNTIL","  DO >","<=<><=",">=+-  ";
NAOACABOUGER:=TRUE; DIF:=72;
PBI:=PB:=POINTER(BUF);   PC:=PCI:=POINTER(BUFCAR);
REPLACE PBI BY " " FOR 80;
DISPO:=NIVELPN:=0;  PROX:=P;
TOPOPR:=1; PONTOPR[1]:=1; TAMOPR[1]:=1;
PILHAPRI[1]:=0; PILHAPAR[1]:=0;
```

```
PILHANEW[0]:=NEWCONST:=INDNEW:=0;
DECLARANDO:=EOPRIMEIRO:=DECLINT:=FALSE;
TOPORES:=TOPOAUX:=INDBEG:=INDITE:=0;
PILHABEG[0]:=OFFSET(PB);
%
WHILE NAOACABOUGER
DO BEGIN
     NPRODGER:=CODN[ABS(PROX)];
     IF PROX > 0 THEN
     CASE NPRODGER OF BEGIN
        1:4:17:28:40:58: PROX:=ESQ[PROX];
        2: NUMCONST[NIVELPN := * +1]:=0;
           PILHANEW[NIVELPN]:=PILHANEW[NIVELPN - 1];
           PTATRIB[NIVELPN] := DISPO + 1;
           TESTATHEN;
           REPLACE PCI BY "BEGIN ";GERACAR(6);
           CONTALGOL:=*+1; WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
           REPLACE PBI BY " " FOR 80;
           INDBEG:=* + 1;  AUX:=OFFSET(PB) - 6;
           IF AUX < 50
           THEN PILHABEG[INDBEG]:=AUX + 2
           ELSE PILHABEG[INDBEG]:=50;
           PB:=POINTER(BUF)+PILHABEG[INDBEG];
           DIF:=72 - PILHABEG[INDBEG];
           TOPOAUX:=* + 1;
           IF CODN[ABS(DIR[PROX])] EQL PONTO AND MAIORTOPONC>0
           THEN BEGIN
                  REPLACE PCI BY "REAL ARRAY OJVFTEMP[1:";
                  GERACAR(22); REPLACE PCI BY MAIORTOPONC
                  FOR 2 DIGITS; GERACAR(2);
                  REPLACE PCI BY "];"; GERACAR(2);GRAVABUF;
                END;
           PROX:=ESQ[PROX];
        3: AUX:=PROX;
           IF PILHAAUX[TOPOAUX] NEQ 0
           THEN IF PILHARES[TOPORES] > 1000
                THEN BEGIN
                       REPLACE PCI BY "FUNXYZ";GERACAR(6);
                     REPLACE PCI BY PILHARES[TOPORES] FOR 4 DIGITS;
                       GERACAR(4);REPLACE PCI BY ":=0;";GERACAR(4);
                       GRAVABUF;
                     END
                ELSE IF PILHARES[TOPORES] > 100
                     THEN BEGIN
                            PROX:=PILHARES[TOPORES] - 100;
                            GERAIDEN;
                            REPLACE PCI BY ":=0;"; GERACAR(4);
                            GRAVABUF;
                          END
                     ELSE BEGIN
                            REPLACE PCI BY "OJVFTEMP[";GERACAR(9);
                            REPLACE PCI BY PILHARES[TOPORES]
                            FOR 2 DIGITS; GERACAR(2);
                            REPLACE PCI BY "]:=0;"; GERACAR(5);
                            GRAVABUF;
                          END;
           PROX:=ESQ[AUX];
```

```
 5: BEGIN
      % NADA A FAZER
    END;
 6: TESTATHEN;
    TOPOPR:= * + 1;
    PONTOPR[TOPOPR]:=2;  TAMOPR[TOPOPR]:=2;
   PILHAPRI[TOPOPR]:=0; PILHAPAR[TOPOPR]:=0;
    PROX:=ESQ[PROX];
 7: TESTATHEN;
    REPLACE PCI BY "GO TO "; GERACAR(6);
    PROX:=ESQ[PROX];         GERAIDEN;
    PROX:=DIR[PROX];TESTAPV;
 8: TESTATHEN;
    PROX:=ESQ[PROX];
 9: PROX:=DIR[PROX]; TAMOPR[TOPOPR]:=7;
    PONTOPR[TOPOPR]:=24; PILHAPRI[TOPOPR]:=0;
10: TESTATHEN;
    PROX:=ESQ[PROX];
    GERAIDEN;
    REPLACE PCI BY ":"; GERACAR(1);
    PROX:=DIR[PROX];
43: BEGIN
      PROX:=DIR[PROX];
    END;
11: AUX:=PROX; TESTATHEN;
    IF TOPORES EQL 0
    THEN BEGIN
           REPLACE PCI BY " ERROTEMP:=";GERACAR(11);
         END
    ELSE IF PILHARES[TOPORES] > 1000
         THEN BEGIN
                REPLACE PCI BY "FUNXYZ";GERACAR(6);
              REPLACE PCI BY PILHARES[TOPORES] FOR 4 DIGITS;
                GERACAR(4);REPLACE PCI BY ":=";GERACAR(2);
              END
    ELSE IF PILHARES[TOPORES] > 100
         THEN BEGIN
                PROX:=PILHARES[TOPORES]-100;
                GERAIDEN;
                REPLACE PCI BY ":="; GERACAR(2);
              END
         ELSE BEGIN
                REPLACE PCI BY "OJVFTEMP[";GERACAR(9);
                REPLACE PCI BY PILHARES[TOPORES]
                FOR 2 DIGITS; GERACAR(2);
                REPLACE PCI BY "]:="; GERACAR(3);
              END;
    PROX:=ESQ[AUX];
12: TESTATHEN;
    PROX:=ESQ[PROX];   GERAIDEN;
    REPLACE PCI BY ";"; GERACAR(1); GRAVABUF;
    PROX:=DIR[ABS(DIR[PROX])];
13: TESTATHEN;
    REPLACE PCI BY "READ("; GERACAR(5);
    PROX:=ESQ[PROX];  GERAARQ;
    REPLACE PCI BY ",";       GERACAR(1);
    PROX:=DIR[PROX];
```

```
14: TESTATHEN;
    REPLACE PCI BY "WRITE("; GERACAR(6);
    PROX:=ESQ[PROX];          GERAARQ;
    REPLACE PCI BY ",";       GERACAR(1);
    PROX:=DIR[ABS(PROX)];
15: TESTATHEN;
    REPLACE PCI BY "WRITE("; GERACAR(6);
    GERAARQ;
    REPLACE PCI BY ")";       GERACAR(1);
    TESTAPV;
16: BEGIN
      TESTATHEN;
      PROX:=DIR[PROX];
    END;
18: PONTOPR[TOPOPR]:=31; TAMOPR[TOPOPR]:=4;
    PILHAPRI[TOPOPR]:=0; PROX:=DIR[PROX];
19: TOPOPR:= * - 1;      PROX:=DIR[PROX];
20: PROX:=ESQ[PROX];
    IF DECLARANDO THEN
       IF EOPRIMEIRO
       THEN BEGIN
              REPLACE PCI BY " ARRAY ";
              GERACAR(7);
            END
       ELSE BEGIN
              P0:=*-1;
              REPLACE PCI BY ";";  GERACAR(1);GRAVABUF;
              EOPRIMEIRO:=TRUE;
              IF DECLINT
              THEN BEGIN
                     REPLACE PCI BY "INTEGER ARRAY ";
                     GERACAR(14);
                   END
              ELSE BEGIN
                     REPLACE PCI BY "REAL ARRAY ";
                     GERACAR(11);
                   END;
            END;
     GERAIDEN;
     IF DECLARANDO
     THEN BEGIN
            REPLACE PCI BY "[1:"; GERACAR(3);
          END
     ELSE BEGIN
            REPLACE PCI BY "(";  GERACAR(1);
          END;
     PROX:=DIR[PROX];
21:22:45:46:47:48: % + - * / MOD DIV (DESCENDO)
    PILHAPAR[TOPOPR:=* + 1] := 0;
    TAMOPR[TOPOPR]:=1; PILHAPRI[TOPOPR]:=2;
    CASE CODN[PROX] OF BEGIN
      21: PONTOPR[TOPOPR]:=4;  PILHAPRI[TOPOPR]:=1;
      22: PONTOPR[TOPOPR]:=5;  PILHAPRI[TOPOPR]:=1;
      45: PONTOPR[TOPOPR]:=6;
      46: PONTOPR[TOPOPR]:=7;
      47: PONTOPR[TOPOPR]:=8;  TAMOPR[TOPOPR]:=5;
      48: PONTOPR[TOPOPR]:=13; TAMOPR[TOPOPR]:=5;
```

```
    END OF CASE;
    IF PILHAPRI[TOPOPR] < PILHAPRI[TOPOPR - 1]
    THEN BEGIN
           REPLACE PCI BY "("; GERACAR(1);
           PILHAPAR[TOPOPR]:=1;
         END;
    PROX:=ESQ[PROX];
24:50:IF DECLARANDO AND EOPRIMEIRO
    THEN EOPRIMEIRO:=FALSE;
    GERAIDEN;
    IF DIR[PROX] > 0 THEN GERAOP;
    PROX:=DIR[PROX];
25: PRIMCOND:=TRUE;  PROX:=ESQ[PROX];
26: IF PRIMCOND
    THEN BEGIN
           INDITE:=*+1; AUX:=OFFSET(PB);
           IF AUX < 50
           THEN PILHAITE[INDITE]:=AUX
           ELSE PILHAITE[INDITE]:=50;
           REPLACE PCI BY "IF ";GERACAR(3);
           PRIMCOND:=FALSE;
           PRIMCOND:=FALSE;
         END
    ELSE BEGIN
           CONTALGOL:= * + 1;
           WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
           REPLACE PBI BY " " FOR 80;
           PB:=POINTER(BUF)+PILHAITE[INDITE];
           DIF:=72 - PILHAITE[INDITE];
           REPLACE PCI BY "ELSE IF "; GERACAR(8);
           AUX:=OFFSET(PB) - 3;
           IF AUX < 50
           THEN PILHAITE[INDITE]:=AUX
           ELSE PILHAITE[INDITE]:=50;
         END;
    PROX:=ESQ[PROX];
    IF CODN[PROX] EQL 88
    THEN BEGIN
           DIR[INFO[PROX]]:=DIR[PROX];
           PROX:=INFO[PROX];
         END;
    IF CODN[PROX] < 51 OR CODN[PROX] > 56
    THEN BEGIN
           REPLACE PCI BY "0 NEG ";  GERACAR(6);
         END;
27: PROX:=ESQ[PROX];
    IF NOT PRIMCOND
    THEN BEGIN
           CONTALGOL:= * + 1;
           WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
           REPLACE PBI BY " " FOR 80;
           PB:=POINTER(BUF)+PILHAITE[INDITE];
           DIF:=72 - PILHAITE[INDITE];
           REPLACE PCI BY "ELSE "; GERACAR(5);
           INDITE:=* - 1;
         END;
31: REPLACE PCI BY "FOR "; GERACAR(4);
```

```
    PROX:=ESQ[PROX];        GERAIDEN;
    REPLACE PCI BY ":=";    GERACAR(2);
    PROX:=DIR[PROX];
    PONTOPR[TOPOPR:=*+1]:=18; TAMOPR[TOPOPR]:=6;
    PILHAPRI[TOPOPR]:=0;
32:33: IF CODN[PROX] EQL 32
    THEN BEGIN
           REPLACE PCI BY "INTEGER ";
           GERACAR(8);DECLINT:=TRUE;
         END
    ELSE BEGIN
           REPLACE PCI BY "REAL "; GERACAR(5);
         END;
    DECLARANDO:=EOPRIMEIRO:=TRUE;
    PROX:=ESQ[PROX];
34:35: AUX:=PROX;  PROX:=ESQ[PROX];
    IF NPRODGER EQL 34
    THEN BEGIN
           REPLACE PCI BY "INTEGER "; GERACAR(8);
         END
    ELSE BEGIN
           REPLACE PCI BY "REAL ";    GERACAR(5);
         END;
    GERAIDEN;
    REPLACE PCI BY ";" ;  GERACAR(1);  GRAVABUF;
    CODN[AUX]:=ATRIB;
    PROX:=DIR[AUX];        DIR[AUX]:=0;
    NUMCONST[NIVELPN]:=NUMCONST[NIVELPN] + 1;
    LISTANC[DISPO:= * + 1]:=AUX;
    IF NUMCONST[NIVELPN] > 1
    THEN DIR[LISTANC[DISPO - 1]]:=AUX;
36: REPLACE PCI BY "LABEL "; GERACAR(6);
    PROX:=ESQ[PROX];          GERAIDEN;
    PROX:=DIR[PROX];
37: REPLACE PCI BY "PROCEDURE "; GERACAR(10);
    PROX:=ESQ[PROX];              GERAIDEN;
    REPLACE PCI BY ";";           GERACAR(1);GRAVABUF;
    PROX:=DIR[PROX];
38:39: IF CODN[PROX] EQL 38
    THEN BEGIN
           REPLACE PCI BY "INTEGER PROCEDURE ";
           GERACAR(18);
         END
    ELSE BEGIN
           REPLACE PCI BY "REAL PROCEDURE ";
           GERACAR(15);
         END;
    PROX:=ESQ[PROX];        GERAIDEN;
    REPLACE PCI BY ";";     GERACAR(1);  GRAVABUF;
    PROX:=DIR[PROX];
41: REPLACE PCI BY "FILE "; GERACAR(5);
    GERAARQ; TESTAPV;
44: NEWCONST:= * + 1;INDNEW:= * + 1;
    CONSTNEW[INDNEW]:=NEWCONST;
    VARNEW[INDNEW]:=INFO[ESQ[PROX]];
    PILHANEW[NIVELPN]:=INDNEW;
    PROX:=DIR[PROX];
```

```
  51:52:53: % > < =
      TOPOPR:= * + 1;
      PILHAPRI[TOPOPR]:=1; TAMOPR[TOPOPR]:=1;
      PILHAPAR[TOPOPR]:=0;
      PONTOPR[TOPOPR]:=CODN[PROX] - 16;
      PROX:=ESQ[PROX];
  54:55:56: % >= <= <>
      TOPOPR:=* + 1;
      PILHAPRI[TOPOPR]:=1; PILHAPAR[TOPOPR]:=0;
      TAMOPR[TOPOPR]:=2;
      CASE CODN[PROX] OF BEGIN
        54: PONTOPR[TOPOPR]:=38;
        55: PONTOPR[TOPOPR]:=40;
        56: PONTOPR[TOPOPR]:=42;
      END OF CASE;
      PROX:=ESQ[PROX];
  60: PILHAAUX[TOPOAUX + 1]:= INFO[PROX];
      PILHARES[TOPORES:=* + 1]:=INFO[PROX];
      PROX:=ESQ[PROX];
  65:66:IF CODN[PROX] EQL 65
      THEN REPLACE PCI BY "(-"
      ELSE REPLACE PCI BY "(+";
      GERACAR(2);
      PONTOPR[TOPOPR:=* + 1]:=CODN[PROX] - 21;
      PILHAPRI[TOPOPR]:=3;   PROX:=ESQ[PROX];
  67: NAUX:=TAMANUM[INFO[PROX]];
      PTP:=POINTER(POCONUMEROS[0])+PONTNUM[INFO[PROX]];
      REPLACE PCI BY PTP FOR NAUX; GERACAR(NAUX);
      IF DIR[PROX] > 0 THEN GERAOP;
         PROX:=DIR[PROX];
  68: % GERA TEMPORARIA
                  REPLACE PCI BY " OJVFTEMP[";GERACAR(10);
                  REPLACE PCI BY INFO[PROX] FOR 2 DIGITS;
                  GERACAR(2);
                  REPLACE PCI BY "]"; GERACAR(1);
                  IF DIR[PROX] > 0 THEN GERAOP;
                  PROX:=DIR[PROX];
  69: PROX:=ESQ[PROX];
  88: DIR[INFO[PROX]]:=DIR[PROX];
      PROX:=INFO[PROX];
  98: REPLACE PCI BY "FUNXYZ";GERACAR(6);
      REPLACE PCI BY INFO[PROX] FOR 4 DIGITS;GERACAR(4);
      IF DIR[PROX] > 0 THEN GERAOP;
      PROX:=DIR[PROX];
  99: REPLACE PCI BY "REAL PROCEDURE ";GERACAR(15);
      REPLACE PCI BY "FUNXYZ";          GERACAR(6);
      REPLACE PCI BY INFO[PROX] FOR 4 DIGITS;GERACAR(4);
      PROX:=ESQ[PROX];
      REPLACE PCI BY ";"; GERACAR(1); GRAVABUF;
 ELSE:BEGIN
        WRITE(SAI,<"NODO INVALIDO ",2I4>,PROX,NPRODGER);
        NAOACABOUGER:=FALSE;
      END;
END OF CASE
ELSE
   CASE NPRODGER OF BEGIN
   1: REPLACE PCI BY ".";  GERACAR(1);
```

```
     CONTALGOL:= * + 1;
     WRITE(ARQGER,<A72,K8>,PBI,CONTALGOL);
    REPLACE PBI BY " " FOR 80;
   %WRITE(SAI,<" *** ARQGER = ",A80>,PBI);
    NAOACABOUGER:=FALSE;
 2: DISPO:= DISPO - NUMCONST[NIVELPN];
    NIVELPN:=* - 1;    PROX:=DIR[ABS(PROX)];
    INDNEW:=PILHANEW[NIVELPN];
    PILHAAUX[TOPOAUX]:=0;TOPOAUX := * - 1;
    PB:=POINTER(BUF)+(PILHABEG[INDBEG] - 2);
    DIF:=72 - PILHABEG[INDBEG]; INDBEG:= * - 1;
    REPLACE PCI BY "END"; GERACAR(3);
    IF CODN[ABS(PROX)] NEQ VERD AND CODN[ABS(PROX)] NEQ FALS
                                AND CODN[ABS(PROX)] NEQ PONTO
                                AND CODN[ABS(PROX)] NEQ ROT
    THEN BEGIN
           REPLACE PCI BY ";" ;  GERACAR(1);GRAVABUF;
         END;
 3:4:26:27:28:31:37:38:39:40:58:25:99:
    PROX:=DIR[ABS(PROX)];
 5:7:9:12:15:16:18:19:24:34:35:41:44:50:67:68:88:98:
    BEGIN
      % NADA A FAZER
    END;
 6: TOPOPR:= * - 1;
    TESTAPV;
 8:11:36: TESTAPV;
10: PROX:=DIR[ABS(PROX)];
    IF CODN[ABS(PROX)] NEQ VERD AND CODN[ABS(PROX)] NEQ FALS
    THEN BEGIN
           REPLACE PCI BY ";"; GERACAR(1); GRAVABUF;
         END;
13:14: REPLACE PCI BY ")"; GERACAR(1);
    TESTAPV;
17: IF NUMCONST[NIVELPN] > 0 AND CODN[ABS(DIR[ABS(PROX)])]
       NEQ DECEND
    THEN BEGIN
           DIR[LISTANC[PTATRIB[NIVELPN] + (NUMCONST
           [NIVELPN] - 1)]]:=DIR[ABS(PROX)];
           PROX:=LISTANC[PTATRIB[NIVELPN]];
         END
    ELSE PROX := DIR[ABS(PROX)];
20: REPLACE PCI BY "]";    GERACAR(1);
    IF DIR[ABS(PROX)] > 0 THEN
       IF DECLARANDO
       THEN BEGIN
              REPLACE PCI BY ";"; GERACAR(1);
              EOPRIMEIRO:=TRUE;GRAVABUF;
              IF DECLINT
              THEN BEGIN
                     REPLACE PCI BY "INTEGER ";GERACAR(8);
                   END
              ELSE BEGIN
                     REPLACE PCI BY "REAL ";GERACAR(5);
                   END;
           END
       ELSE GERAOP;
```

```
                PROX:=DIR[ABS(PROX)];
            21:22:45:46:47:48: % + - * / MOD DIV
                IF PILHAPAR[TOPOPR] EQL 1
                THEN BEGIN
                       REPLACE PCI BY ")";  GERACAR(1);
                     END;
                TOPOPR:= * - 1;
                PROX:=DIR[ABS(PROX)];
                IF PROX > 0 THEN GERAOP;
            32:33: DECLARANDO:= DECLINT := FALSE;
                REPLACE PCI BY ";"; GERACAR(1);GRAVABUF;
                PROX:=DIR[ABS(PROX)];
            60: PROX:=DIR[ABS(PROX)];
                TOPORES:= * - 1;
            51:52:53:54:55:56: % < > = <= >= <>
                TOPOPR:= * - 1;
                PROX:=DIR[ABS(PROX)];
            65:66: REPLACE PCI BY ")"; GERACAR(1);
                TOPOPR:= * - 1;         PROX:=DIR[ABS(PROX)];
                IF PROX > 0 THEN GERAOP;
            69: PROX:=DIR[ABS(PROX)];
          ELSE: BEGIN   %DURANTE A FASE DE TESTES **************
                WRITE(SAI,<"NODO INVALIDO - SUB. ",2I4>,PROX,NPRODGER);
                NAOACABOUGER:=FALSE;
                END;
          END OF CASE;
      END OF WHILE;
END OF PROCEDURE GERAALGOL;
%
%    FUNCAO = RESTAURAR UMA MACRO REFERENCIADA ERRONEAMENTE
%
PROCEDURE RESTAURAMACRO;
BEGIN
  IND:=MACROALFA[AUX1];
  WHILE VETMACRO[IND].C1 NEQ PILHA1[TOPO] DO IND:=* + 1;
  WHILE PILHA1[TOPO+1] EQL VETMACRO[IND + 1].C1
  DO BEGIN
          TOPO:=TOPO+1; IND:=IND+1;
        END;
  IF PTSHIFT[PILHA[TOPO]] NEQ 0
  THEN BEGIN
          AUX:=PTSHIFT[PILHA[TOPO]];
          WHILE SIMBENT[AUX] NEQ VETMACRO[IND+1].C1 DO AUX:=LINK[AUX];
          PILHA1[TOPO:=*+1]:=VETMACRO[IND+1].C1;
          STACK[TOPO]:=VETMACRO[IND+1].C2;
          PILHA[TOPO]:=ESTPROD[AUX];
        END
  ELSE BEGIN
          REDUZ:=TRUE; TOPO:=* - MACROTA[AUX1];
          GUARDATOK[1]:=TOKEN[1]; GUARDATOK[2]:=TOKEN[2];
          TOKEN[1]:=MACROCAT[AUX1];TOKEN[2]:=0;
        END;
END OF PROCEDURE;
%
%    FUNCAO = SINCRONIZAR UMA MACRO REFERENCIADA ERRONEAMENTE
%
PROCEDURE SINCRONIZA;
```

```
BEGIN
  AUX2:=MACRONOME[AUX1];IND:=MACROALFA[AUX1];
  PROCURA:=TRUE; AUX4:=TOPO;
  WHILE PILHA1[TOPO] NEQ AUX2 AND TOPO > 1
  DO TOPO:=TOPO - 1;
  IF TOPO EQL 1
  THEN BEGIN
         TOPO:=AUX4;
         WHILE PILHA1[TOPO] NEQ AUX2 AND TOPO > 1
         DO IF PILHA1[TOPO] > 70
            THEN BEGIN
                   AUX:=PONTPXM[STACK[TOPO]];
                   IF PILHA1[TOPO] EQL MACRONOME[AUX]
                   THEN BEGIN
                          AUX2:=MACRONOME[AUX];
                          IND:=MACROALFA[AUX];
                        END
                   ELSE IF PILHA1[TOPO] EQL TOKEN[1]
                        THEN BEGIN
                               AUX4:=TOPO;
                               TOPO:=1;
                             END
                        ELSE TOPO := * - 1
                 END
            ELSE TOPO:= * - 1;
       END;
  IF TOPO EQL 1
  THEN BEGIN
         SCANNER;
         TOPO:=AUX4;
       END
  ELSE BEGIN
         WHILE PILHA1[TOPO] EQL VETMACRO[IND].C1 AND PROCURA
         DO BEGIN
              IF VETMACRO[IND].C1 EQL TOKEN[1]
              THEN PROCURA:=FALSE;
              TOPO:= * + 1; IND:= * + 1;
            END;
         TOPO:=* - 1;
         IF NOT PROCURA
         THEN SCANNER
         ELSE WHILE VETMACRO[IND].C1 NEQ TOKEN[1]
              DO BEGIN
                   AUX5:=PTSHIFT[PILHA[TOPO]];
                   WHILE SIMBENT[AUX5] NEQ VETMACRO[IND].C1
                   DO AUX5:=LINK[AUX5];
                   PILHA1[TOPO:=*+1]:=VETMACRO[IND].C1;
                   STACK[TOPO]:=VETMACRO[IND].C2;
                   PILHA[TOPO]:=ESTPROD[AUX5]; IND:=IND + 1;
                 END;
         END;
END OF PROCEDURE;
```

```
%*****************************************************************
%
PROCEDURE RECUPERAERRO;
%
%          FUNCAO = RECUPERAR O PARSER DE UMA SITUACAO DE ERRO
%                   PERMITINDO A CONTINUIDADE DA ANALISE SINTATICA.
%*****************************************************************
BEGIN
  IF NOT DEUERRO
  THEN BEGIN
         DEUERRO:=TRUE;
         IF PILHA[TOPO] < 148
         THEN MSGERRO(CODERRO[PILHA[TOPO]])
         ELSE MSGERRO(130);
       END;
  WHILE TOKEN[1] NEQ 23 AND TOKEN[1] NEQ 22 AND TOKEN[1] NEQ 30 AND
        TOKEN[1] NEQ 35 AND TOKEN[1] NEQ 67 AND TOKEN[1]  < 71
  DO IF GUARDATOK[1] NEQ 0
     THEN BEGIN
            TOKEN[1]:=GUARDATOK[1];
            GUARDATOK[1]:=0;
          END
     ELSE SCANNER;
  IF TOKEN[1] EQL 67
  THEN BEGIN
         ACEITA:=TRUE;
         MSGERRO(131);
       END
  ELSE BEGIN
         AUX5:=TOPO; PROCURA:=TRUE;
         IF TOKEN[1] EQL 23 OR TOKEN[1] EQL 22
         THEN BEGIN
                WHILE PROCURA
                DO BEGIN
                     IF PILHA[TOPO] < 148
                     THEN IF ESTXNT[PILHA[TOPO], 4] EQL 1 OR
                             ESTXNT[PILHA[TOPO],12] EQL 1
                          THEN PROCURA:=FALSE
                          ELSE BEGIN      END
                     ELSE IF PILHA1[TOPO] > 70
                          THEN PROCURA:=FALSE;
                     IF PROCURA AND TOPO > 1
                     THEN TOPO := TOPO - 1
                     ELSE PROCURA:=FALSE;
                   END;
                IF TOPO EQL 1
                THEN BEGIN
                       TOPO:=AUX5;
                       SCANNER;
                     END
                ELSE IF PILHA1[TOPO] > 70
                     THEN BEGIN
                            AUX1:=PONTPXM[STACK[TOPO]];
                            RESTAURAMACRO;
                          END
                     ELSE SCANNER;
              END
```

```
ELSE IF TOKEN[1] EQL 45 OR TOKEN[1] EQL 30 OR TOKEN[1]=35
     THEN BEGIN
             WHILE PROCURA
             DO BEGIN
                  IF PILHA[TOPO] < 148
                  THEN IF ESTXNT[PILHA[TOPO],4] EQL 1
                       THEN PROCURA:=FALSE
                       ELSE BEGIN END
                  ELSE IF PILHA1[TOPO] > 70
                       THEN PROCURA:=FALSE;
                  IF PROCURA AND TOPO > 1
                  THEN TOPO := TOPO - 1
                  ELSE PROCURA:=FALSE;
                END;
             IF TOPO EQL 1
             THEN BEGIN TOPO:=AUX5;SCANNER; END
             ELSE IF PILHA1[TOPO] > 70
                  THEN BEGIN
                         AUX1:=PONTPXM[STACK[TOPO]];
                         RESTAURAMACRO;
                       END
                   ELSE IF TOKEN[1] = 45 THEN SCANNER;
          END
     ELSE BEGIN
             AUX:=PONTPXM[INDICE];
             IF MACRONOME[AUX] EQL TOKEN[1]
             THEN BEGIN
                   AUX4:=MACROCAT[AUX];
                   WHILE PROCURA
                   DO BEGIN
                   IF PILHA[TOPO] < 148
                   THEN IF ESTXNT[PILHA[TOPO],AUX4] = 1
                        THEN PROCURA:=FALSE
                        ELSE BEGIN  END
                   ELSE IF PILHA1[TOPO] > 70
                        THEN PROCURA:=FALSE;
                  IF PROCURA  AND TOPO > 1
                  THEN TOPO := TOPO - 1
                  ELSE PROCURA:=FALSE;
                     END;
                  IF TOPO EQL 1
                  THEN BEGIN TOPO:=AUX5;SCANNER;END
                  ELSE IF PILHA1[TOPO] > 70
                       THEN BEGIN
                            AUX1:=PONTPXM[STACK[TOPO]];
                            IF MACRONOME[AUX1]=PILHA1[TOPO]
                            THEN IF TOKEN[1]=PILHA1[TOPO]
                                 THEN SCANNER
                                 ELSE BEGIN
                                      RESTAURAMACRO;
                                      IF NOT REDUZ
                                      THEN SCANNER;
                                      END
                            ELSE IF AUX EQL AUX1
                                 THEN SINCRONIZA
                                 ELSE BEGIN
                                        RESTAURAMACRO;
```

```
                                        IF NOT REDUZ
                                        THEN SCANNER;
                                      END;
                          END;
                    END
               ELSE BEGIN % NAO E NOMEMACRO
                      WHILE PILHA1[TOPO] < 70 AND TOPO > 1
                      DO TOPO:=TOPO - 1;
                      IF TOPO EQL 1
                      THEN BEGIN TOPO:=AUX5;SCANNER; END
                      ELSE BEGIN
                             AUX1:=PONTPXM[STACK[TOPO]];
                             IF AUX EQL AUX1
                             THEN IF PILHA1[TOPO] = TOKEN[1]
                                  THEN SCANNER
                                  ELSE SINCRONIZA
                             ELSE BEGIN
                                    RESTAURAMACRO;
                                    IF NOT REDUZ THEN SCANNER;
                                  END;
                           END;
                    END;
             END;
      END;
END OF PROCEDURE;
```

```
%*****************************************************************
%
PROCEDURE PARSER;
%
%          FUNCAO =  EFETUAR ANALISE SINTATICA DO PROGRAMA EXPAND II
%                    DE ENTRADA.
%*****************************************************************
BEGIN
  WHILE NOT ACEITA
  DO BEGIN
       SHIFT:=REDUZ:=FALSE;
       IF PTSHIFT[PILHA[TOPO]] NEQ 0
       THEN BEGIN
              PROX:=PTSHIFT[PILHA[TOPO]];
              WHILE TOKEN[1] NEQ SIMBENT[PROX] AND PROX LEQ INDSHIFT
              DO BEGIN
                   PROX:=LINK[PROX];
                   IF PROX = SHIFTMAX
                   THEN PROX:=INDSHIFT + 1;
                 END;
              IF PROX LEQ INDSHIFT
              THEN BEGIN
                     SHIFT:=TRUE;
                     IF TOKEN[1] EQL CODEND
                     THEN BEGIN
                            NIV:=NIV-1; NIVELD:=NIVELD - 1;
                            IF NIVELD > 0 AND LECAR
                            THEN BEGIN
                                   INDSHIFT:=VETINDSHIF[NIVELD];
                                   INDESTADO:=VETINDESTADO[NIVELD];
                                   INDFOL:=VETINDFOL[NIVELD];
                                   INDRED:=VETINDRED[NIVELD];
                                   IF COL NEQ VETCOL[NIVELD]
                                   THEN BEGIN
                                          COL:=VETCOL[NIVELD];
                                          PTVM:=VETPTVM[NIVELD];
                                          PTD:=VETPTD[NIVELD];
                                          PDP:=VETPDP[NIVELD];
                                          FOR I:=1 STEP 1 UNTIL
                                                        NMAXIDEN
                                          DO IF TABSIMB[I,2] > COL
                                             THEN TABSIMB[I,2]:=0;
                                        END;
                                 END;
                          END;
                   IF TOKEN[1] EQL 21 OR TOKEN[1] EQL 26
                      OR TOKEN[1] EQL 50
                   THEN BEGIN
                          QUANTRES[NIV:=*+1]:=0;PONTRES[NIV]:=0;
                          NIVELD:=* + 1;
                          IF NIVELD > 0 AND LECAR THEN
                          BEGIN
                            VETINDSHIF[NIVELD]:=INDSHIFT;
                            VETINDESTADO[NIVELD]:=INDESTADO;
                            VETINDFOL[NIVELD]:=INDFOL;
                            VETPTVM  [NIVELD]:=PTVM;
                            VETCOL   [NIVELD]:=COL;
```

```
                        VETPTD    [NIVELD]:=PTD;
                        VETPDP    [NIVELD]:=PDP;
                        VETINDRED[NIVELD]:=INDRED;
                      END;
                      IF TOKEN[1] EQL 21 THEN NIVBEG:=*+1;
                    END;
                DEUERRO:=FALSE;
                TOPO := TOPO + 1;
                IF TOPO GEQ MAXTOPO
                THEN BEGIN
                       MSGERRO(210); ACEITA:=TRUE;
                     END;
                PILHA[TOPO]:=ESTPROD[PROX];
                PILHA1[TOPO]:=TOKEN[1];
                IF TOKEN[2] NEQ 0
                THEN STACK[TOPO]:=TOKEN[2];
                %PTNODO[TOPO]:=TOKEN[3];
                IF GUARDATOK[1] EQL 0
                THEN IF TESTANDO
                     THEN PEGABETA
                     ELSE BEGIN
                            IF PILHA[TOPO] =54 OR PILHA[TOPO]=55
                            THEN QUERARQFOR:=TRUE;
                            SCANNER;
                          END
                ELSE BEGIN
                       TOKEN[1]:=GUARDATOK[1];
                       TOKEN[2]:=GUARDATOK[2];
                       GUARDATOK[1]:=GUARDATOK[2]:=0;
                     END;
              END;
     END;
IF NOT SHIFT AND (PTREDUCAO[PILHA[TOPO]] NEQ 0 OR
                  PILHA[TOPO] EQL 1 )
THEN BEGIN
       ACHOUF:=FALSE;
       AUX:=FOLLOWNT[LESQUERDO[PTREDUCAO[PILHA[TOPO]]]];
       TOPOAR:=0;
       AUXFOL:=TOKEN[1];
       VERFOLLOW;
       IF ACHOUF
       THEN BEGIN
              REDUZ:=TRUE; DEUERRO:=FALSE;
              AUX:=PTREDUCAO[PILHA[TOPO]];
              LESQ:=LESQUERDO[AUX];
              NDIR:=LDIREITO[AUX];
              IF TESTANDO
              THEN IF ( TOPO - NDIR) < ( TOPOANTIGO + 1)
                   THEN BEGIN
                          NAOEDER:=TRUE;
                          ACEITA:=TRUE;
                        END;
              IF NOT ACEITA
              THEN BEGIN
                     IF AUX > PBASE AND NOT TESTANDO
                                    AND NOT ERROSINTATICO
                                    AND TRADUZFONTE
```

```
                        THEN BEGIN
                               IF LECAR
                               THEN BEGIN
                                      GUARDA1:=TOKEN[1];
                                      GUARDA2:=TOKEN[2];
                                    END
                               ELSE BEGIN
                                      INDENT:=INDENT+1;
                                      ENTSINT[INDENT]:=TOKEN[1];
                                    END;
                               MACROEXPANSOR;
                               SCANNER;
                             END
                        ELSE BEGIN
                               IF NOT TESTANDO AND NOT ERROSINTATICO
                                  AND TRADUZFONTE
                               THEN BEGIN
                                      NUMPRODRED:=AUX;
                                      GERAARVORE;
                                    END;
                               TOPO:= * - NDIR;
                               IF GUARDATOK[1] EQL 0
                               THEN BEGIN
                                      GUARDATOK[1]:=TOKEN[1];
                                      GUARDATOK[2]:=TOKEN[2];
                                    END;
                               IF LESQ EQL 0
                               THEN TOKEN[1]:=CODDOLLAR
                               ELSE TOKEN[1]:=LESQ;
                               TOKEN[2]:=0;
                               END;
                      END;
             END;
      END;
IF NOT SHIFT AND NOT REDUZ
THEN IF TESTANDO
      THEN IF TOPO EQL (TOPOANTIGO + 1) AND
              PILHA[TOPO] EQL ESTADONT[CAT] AND
              TOKEN[1] EQL CAT
           THEN ACEITA:=TRUE
           ELSE BEGIN
                  ACEITA:=TRUE;
                  NAOEDER:=TRUE;
                END
      ELSE IF TOPO EQL 1 AND PILHA[TOPO] EQL 0 AND
              TOKEN[1] EQL CODDOLLAR
           THEN BEGIN
                  ACEITA:=TRUE;
                  IF NOT ERROSINTATICO  AND TRADUZFONTE
                  THEN BEGIN
                         GERAALGOL;
                       END;
                END
           ELSE BEGIN
                  RECUPERAERRO;
                  ERROSINTATICO:=TRUE;
                END;
```

```
    END; % WHILE NOT ACEITA
END;  % PROCEDURE PARSER
```

```
%
%
%   INCIALZACOES DO PROGRAMA PRINCIPAL
%   ========== == ======== ========
%
%
%   *** TABSIMB ***
%   *** REDUCAO ***
%   *** TABFOLLOW ***
%   *** TABFIRST ***
%   *** NAOTERMINAL ***
%   *** ESTADO ***
%   *** TABSLR ***
%
%   *** POCOID - POCO DE IDENTIFICADORES
 FILL TABSIMB [ 85,*] WITH 4"050001000000",  1,0;
 FILL TABSIMB [337,*] WITH 4"050006000000",  2,0;
 FILL TABSIMB [148,*] WITH 4"050008000000",  3,0;
 FILL TABSIMB [ 10,*] WITH 4"030010000000",  4,0;
 FILL TABSIMB [235,*] WITH 4"050013000000",  5,0;
 FILL TABSIMB [326,*] WITH 4"030018000000",  6,0;
 FILL TABSIMB [156,*] WITH 4"04001B000000",  7,0;
 FILL TABSIMB [179,*] WITH 4"04001F000000",  8,0;
 FILL TABSIMB [ 18,*] WITH 4"050024000000",  9,0;
 FILL TABSIMB [348,*] WITH 4"040028000000",10,0;
 FILL TABSIMB [ 74,*] WITH 4"03002C000000",11,0;
 FILL TABSIMB [262,*] WITH 4"04002F000000",12,0;
 FILL TABSIMB [ 20,*] WITH 4"040033000000",13,0;
 FILL TABSIMB [ 17,*] WITH 4"030037000000",14,0;
 FILL TABSIMB [146,*] WITH 4"04003A000000",15,0;
 FILL TABSIMB [214,*] WITH 4"04003E000000",16,0;
 FILL TABSIMB [226,*] WITH 4"050042000000",17,0;
 FILL TABSIMB [323,*] WITH 4"040047000000",18,0;
 FILL TABSIMB [157,*] WITH 4"05004B000000",19,0;
 FILL TABSIMB [199,*] WITH 4"050050000000",21,0;
 FILL TABSIMB [294,*] WITH 4"030055000000",22,0;
 FILL TABSIMB [324,*] WITH 4"040058000000",25,0;
 FILL TABSIMB [321,*] WITH 4"04005C000000",26,0;
 FILL TABSIMB [128,*] WITH 4"060060000000",28,0;
 FILL TABSIMB [186,*] WITH 4"040066000000",30,0;
 FILL TABSIMB [151,*] WITH 4"06006A000000",32,0;
 FILL TABSIMB [107,*] WITH 4"050070000000",35,0;
 FILL TABSIMB [ 35,*] WITH 4"030075000000",36,0;
 FILL TABSIMB [ 72,*] WITH 4"040078000000",37,0;
 FILL TABSIMB [  5,*] WITH 4"05007C000000",38,0;
 FILL TABSIMB [119,*] WITH 4"020081000000",39,0;
 FILL TABSIMB [ 44,*] WITH 4"070083000000",46,0;
 FILL TABSIMB [ 84,*] WITH 4"04008A000000",47,0;
 FILL TABSIMB [249,*] WITH 4"05008E000000",48,0;
 FILL TABSIMB [ 40,*] WITH 4"090093000000",49,0;
 FILL TABSIMB [138,*] WITH 4"07009C000000",50,0;
 FILL TABSIMB [ 96,*] WITH 4"0400A3000000",51,0;
 FILL TABSIMB [ 87,*] WITH 4"0600A7000000",52,0;
 FILL TABSIMB [265,*] WITH 4"0800AD000000",53,0;
 FILL TABSIMB [117,*] WITH 4"0300B5000000",54,0;
 FILL TABSIMB [122,*] WITH 4"0300B8000000",57,0;
 FILL TABSIMB [ 15,*] WITH 4"0300BB000000",58,0;
```

```
 FILL TABSIMB [177,*] WITH 4"0500BE000000",68,0;
 FILL TABSIMB [215,*] WITH 4"0800C3000000",69,0;
 FILL TABSIMB [330,*] WITH 4"0600CB000000",70,0;
 FILL TABSIMB [ 54,*] WITH 4"0400D1000000",29,0;
 FILL TABSLR [*] WITH  4"000000000000",
4"000100010002" , 4"000200020003" , 4"00150003FFFF" , 4"00140004FFFF",
4"0002000C0006" , 4"000300050007" , 4"000400070008" , 4"000500060009",
4"00060009000A" , 4"0008000D000B" , 4"000C0008000C" , 4"000E0010000D",
4"0019000A000E" , 4"001A0008000F" , 4"001C000E0010" , 4"001D000F0011",
4"001E00100012" , 4"002300110013" , 4"002E00120014" , 4"002F00130015",
4"003000140016" , 4"003100150017" , 4"003200160018" , 4"003300170019",
4"00340018001A" , 4"003500190018" , 4"0036001A001C" , 4"00150003001D",
4"0038001CFFFF" , 4"0016001D001F" , 4"0017001EFFFF" , 4"0017001FFFFF",
4"00180020FFFF" , 4"000800210023" , 4"000E001B0024" , 4"003B001CFFFF",
4"0002002F0026" , 4"000600290027" , 4"000700250028" , 4"0008002E0029",
4"00090022002A" , 4"000D00260028" , 4"000E001B002C" , 4"000F0023002D",
4"0011002A002E" , 4"00120027002F" , 4"0013002B0030" , 4"002D00240031",
4"001F00280032" , 4"002B002C0033" , 4"002A002D0034" , 4"004200300035",
4"001500030036" , 4"003B001CFFFF" , 4"001B00310038" , 4"002800320039",
4"002C0033FFFF" , 4"0002002F0038" , 4"000600290038" , 4"000700340038",
4"0008002E003E" , 4"000D0026003F" , 4"000E001B0040" , 4"0011002A0041",
4"001200270042" , 4"00130028 0043" , 4"001F00280044" , 4"002B002C0045",
4"002A002D0046" , 4"004200300047" , 4"001500030048" , 4"003B001CFFFF",
4"000800350048" , 4"000E001B0048" , 4"003B001CFFFF" , 4"001F0036FFFF",
4"001F0037FFFF" , 4"000600380048" , 4"000800390050" , 4"000E001B0051",
4"001000380052" , 4"0031003A0053" , 4"003B001CFFFF" , 4"000600380055",
4"0008003D0056" , 4"000E001B0057" , 4"0010003C0058" , 4"0031003E0059",
4"0038001CFFFF" , 4"0008003F0058" , 4"000E001B005C" , 4"003B001CFFFF",
4"00080040005E" , 4"000E001B005F" , 4"0038001CFFFF" , 4"000500410061",
4"000C00080062" , 4"002C00120063" , 4"002F00130064" , 4"003000140065",
4"003100150066" , 4"003200160067" , 4"003300170068" , 4"003400180069",
4"00350019006A" , 4"0036001AFFFF" , 4"000E0042006C" , 4"003B001CFFFF",
4"0002000C006E" , 4"000400430060F" , 4"000600090070" , 4"0008000D0071",
4"000E001B0072" , 4"0019000A0073" , 4"001A000B0074" , 4"001C000E0075",
4"001D000F0076" , 4"001E00100077" , 4"002300110070" , 4"00150003 0079",
4"0038001CFFFF" , 4"0002000C0078" , 4"000400440070" , 4"000600090070",
4"0008000D007E" , 4"000C00450070" , 4"000E001B0080" , 4"0019000A0081",
4"001A000B0082" , 4"001C000E0083" , 4"001D000F0084" , 4"001E00100085",
4"002300110086" , 4"002E00120087" , 4"002F00130088" , 4"003000140089",
4"00310015008A" , 4"00320016008B" , 4"00330017008C" , 4"0034001800 8D",
4"00350019008E" , 4"0036001A008F" , 4"00150003 0090" , 4"003B001CFFFF",
4"0002002F0092" , 4"000600290093" , 4"000700460094" , 4"0008002E0095",
4"000D00260096" , 4"000E00180097" , 4"0011002A0098" , 4"0012002700 99",
4"00130028009A" , 4"001F00280098" , 4"002B002C009C" , 4"002A002D009D",
4"00420030009E" , 4"00150003009F" , 4"0038001CFFFF" , 4"002C0033FFFF",
4"00160047FFFF" , 4"002D0048FFFF" , 4"0002000C00A4" , 4"0004004900A5",
4"0006000900A6" , 4"0008000D00A7" , 4"000E001B00A8" , 4"0019000A00A9",
4"001A000B00AA" , 4"001C000E00AB" , 4"001D000F00AC" , 4"001E001000AD",
4"00230011 00AE" , 4"00150003 00AF" , 4"0038001CFFFF" , 4"002A004A00B1",
4"002B004B00B2" , 4"003C004C00B3" , 4"003D004D00B4" , 4"003E004E00B5",
4"003F004F00B6" , 4"00400050 00B7" , 4"00410051FFFF" , 4"0037005200B9",
4"00380053 00BA" , 4"00390054 00BB" , 4"003A0055FFFF" , 4"0002002F00BD",
4"00600290 00BE" , 4"00070025 00BF" , 4"0008002E00C0" , 4"000D002600C1",
4"000E001B00C2" , 4"000F0056 00C3" , 4"0011002A00C4" , 4"0012002700C5",
4"00130028 00C6" , 4"001F0028 00C7" , 4"0028002C00C8" , 4"002A002D00C9",
4"00420030 00CA" , 4"00150003 00CB" , 4"0038001CFFFF" , 4"0002002F00CD",
4"0006002900CE" , 4"0008002E00CF" , 4"000E001B00D0" , 4"0011002A00D1",
```

```
4"00120057000D2" , 4"00130028000D3" , 4"001F002800D4" , 4"0028002C00D5",
4"002A002D00D6" , 4"00420030000D7" , 4"00150003000D8" , 4"003B001CFFFF",
4"0002002F00DA" , 4"00060029000DB" , 4"0008002E00DC" , 4"000E001B00DD",
4"0011002A00DE" , 4"00120058000DF" , 4"00130028000E0" , 4"001F002800E1",
4"0028002C00E2" , 4"002A002D00E3" , 4"00420030000E4" , 4"00150003000E5",
4"003B001CFFFF" , 4"00280032000E7" , 4"002C0033FFFF" , 4"0002000C00E9",
4"0004005900EA" , 4"00060009000EB" , 4"0008000D00EC" , 4"000E001B00ED",
4"0019000A00EE" , 4"001A000B00EF" , 4"001C000E00F0" , 4"001D000F00F1",
4"001E001000F2" , 4"00230011000F3" , 4"00150003000F4" , 4"003B001CFFFF",
4"0002002F00F6" , 4"00060029000F7" , 4"0007005A00F8" , 4"0008002E00F9",
4"000D002600FA" , 4"000E001B00FB" , 4"0011002A00FC" , 4"00120027000FD",
4"00130028000FE" , 4"001F002800FF" , 4"0028002C0100" , 4"002A002D0101",
4"0042003001002" , 4"0015000301003" , 4"003B001CFFFF" , 4"000E00580105",
4"003B001CFFFF" , 4"002A004A0107" , 4"0028004BFFFF" , 4"002C0033FFFF",
4"0020005CFFFF" , 4"0020005DFFFF" , 4"0021005EFFFF" , 4"00280032010D",
4"002C0033010E" , 4"001B005FFFFF" , 4"0080060001110" , 4"000E001B0111",
4"003B001CFFFF" , 4"0021005EFFFF" , 4"0028003201114" , 4"002C00330115",
4"001B0061FFFF" , 4"0080062011117" , 4"000E001B0116" , 4"003B001CFFFF",
4"002C0033FFFF" , 4"002C00330118" , 4"00170063FFFF" , 4"0017006401110",
4"0016006SFFFF" , 4"002A004A011F" , 4"0028004BFFFF" , 4"0002000C0121",
4"0040066012122" , 4"0060009012123" , 4"0008000D0124" , 4"000E001B0125",
4"0019000A0126" , 4"001A000B0127" , 4"001C000E0128" , 4"001D000F0129",
4"001E001001 2A" , 4"0023001101 2B" , 4"0015000301 2C" , 4"003B001CFFFF",
4"0002002F012E" , 4"0060029012F" , 4"0008002E0130" , 4"000D00670131",
4"000E001B0132" , 4"0011002A0133" , 4"0012002701 34" , 4"0013002801 36",
4"0000000000000" , 4"001F00280137" , 4"0028002C0138" , 4"002A002D0139",
4"0042003001 3A" , 4"0015000301 38" , 4"003B001CFFFF" , 4"0002002F013D",
4"0060029013E" , 4"0008002E013F" , 4"000D0068 0140" , 4"000E001B0141",
4"0011002A0142" , 4"0012002701 43" , 4"0013002B0144" , 4"001F00280145",
4"0028002C0146" , 4"002A002D0147" , 4"0042003001 48" , 4"0015000301 49",
4"003B001CFFFF" , 4"0002002F014B" , 4"0060029014C" , 4"0070069014D",
4"0008002E014E" , 4"000D0026014F" , 4"000E001B0150" , 4"0011002A0151",
4"0012002701 52" , 4"0013002B0153" , 4"001F00280154" , 4"002B002C0155",
4"002A002D0156" , 4"0042003001 57" , 4"0015000301 58" , 4"0036001CFFFF",
4"0002002F015A" , 4"0060029015B" , 4"0007006A015C" , 4"0008002E015D",
4"000D0026015E" , 4"000E001B015F" , 4"0011002A0160" , 4"0012002701 61",
4"0013002B0162" , 4"001F00280163" , 4"0028002C0164" , 4"002A002D0165",
4"0042003001 66" , 4"0015000301 67" , 4"003B001CFFFF" , 4"0002002F0169",
4"0060029016A" , 4"0007006B0168" , 4"0008002E016C" , 4"000D0026016D",
4"000E001B016E" , 4"0011002A016F" , 4"0012002701 70" , 4"0013002B0171",
4"001F00280172" , 4"002B002C0173" , 4"002A002D0174" , 4"0042003001 75",
4"0015000301 76" , 4"003B001CFFFF" , 4"0002002F0178" , 4"0060029017 9",
4"0007006C017A" , 4"0008002E0178" , 4"000D0026017C" , 4"000E001B017D",
4"0011002A017E" , 4"0012002701 7F" , 4"0013002B0180" , 4"001F00280181",
4"0028002C0182" , 4"002A002D0183" , 4"0042003001 84" , 4"0015000301 85",
4"003B001CFFFF" , 4"0002002F0187" , 4"0060029018B" , 4"0007006D0189",
4"0008002E018A" , 4"000D0026018B" , 4"000E001B018C" , 4"0011002A018D",
4"0012002701 8E" , 4"0013002B018F" , 4"001F00280190" , 4"002B002C0191",
4"002A002D0192" , 4"0042003001 93" , 4"0015000301 94" , 4"003B001CFFFF",
4"0002002F0196" , 4"0060029019 7" , 4"0007006E0198" , 4"0008002E0199",
4"000D0026019A" , 4"000E001B019B" , 4"0011002A019C" , 4"0012002701 9D",
4"0013002B019E" , 4"001F00280 19F" , 4"0028002C01A0" , 4"002A002D01A1",
4"0042003001 A2" , 4"0015000301 A3" , 4"003B001CFFFF" , 4"0002002F01A5",
4"0060029019 A6" , 4"0008002E01A7" , 4"000E001B01A8" , 4"0011002A01A9",
4"0012006F01AA" , 4"0013002B01AB" , 4"001F00280 1AC" , 4"002B002C01AD",
4"002A002D01AE" , 4"0042003001 AF" , 4"0015000301B0" , 4"003B001CFFFF",
4"0002002F01B2" , 4"0060029019 3" , 4"0008002E01B4" , 4"000E001B01B5",
```

```
4"0011002A01B6" , 4"0012007001B7" , 4"0013002B01B8" , 4"001F002801B9",
4"002B002C01BA" , 4"002A002D01BB" , 4"00420030001BC" , 4"001500030018D",
4"0038001CFFFF" , 4"0002002F01BF" , 4"0006002901C0" , 4"0008002E01C1",
4"000E001B01C2" , 4"0011002A01C3" , 4"0012007101C4" , 4"0013002B01C5",
4"001F002801C6" , 4"002B002C01C7" , 4"002A002D01C8" , 4"00420030001C9",
4"0015000301CA" , 4"0038001CFFFF" , 4"0002002F01CC" , 4"0006002901CD",
4"0008002E01CE" , 4"000E001B01CF" , 4"0011001801D0" , 4"0012007201D1",
4"0013002B01D2" , 4"001F002801D3" , 4"002B002C01D4" , 4"002A002D01D5",
4"00420030001D6" , 4"0015000301D7" , 4"0038001CFFFF" , 4"00220073FFFF",
4"0029007401DA" , 4"002A004A01DB" , 4"002B004BFFFF" , 4"00210075FFFF",
4"0021007601DE" , 4"00220077FFFF" , 4"0006007801E0" , 4"0008002E01E1",
4"000E001B01E2" , 4"0038001CFFFF" , 4"0002002F01E4" , 4"0006002901E5",
4"0007007901E6" , 4"0008002E01E7" , 4"000D002601E8" , 4"000E001B01E9",
4"0011002A01EA" , 4"0012002701EB" , 4"0013002B01EC" , 4"001F002801ED",
4"002B002C01EE" , 4"002A002D01EF" , 4"00420030001F0" , 4"0015000301F1",
4"0038001CFFFF" , 4"002C003301F3" , 4"0017007AFFFF" , 4"0002002F01F5",
4"0006002901F6" , 4"0007007B01F7" , 4"0008002E01F8" , 4"000D002601F9",
4"000E001B01FA" , 4"0011002A01FB" , 4"0012002701FC" , 4"0013002B01FD",
4"001F002B01FE" , 4"002B002C01FF" , 4"002A002D0200" , 4"004200300201",
4"0015000302O2" , 4"0038001CFFFF" , 4"002C00330204" , 4"0017007CFFFF",
4"0020007D0206" , 4"00150003FFFF" , 4"000C00450208" , 4"002E00120209",
4"002F0013020A" , 4"003000140208" , 4"00310015020C" , 4"00320016020D",
4"0033001702OE" , 4"0034001B020F" , 4"003500190210" , 4"0036001AFFFF",
4"0017007EFFFF" , 4"00370052O213" , 4"00380053O214" , 4"00390054O215",
4"003A0055FFFF" , 4"00370052O217" , 4"00380053O218" , 4"00390054O219",
4"003A0055FFFF" , 4"002A004A021B" , 4"002B004BFFFF" , 4"002A004A021D",
4"002B004BFFFF" , 4"002A004A021F" , 4"002B004BFFFF" , 4"002A004A0221",
4"002B004BFFFF" , 4"002A004A0223" , 4"002B004BFFFF" , 4"002A004A0225",
4"002B004BFFFF" , 4"0002002F0227" , 4"000600290228" , 4"000700810229",
4"0008002E022A" , 4"000A007F022B" , 4"000B00800022C" , 4"000D0026022D",
4"000E001B022E" , 4"0011002A022F" , 4"00120027O230" , 4"0013002B0231",
4"0024008202O2" , 4"001F002B0233" , 4"002B002C0234" , 4"002A002D0235",
4"0042003002O6" , 4"0015000302O7" , 4"0038001CFFFF" , 4"0002002F0239",
4"000600290O3A" , 4"000700810O3B" , 4"0008002E023C" , 4"000A0083023D",
4"000B00800O3E" , 4"000D0026023F" , 4"000E001B0240" , 4"0011002A0241",
4"0012002702O2" , 4"0013002B0243" , 4"0024008202O4" , 4"001F002B0245",
4"002B002C0246" , 4"002A002D0247" , 4"0042003002O8" , 4"0015000302O9",
4"003B001CFFFF" , 4"002A004A024B" , 4"002B004BFFFF" , 4"0002002F024D",
4"0011008402O4E" , 4"0015000302O4F" , 4"002A004A0250" , 4"002B004BFFFF",
4"0002002F0252" , 4"0011008502O53" , 4"0015000302O54" , 4"0002002F0255",
4"000600290256" , 4"000700250257" , 4"0008002E0258" , 4"000900860259",
4"000D0026025A" , 4"000E001B025B" , 4"000F0023025C" , 4"0011002A025D",
4"0012002702O5E" , 4"0013002B025F" , 4"002D00240260" , 4"001F002B0261",
4"002B002C0262" , 4"002A002D0263" , 4"0042003002O64" , 4"0015000302O65",
4"003B001CFFFF" , 4"0022008702O67" , 4"0021008BFFFF" , 4"002A004A0269",
4"002B004BFFFF" , 4"000B008902O6B" , 4"000E001B026C" , 4"003B001CFFFF",
4"0022008A026E" , 4"0021008BFFFF" , 4"0002002F0270" , 4"000600290271",
4"000700810272" , 4"0008002E0273" , 4"000B008B0274" , 4"000D0026027S",
4"000E001B0276" , 4"0011002A0277" , 4"0012002702O78" , 4"0013002B0279",
4"0024008202O7A" , 4"001F002B027B" , 4"002B002C027C" , 4"002A002D027D",
4"0042003002O7E" , 4"0015000302O7F" , 4"003B001CFFFF" , 4"002C00330281",
4"0018008CFFFF" , 4"0002002F0283" , 4"0006002902O84" , 4"0007008D0285",
4"0008002E0286" , 4"000D002602O87" , 4"000E001B0288" , 4"0011002A0289",
4"0012002702O8A" , 4"0013002B028B" , 4"001F002B028C" , 4"002B002C028D",
4"002A002D028E" , 4"0042003002O8F" , 4"0015000302O90" , 4"003B001CFFFF",
4"002A004A0292" , 4"002B004B0293" , 4"0025008EFFFF" , 4"0002002F0295",
4"0006002902O96" , 4"0007008F0297" , 4"0008002E0298" , 4"000D002602O99",
```

```
4"000E0018029A" , 4"0011002A0298" , 4"00120027029C" , 4"0013002B029D",
4"001F0028029E" , 4"002B002C029F" , 4"002A002D02A0" , 4"00420030002A1",
4"0015000302A2" , 4"003B001CFFFF" , 4"002A004A02A4" , 4"002B004802A5",
4"00260090FFFF" , 4"0002002F02A7" , 4"00060029002A8" , 4"0007009102A9",
4"0008002E02AA" , 4"000D00260002AB" , 4"000E001B02AC" , 4"0011002A02AD",
4"00120027002AE" , 4"0013002B02AF" , 4"001F002802B0" , 4"002B002C02B1",
4"002A002D02B2" , 4"00420030002B3" , 4"0015000302B4" , 4"003B001CFFFF",
4"002A004A02B6" , 4"002B004B02B7" , 4"00270092FFFF" , 4"0002002F02B9",
4"00060029002BA" , 4"0007008102BB" , 4"0008002E02BC" , 4"000B009302BD",
4"000D002602BE" , 4"000E001B02BF" , 4"0011002A02C0" , 4"00120027002C1",
4"0013002B02C2" , 4"0024008202C3" , 4"001F002802C4" , 4"002B002C02C5",
4"002A002D02C6" , 4"00420030002C7" , 4"0015000302C8" , 4"003B001CFFFF",
4"0002000202CA" , 4"00150003FFFF" , 4"00150003FFFF" , 4"0002000C02CD",
4"0003009702CE" , 4"0004000702CF" , 4"0005000602D0" , 4"0006000902D1",
4"0008000D02D2" , 4"000C000802D3" , 4"000E001B02D4" , 4"0019000A02D5",
4"001A000B02D6" , 4"001C000E02D7" , 4"001D000F02D8" , 4"001E001002D9",
4"0023001102DA" , 4"002E001202DB" , 4"002F001302DC" , 4"0030001402DD",
4"0031001502DE" , 4"0032001602DF" , 4"0033001702E0" , 4"0034001802E1",
4"0035001902E2" , 4"0036001A02E3" , 4"0015000302E4" , 4"003B001A02E5",
4"0017001EFFFF" , 4"0002000C02E7" , 4"0006000902E8" , 4"0008000D02E9",
4"000E001B02EA" , 4"0019000A02EB" , 4"001A000B02EC" , 4"001C000E02ED",
4"001D000F02EE" , 4"001E001002EF" , 4"0023001102F0" , 4"0015000302F1",
4"0038001CFFFF" , 4"0005009A02F3" , 4"000C000802F4" , 4"002E001202F5",
4"002F001302F6" , 4"0030001402F7" , 4"0031001502F8" , 4"0032001602F9",
4"0033001702FA" , 4"0034001802FB" , 4"0035001902FC" , 4"0036001AFFFF",
4"00170064FFFF" , 4"0008002E02FF" , 4"000E001B0300" , 4"003B001CFFFF",
4"0002002F0302" , 4"0006002903003" , 4"0007009D0304" , 4"0008002E0305",
4"000D00260306" , 4"000E001B0307" , 4"0011002A0308" , 4"0012002703009",
4"0013002B030A" , 4"001500030308" , 4"001F0028030C" , 4"002A002D030D",
4"002B002C030E" , 4"003B001C030F" , 4"00420030FFFF" , 4"002A004A0311",
4"0028004BFFFF" , 4"0008009F0313" , 4"000E001B0314" , 4"003B001CFFFF",
4"002C0033FFFF" , 4"0002002F0317" , 4"0006002903018" , 4"0007002503019",
4"0008002E031A" , 4"000D00260031B" , 4"000E001B031C" , 4"000F0023031D",
4"0011002A031E" , 4"00120027031F" , 4"0013002B0320" , 4"0015000303021",
4"001F00280322" , 4"002A002D0323" , 4"002B002C0324" , 4"002D00240325",
4"003B001C0326" , 4"00420030FFFF" , 4"0002002F0328" , 4"0006002903029",
4"00070081032A" , 4"0008002E032B" , 4"000A00A2032C" , 4"000B00800032D",
4"000D0026032E" , 4"000E001B032F" , 4"0011002A0330" , 4"0012002703031",
4"0013002B0332" , 4"0015000303033" , 4"001F0028033A" , 4"0024008203035",
4"002A002D0336" , 4"002B002C0337" , 4"003B001C0338" , 4"00420030FFFF",
4"00210088FFFF" , 4"0002002F033B" , 4"0006002903003C" , 4"0007008103003D",
4"0008002E033E" , 4"000D0026033F" , 4"000E001B0340" , 4"0011002A0341",
4"0012002703042" , 4"0013002B0343" , 4"0015000303044" , 4"001F0028034E",
4"0024008203046" , 4"002A002D0347" , 4"002B002C0348" , 4"003B001C0349",
4"00420030FFFF" , 4"002E0012034B" , 4"002F0013034C" , 4"0030001403040",
4"0031001503045" , 4"0032001603050" , 4"0033003703051" , 4"0034001803052",
4"0035001903052" , 4"0036001AFFFF" , 4"0002002F0354" , 4"0006002903055",
4"0008002E0356" , 4"000D00A60357" , 4"000E001B0358" , 4"0011002A0359",
4"0012002703054" , 4"0013002B035B" , 4"0015000303054" , 4"001F0028035D",
4"002A002D035E" , 4"002B002C035F" , 4"003B001C0360" , 4"00420030FFFF",
4"0037005203062" , 4"0038005303063" , 4"0039005403064" , 4"003A0055FFFF",
4"003B001CFFFF" , 4"0002002F0367" , 4"0006002903068" , 4"0007002503069",
4"0008002E036A" , 4"000D0026036B" , 4"000E001B036C" , 4"0011002A036D",
4"0012002703062" , 4"0013002B036F" , 4"0015000303070" , 4"001F0028037E",
4"002A002D0372" , 4"002B002C0373" , 4"003B001C0374" , 4"00420030FFFF",
4"0060030B0376" , 4"0008002E0377" , 4"000E001B0378" , 4"001000AA0379",
4"003B001CFFFF" , 4"0021005EFFFF" , 4"0002002F037C" , 4"0015000 3FFFF",
```

```
4"0002002F037E" , 4"000600290037F" , 4"0008002E0380" , 4"000E001B0381",
4"0011002A0382" , 4"0013002B0383" , 4"0015000300384" , 4"001F00280385",
4"002A002D0386" , 4"0035002C0387" , 4"003B001C0388" , 4"00420030FFFF",
4"00420030FFFF";
 FILL NAOTERMINAL [*] WITH
                                4"000000000001",                  % 0
                                4"009400010002",                  % 1
                                4"009500020003",                  % 2
                                4"009600030006",                  % 3
                                4"009800060008",                  % 4
                                4"0099000F0009",                  % 5
                                4"009B00100008",                  % 6
                                4"009C0011000E",                  % 7
                                4"009E0012001E",                  % 8
                                4"00A000130022",                  % 9
                                4"00A100150023",                  % 10
                                4"00A300160025",                  % 11
                                4"00A400180026",                  % 12
                                4"00A500210027",                  % 13
                                4"00A70022002C",                  % 14
                                4"00A80023002D",                  % 15
                                4"00A90024002F",                  % 16
                                4"00AB00250032",                  % 17
                                4"00AC00260033",                  % 18
                                4"00AD002C0034";                  % 19
 FILL ESTADO [*] WITH
4"000001000000" , 4"000000000000" , 4"000004000000",   %   0   1   2
4"000005000010" , 4"000000000001" , 4"00001E000000",   %   3   4   5
4"000020000000" , 4"000000000005" , 4"000000000012",   %   6   7   8
4"000021000000" , 4"000022000000" , 4"000025000000",   %   9  10  11
4"000000000009" , 4"000037000013" , 4"00003A000000",   %  12  13  14
4"000049000000" , 4"00004C000000" , 4"00004D000000",   %  15  16  17
4"00004E000000" , 4"000054000000" , 4"00005A000000",   %  18  19  20
4"00005D000000" , 4"000060000000" , 4"000000000029",   %  21  22  23
4"00000000002A" , 4"00000000002B" , 4"00006B000000",   %  24  25  26
4"000000000019" , 4"000000000032" , 4"000000000002",   %  27  28  29
4"00006D000010" , 4"00007A000010" , 4"000091000000",   %  30  31  32
4"0000A0000007" , 4"0000A1000000" , 4"0000A2000000",   %  33  34  35
4"0000A3000010" , 4"0000B0000039" , 4"0000B8000017",   %  36  37  38
4"000000000031" , 4"0000BC000000" , 4"00000000003E",   %  39  40  41
4"00000000003F" , 4"000000000040" , 4"0000CC000000",   %  42  43  44
4"0000D9000000" , 4"0000E6000013" , 4"00000000003C",   %  45  46  47
4"000000000043" , 4"0000E8000010" , 4"0000F5000000",   %  48  49  50
4"000104000000" , 4"00010600000B" , 4"000108000000C",  %  51  52  53
4"000109000000" , 4"00010A000000" , 4"00010B000020",   %  54  55  56
4"00010C000013" , 4"00010F000000" , 4"00000000003B",   %  57  58  59
4"000112000021" , 4"000113000013" , 4"000116000000",   %  60  61  62
4"000119000024" , 4"00011A000000" , 4"00011C000000",   %  63  64  65
4"00000000002C" , 4"000000000003" , 4"000000000004",   %  66  67  68
4"000000000011" , 4"00011E000006" , 4"000000000008",   %  69  70  71
4"000120000010" , 4"000000000018" , 4"00012D000000",   %  72  73  74
4"00013C000000" , 4"00014A000000" , 4"000159000000",   %  75  76  77
4"000168000000" , 4"000177000000" , 4"000186000000",   %  78  79  80
4"000195000000" , 4"0001A4000000" , 4"0001B1000000",   %  81  82  83
4"0001BE000000" , 4"0001CB000000" , 4"0001D8000000",   %  84  85  86
4"000000000041" , 4"000000000042" , 4"00000000000A",   %  87  88  89
4"0001D9000000" , 4"000000000018" , 4"0001DC000000",   %  90  91  92
```

```
4"0001DD000000" , 4"0001DF000000" , 4"0001E3000000",     %  93  94  95
4"0001F2000000" , 4"0001F4000000" , 4"000203000000",     %  96  97  98
4"000205000000" , 4"000207000000" , 4"000000000028",     %  99 100 101
4"000211000000" , 4"000212000015" , 4"000216000016",     % 102 103 104
4"00021A000033" , 4"00021C000034" , 4"00021E000035",     % 105 106 107
4"000220000036" , 4"000222000037" , 4"000224000038",     % 108 109 110
4"00000000002D" , 4"00000000002E" , 4"00000000002F",     % 111 112 113
4"000000000030" , 4"00000000003D" , 4"000000000014",     % 114 115 116
4"000226000000" , 4"000238000000" , 4"00000000000F",     % 117 118 119
4"00000000003A" , 4"00024A000022" , 4"00024C000000",     % 120 121 122
4"00024F000023" , 4"000251000000" , 4"000000000025",     % 123 124 125
4"000254000000" , 4"000266000000" , 4"00000000001D",     % 126 127 128
4"00026800001E" , 4"00026A000000" , 4"00026D000000",     % 129 130 131
4"000000000026" , 4"000000000027" , 4"00000000001A",     % 132 133 134
4"00000000000D" , 4"00026F000000" , 4"000280000000",     % 135 136 137
4"00000000000E" , 4"00000000001C" , 4"000282000000",     % 138 139 140
4"000291000000" , 4"000294000000" , 4"0002A3000000",     % 141 142 143
4"0002A6000000" , 4"0002B5000000" , 4"0002B8000000",     % 144 145 146
4"00000000001F" , 4"0002C9000000" , 4"0002CB000000",     % 147 148 149
4"0002CC000010" , 4"0002E5000000" , 4"0002E6000010",     % 150 151 152
4"0002F2000000" , 4"0002FD000000" , 4"0002FE000000",     % 153 154 155
4"000301000000" , 4"000310000000" , 4"000312000000",     % 156 157 158
4"000315000000" , 4"000316000000" , 4"000327000000",     % 159 160 161
4"000339000000" , 4"00033A000000" , 4"00034A000000",     % 162 163 164
4"000353000000" , 4"000361000000" , 4"000365000000",     % 165 166 167
4"000366000000" , 4"000375000000" , 4"00037A000000",     % 168 169 170
4"000378000000" , 4"00037D000000" , 4"000389000000";     % 171 172 173
FILL TABFOLLOW [*] WITH
                    4"000043FFFFFF",
                    4"000043FFFFFF",
                    4"000004000004",
                    4"000011000005",
                    4"000014FFFFFF",
                    4"000016000007",
                    4"000017FFFFFF",
                    4"00001600003A",
                    4"00001600000A",
                    4"000017FFFFFF",
                    4"00001200000C",
                    4"00001000000D",
                    4"000018FFFFFF",
                    4"00000B00000F",
                    4"00000F000010",
                    4"000016000011",
                    4"000017000012",
                    4"000025000013",
                    4"000026000014",
                    4"000027000015",
                    4"000029000016",
                    4"00002A000017",
                    4"000028000018",
                    4"00003C000019",
                    4"00003D00001A",
                    4"00003E00001B",
                    4"00003F00001C",
                    4"000040000001D",
                    4"000041000035",
```

```
                    4"00001B00001F",
                    4"000028000020",
                    4"00002C000021",
                    4"000006000037",
                    4"000016FFFFFF",
                    4"000021000024",
                    4"000022FFFFFF",
                    4"00000AFFFFFF",
                    4"000005FFFFFF",
                    4"000037000028",
                    4"000038000029",
                    4"00003900002A",
                    4"00003A00002B",
                    4"000007FFFFFF",
                    4"000008FFFFFF",
                    4"000022000002E",
                    4"00002DFFFFFF",
                    4"000016000030",
                    4"000017000031",
                    4"000021000039",
                    4"000012FFFFFF",
                    4"00000DFFFFFF",
                    4"000012FFFFFF",
                    4"000004000036",
                    4"00000CFFFFFF",
                    4"000004000038",
                    4"00000CFFFFFF",
                    4"00000CFFFFFF",
                    4"000017000039",
                    4"000003FFFFFF";
FILL TABEST [*] WITH
4"640000000006" , 4"650000000000" , 4"660000000000",    %   0   1   2
4"67000000517C" , 4"650000000000" , 4"680000000000",    %   3   4   5
4"690000000000" , 4"680000000000" , 4"680000000000",    %   6   7   8
4"6A0000000000" , 4"6B0000004100" , 4"6C00000EE3C4",    %   9  10  11
4"680000000000" , 4"850000000000" , 4"6C00000E61C4",    %  12  13  14
4"6B0000004100" , 4"6E0000000000" , 4"6E0000000000",    %  15  16  17
4"6F0000014140" , 4"6F0000014140" , 4"6B0000004100",    %  18  19  20
4"6B0000004100" , 4"700000001020" , 4"680000000000",    %  21  22  23
4"680000000000" , 4"680000000000" , 4"6B0000004000",    %  24  25  26
4"6D0000000000" , 4"6D0000000000" , 4"840000000000",    %  27  28  29
4"710000004154" , 4"67000000515A" , 4"6C00000E61C4",    %  30  31  32
4"680000000000" , 4"720000000000" , 4"730000000000",    %  33  34  35
4"710000004154" , 4"740000000000" , 4"740000000000",    %  36  37  38
4"740000000000" , 4"6C00000EE1C4" , 4"740000000000",    %  39  40  41
4"740000000000" , 4"740000000000" , 4"6C00000E4144",    %  42  43  44
4"6C00000E4144" , 4"850000000000" , 4"870000000000",    %  45  46  47
4"860000000000" , 4"710000004154" , 4"6C00000E61C4",    %  48  49  50
4"6B0000004000" , 4"880000000000" , 4"890000000000",    %  51  52  53
4"750000000000" , 4"750000000000" , 4"810000000000",    %  54  55  56
4"850000000000" , 4"680000004100" , 4"810000000000",    %  57  58  59
4"810000000000" , 4"850000000000" , 4"6B0000004100",    %  60  61  62
4"890000000000" , 4"760000000000" , 4"680000000000",    %  63  64  65
4"680000000000" , 4"680000000000" , 4"680000000000",    %  66  67  68
4"680000000000" , 4"880000000000" , 4"680000000000",    %  69  70  71
4"710000004154" , 4"8A0000000000" , 4"6C00000E6144",    %  72  73  74
4"6C00000E6144" , 4"6C00000E61C4" , 4"6C00000E61C4",    %  75  76  77
```

```
                        4"00000C000002",          %  32
                        4"00000C000002",          %  33
                        4"00000C000004",          %  34
                        4"00000C000004",          %  35
                        4"00000C000002",          %  36
                        4"00000C000004",          %  37
                        4"00000C000005",          %  38
                        4"00000C000005",          %  39
                        4"00000C000003",          %  40
                        4"00000C000001",          %  41
                        4"00000C000001",          %  42
                        4"00000C000001",          %  43
                        4"00000C000002",          %  44
                        4"00000D000003",          %  45
                        4"00000D000003",          %  46
                        4"00000D000003",          %  47
                        4"00000D000003",          %  48
                        4"00000D000001",          %  49
                        4"00000E000001",          %  50
                        4"00000F000003",          %  51
                        4"00000F000003",          %  52
                        4"00000F000003",          %  53
                        4"00000F000003",          %  54
                        4"00000F000003",          %  55
                        4"00000F000003",          %  56
                        4"00000F000001",          %  57
                        4"000010000003",          %  58
                        4"000010000001",          %  59
                        4"000011000001",          %  60
                        4"000012000003",          %  61
                        4"000012000001",          %  62
                        4"000012000001",          %  63
                        4"000012000001",          %  64
                        4"000012000002",          %  65
                        4"000012000002",          %  66
                        4"000013000001";          %  67
%
FILL POCOID[*] WITH " PROGR","BLOCOL","ISTACO","MLDECL","VARARI",
                    "TIDENL","CONDLE","ESEESD","ECLTER","MIDSEX",
                    "PRLVAR","BLRESF","ACTCON","STBEGI","NENDGO",
                    "TOCOND","RESULT","READAR","QFORWR","ITEFOR",
                    "STEPUN","TILDOI","NTEGER","REALLA","BELPRO",
                    "CEDURE","DECLAR","EFILEE","NDARQE","NDMACR",
                    "ONEWMO","DDIVMA","CROARQ","MACROD","EFINEC",
                    "ALL   ";
INDAF:=PTAF:=1;
QUERARQFOR:=DCLMACRO:=PESQUIDIR:=FALSE;
ERROMACRO:=DEUERRO:=LEARQ:=ERROMACROARQ:=FALSE;
LECAR:=LISTAFONTE:=TRADUZFONTE:=TRUE;
NCOLS:= -1; NIVBEG:=0; NUMFUN:=1000;
TOPENT:=PTVM:= 1;
FIMPROG:=ESTOURO:=PALRES:=FALSE;
PINICIOID:=PTID:=POINTER(POCOID[0]);
PTID:=POINTER(POCOID[0] ) + 213;
INDTABNUM:=0;
PTNI:=PTNF:=POINTER(POCONUMEROS[0]);
PTD:=PDP:=P:=0;
```

```
  COL:=71;
  %
  % INICIALIZACOES PARA O PARSER
  %
  ACEITA:=FALSE;
  TOPO:=1;
  TOPONC:=MAIORTOPONC:=0; TNC:=35;
  INDENT:=0;
  NIVEL:= NIV:=NIVELD:=0;
  PILHA[TOPO]:=0;
  PBASE:=67;
  SHIFTMAX:=2**16 - 1;
  REDMAX:= 2**24 - 1;
  GUARDATOK[1]:=GUARDATOK[2]:=0;
  TESTANDO:=FALSE;
  INDSHIFT:=905;
  INDESTADO:=173;
  INDFOL:=59;
  INDRED:=67;
  %  PEGA PRIMEIRO SIMBOLO
  WRITE(SAI,<X6,46("=")>);
  WRITE(SAI,<X6,"COMPILADOR EXPAND II        -        VERSAO 1.84">);
  WRITE(SAI,<X6,46("=")>);
  READ(ENT,<I1>,CODEXEC);
  IF CODEXEC EQL 2
  THEN LISTAFONTE:=FALSE
  ELSE IF CODEXEC EQL 3
       THEN TRADUZFONTE:=FALSE
       ELSE IF CODEXEC NEQ 1
            THEN BEGIN
                   WRITE(SAI,<X12," *** OPCAO INVALIDA ">);
                   WRITE(SAI,<X12," *** SERA CONSIDERADA A OPCAO 1">);
                 END;
  SCANNER;
  % COMECA ANALISE SINTATICA
  PARSER;
  WRITE(SAI,<X6,45("=")>);
  WRITE(SAI,<X6,"NUMERO DE REGISTROS DO PROGRAMA FONTE = ",I5>,
        CONTREG);
  WRITE(SAI,<X6,"NUMERO DE ERROS DETECTADOS             = ",I5>,
        CONTERRO);
  WRITE(SAI,<X6,"NUMERO DE REGISTROS DO PROGRAMA GERADO= ",I5>,
        CONTALGOL);
  WRITE(SAI,<X6,45("=")>);
  END;
END.
```

APENDICE 4

LISTAGEM DO DIRETORIO DE MACROS

0024700266

*** ORGANIZACAO DO DIRETORIO ***

ESTE DIRETORIO EH COMPOSTO DE :

1 - HEADER

2 - ORGANIZACAO

3 - CORPO

4 - DICIONARIO (REGISTROS DE REFERENCIA)

1 - HEADER

EH O REGISTRO INICIAL DO DIRETORIO (PRIMEIRO REGISTRO) E EM COMPOSTO POR DOIS CAMPOS , CUJO CONTEUDO DE AMBOS DEVE ESTAR NO FORMATO " I5 " ; SENDO QUE O PRIMEIRO CAMPO INFORMA O ENDERECO DO PRIMEIRO REGISTRO DE REFERENCIA, E O SEGUNDO CONTEM O ENDERECO DO ULTIMO REGISTRO DE REFERENCIA.

OBSERVACAO : O ENDERECO CORRESPONDE A ORDEM FISICA DO REGISTRO (SENDO QUE O PRIMEIRO REGISTRO TEM ENDERECO " 0 ") DENTRO DO ARQUIVO.

2 - ORGANIZACAO

EH O TEXTO COMPREENDIDO ENTRE O REGISTRO " 1 " E O " 80 " E TEM COMO OBJETIVO ORIENTAR O USUARIO NA UTILIZACAO DO DIRETORIO.

3 - CORPO

EH COMPOSTO POR DECLARACOES COMPLETAS (ESTRUTURA E DEFINICAO) DE MACROS , INCLUSIVE COM A ESPECIFICACAO DOS DELIMITADORES MACRO , DEFINE E ENDMACRO.

4 - DICIONARIO (REGISTROS DE REFERENCIA)
==========

EH A PARTE DO DIRETORIO QUE CONTEM O " MENU " DE MACROS EXISTENTES , CADA REGISTRO EH COMPOSTO POR 3 CAMPOS , A SABER :

* PRIMEIRO CAMPO - (POSICOES 1 A 5) => CONTEM O ENDERECO DO REGISTRO NO QUAL ENCONTRA-SE O INICIO DA DECLARACAO DA MACRO ; DEVE ESTAR NO FORMATO " I5".

* SEGUNDO CAMPO - (POSICOES 6 A 10) => CONTEM O ENDERECO DO ULTIMO REGISTRO UTILIZADO NA DECLARACAO DA MACRO;DEVE ESTAR NO FORMATO " I5 ".

* TERCEIRO CAMPO => CONTEM O NOME DA MACRO (O QUAL NAO DEVERA ULTRAPASSAR 60 CARACTERES) ; DEVE ESTAR NO FORMATO " A60 ".

```
* * * * * * * * *   INICIO DO CORPO DO DIRETORIO   * * * * * * * * *
MACRO COM SE EXPR E1 ENTAO COM C1 SENAO COM C2
DEFINE COND
          E1 => C1;
             => C2
        END
ENDMACRO
MACRO COM SSE EXPR E1 ENTAO COM C1
DEFINE COND
          E1 => C1;
             =>
        END
ENDMACRO
MACRO COM ENQUANTO EXPR E1 FACA COM C1
DEFINE BEGIN
          LABEL L1;
          L1 : COND
                  E1 => BEGIN
                          C1;
                          GOTO L1
                        END;
                     =>
               END
      END
ENDMACRO
MACRO COM PARA VAR V1 DE ARIT A1 PASSO ARIT A2 ATE ARIT A3 FACA COM C1
DEFINE BEGIN
          NEW L1; LABEL L1;
          V1 := A1;
          L1 : COND V1 <= A3 => BEGIN
```

```
                                        C1;
                                        V1 := V1 + ( A2 );
                                        GOTO L1
                                    END;
                                =>
                END
        END
ENDMACRO
MACRO COM PARA1 VAR V1 DE ARIT A1 ATE ARIT A2 FACA COM C1
DEFINE BEGIN
          NEW L1; LABEL L1;
          V1 := A1;
          L1 : COND V1 <= A2 => BEGIN
                                        C1;
                                        V1 := V1 + 1;
                                        GOTO L1
                                    END;
                                =>
                END
        END
ENDMACRO
MACRO COM REPITA COM C1 ATE EXPR E1
DEFINE BEGIN
          NEW L1;LABEL L1;
          L1 : BEGIN
                  C1;
                  COND E1 =>    ;
                          => GOTO L1
                  END
               END
        END
ENDMACRO
MACRO DECL MATI IDEN I1 [ ARIT A1 , ARIT A2 ]
DEFINE DECLARE
          INTEGER I1#FISICO [(A1)*(A2)];
          MACRO VAR I1 [ARIT A3 ,ARIT A4]
          DEFINE I1#FISICO[(A3 - 1)*(A2) + (A4) ]
          ENDMACRO
        END
ENDMACRO
MACRO DECL MATR IDEN I1 [ ARIT A1 , ARIT A2 ]
DEFINE DECLARE
          REAL I1#FISICO [(A1)*(A2)];
          MACRO VAR I1 [ARIT A3 ,ARIT A4]
          DEFINE I1#FISICO[(A3 - 1)*(A2) + (A4) ]
          ENDMACRO
        END
ENDMACRO
MACRO DECL PROCPAR2 IDEN I1 (IDEN I2,IDEN I3);BLOCO B1
DEFINE DECLARE
          REAL I2,I3;
          PROCEDURE I1;B1;
          MACRO COM I1 (ARIT A1,ARIT A2)
          DEFINE BEGIN
                    I2:=A1; I3:=A2;
                    CALL I1;
                 END
```

```
          ENDMACRO
        END
ENDMACRO
MACRO DECL PROCIO2 IDEN I1 ( VAR V1 , VAR V2 ); BLOCO B1
DEFINE DECLARE
          REAL V1,V2;
          PROCEDURE I1; B1 ;
          MACRO COM I1(VAR V3,VAR V4)
          DEFINE BEGIN
                    V1:=V3; V2:=V4;
                    CALL I1;
                    V3:=V1; V4:=V2;
                  END
          ENDMACRO
        END
ENDMACRO
MACRO COM SOME ARIT A1 EM VAR V1
DEFINE V1:=V1 + (A1)
ENDMACRO
MACRO COM SUBTRAIA ARIT A1 DE VAR V1
DEFINE V1:=V1 - (A1)
ENDMACRO
MACRO COM MOVA ARIT A1 PARA VAR V1
DEFINE V1:=A1
ENDMACRO
MACRO COM MULTIPLIQUE ARIT A1 POR ARIT A2 DANDO VAR V1
DEFINE V1 := (A1) * (A2)
ENDMACRO
MACRO COM DIVIDA ARIT A1 POR ARIT A2 DANDO VAR V1 RESTANDO VAR V2
DEFINE V1:=(A1) DIV (A2) ; V2 := (A1) MOD (A2)
ENDMACRO
MACRO COM ELEVE ARIT A1 A ARIT A2 DANDO VAR V1
DEFINE BEGIN
          NEW I; NEW L1; INTEGER I; LABEL L1;
          I:=1; V1:=A1;
          L1: COND I < A2 => BEGIN
                                V1:= V1 * V1;
                                I:=I + 1;
                                GOTO L1
                              END;
                            =>
              END
        END
ENDMACRO
MACRO DECL MATILI IDEN I1 [ARIT A1:ARIT A2,ARIT A3:ARIT A4]
DEFINE DECLARE
          INTEGER I1#FISICO[(A2 -(A1)+ 1)*(A4 -(A3)+ 1)];
          MACRO VAR I1 [ARIT A5 ,ARIT A6]
          DEFINE I1#FISICO[(A5-(A1))*(A4-(A3)+1)+(A6)]
          ENDMACRO
        END
ENDMACRO
MACRO DECL MATRLI IDEN I1 [ARIT A1:ARIT A2,ARIT A3:ARIT A4]
DEFINE DECLARE
          REAL I1#FISICO[(A2 -(A1)+ 1)*(A4 -(A3)+ 1)];
          MACRO VAR I1 [ARIT A5 ,ARIT A6]
          DEFINE I1#FISICO[(A5-(A1))*(A4-(A3)+1)+(A6)]
```

```
          ENDMACRO
        END
  ENDMACRO
  MACRO DECL PROCPAR1 IDEN I1 ( IDEN I2 );BLOCO B1
  DEFINE DECLARE
           REAL I2;
           PROCEDURE I1;B1;
           MACRO COM I1 (ARIT A1)
           DEFINE BEGIN
                    I2:=A1;
                    CALL I1;
                  END
           ENDMACRO
         END
  ENDMACRO
  MACRO DECL PROCPAR3 IDEN I1 (IDEN I2,IDEN I3,IDEN I4);BLOCO B1
  DEFINE DECLARE
           REAL I2,I3,I4;
           PROCEDURE I1;B1;
           MACRO COM I1 (ARIT A1,ARIT A2,ARIT A3)
           DEFINE BEGIN
                    I2:=A1; I3:=A2;
                   CALL I1;
                  END
           ENDMACRO
         END
  ENDMACRO
0007700082SE
0008300088SSE
0008900100ENQUANTO
0010100113PARA
0011400126PARA1
0012700137REPITA
0013800145MATI
0014600153MATR
0015400165PROCPAR2
0016600178PROCI02
0017900181SOME
0018200184SUBTRAIA
0018500187MOVA
0018800190MULTIPLIQUE
0019100193DIVIDA
0019400206ELEVE
0020700214MATILI
0021500222MATRLI
0022300234PROCPAR1
0023500246PROCPAR3
```

APENDICE 5

PROGRAMAS EXEMPLO

```
1
BEGIN
  %
  %            ***   PROGRAMA  1   ***
  %
  % FUNCAO : - ILUSTRAR O USO DAS MACROS "SSE"  E "ENQUANTO"
  %           - ORDENACAO DE UM VETOR DE N ELEMENTOS
  %
  FILE ENT1(KIND=PACK), SAI1(KIND=PACK);
  INTEGER N;
  READ(ENT1,<I3>,N);
  BEGIN
    INTEGER I,J;
    REAL AUX,X[N];
    MACRO COM SE EXPR E1 ENTAO COM C1
    DEFINE COND
             E1 => C1;
                =>
          END
    ENDMACRO;
    MACRO COM ENQUANTO EXPR E2 FACA COM C2
    DEFINE BEGIN
             LABEL L1;
             L1 : COND
                    E2 => BEGIN
                            C2;
                            GOTO L1
                          END;
                       =>
                  END
          END
    ENDMACRO;
    READ(ENT1,<*F6,2>,N,FOR I:=1 STEP 1 UNTIL N DO X[I]);
    J:=0;
    ENQUANTO J < N FACA
    BEGIN
      I:=1;
      ENQUANTO I < N FACA
      BEGIN
        SE X[I] > X[I + 1]
        ENTAO BEGIN
                AUX:=X[I];
                X[I]:= X[I + 1];
                X[I + 1]:= AUX
              END;
        I:= I + 1
      END;
```

```
      J := J + 1
    END;
    I:=1;
    ENQUANTO I <= N FACA
    BEGIN
      WRITE(SAI1,<X3,I3,X2,F6.2>,I,X[I]);
      I := I + 1
    END;
  END
END.
```

```
==========================================
COMPILADOR EXPAND II        -        VERSAO 1.84
==========================================
     BEGIN
       %
       %             ***    PROGRAMA  1    ***
       %
       % FUNCAO : - ILUSTRAR O USO DAS MACROS "SSE"  E "ENQUANTO"
       %           - ORDENACAO DE UM VETOR DE N ELEMENTOS
       %
       FILE ENT1(KIND=PACK), SAI1(KIND=PACK);
       INTEGER N;
       READ(ENT1,<I3>,N);
       BEGIN
         INTEGER I,J;
         REAL AUX,X[N];
         MACRO COM SE EXPR E1 ENTAO COM C1
         DEFINE COND
                  E1 => C1;
                     =>
               END
         ENDMACRO;
         MACRO COM ENQUANTO EXPR E2 FACA COM C2
         DEFINE BEGIN
                  LABEL L1;
                  L1 : COND
                         E2 => BEGIN
                                 C2;
                                 GOTO L1
                               END;
                            =>
                       END
              END
         ENDMACRO;
         READ(ENT1,<*F6.2>,N,FOR I:=1 STEP 1 UNTIL N DO X[I]);
         J:=0;
         ENQUANTO J < N FACA
         BEGIN
           I:=1;
           ENQUANTO I < N FACA
           BEGIN
             SE X[I] > X[I + 1]
             ENTAO BEGIN
                     AUX:=X[I];
                     X[I]:= X[I + 1];
                     X[I + 1]:= AUX
                   END;
             I:= I + 1
           END;
           J := J + 1
         END;
         I:=1;
         ENQUANTO I <= N FACA
         BEGIN
           WRITE(SAI1,<X3,I3,X2,F6.2>,I,X[I]);
           I := I + 1
         END;
```

```
        END
      END.
==========================================================
NUMERO DE REGISTROS DO PROGRAMA FONTE =     56
NUMERO DE ERROS DETECTADOS            =      0
NUMERO DE REGISTROS DO PROGRAMA GERADO=     55
==========================================================
```

*** RESULTADO DA TRADUCAO DO PROGRAMA 1 ***

```
BEGIN
  FILE  ENT1(KIND=PACK), SAI1(KIND=PACK);
  INTEGER N;
  READ(ENT1,<I3>,N);
  BEGIN
    INTEGER I,J;
    REAL AUX;
    REAL ARRAY X[1:N];
    READ(ENT1,<*F6.2>,N,FOR I:=1 STEP 1 UNTIL N DO X[I]);
    J:=0;
    BEGIN
      LABEL L1;
      L1:IF J<N
         THEN BEGIN
                BEGIN
                  I:=1;
                  BEGIN
                    LABEL L1;
                    L1:IF I<N
                       THEN BEGIN
                              BEGIN
                                IF X[I]>X[I+1]
                                THEN BEGIN
                                       AUX:=X[I];
                                       X[I]:=X[I+1];
                                       X[I+1]:=AUX;
                                     END
                                ELSE ;
                                I:=I+1;
                              END;
                              GO TO L1;
                            END
                       ELSE ;
                  END;
                  J:=J+1;
                END;
                GO TO L1;
              END
         ELSE ;
    END;
    I:=1;
    BEGIN
      LABEL L1;
      L1:IF I<=N
         THEN BEGIN
                BEGIN
                  WRITE(SAI1,<X3,I3,X2,F6.2>,I,X[I]);
                  I:=I+1;
                END;
                GO TO L1;
              END
```

```
        ELSE ;
    END;
  END;
END.
```

```
1
BEGIN
  %
  %          ***  PROGRAMA 2  ***
  %
  % FUNCAO : - ILUSTRAR O USO DAS MACROS "MATI" E "PARA"
  %           - MULTIPLICACO DE MATRIZES (  (M,N) , (N,P) )
  %
  FILE ENT1(KIND = PACK),
       SAI1(KIND = PACK);
  INTEGER N,M,P;
  READ(ENT1,<3I3>,N,M,P);
  BEGIN
    ARQMACRO MATI,ENQUANTO;
    INTEGER I,J,K;
    MATI A[M,N];
    MATI B[N,P];
    MATI C[M,P];
    MACRO COM PARA1 VAR V1 DE ARIT A1 ATE ARIT A2 FACA COM C1
    DEFINE BEGIN
             V1 := A1;
             ENQUANTO V1 <= A2
             FACA BEGIN
                    C1;
                    V1 := V1 + 1
                  END
         END
    ENDMACRO;
    BEGIN
      PARA1 I DE 1 ATE M
      FACA READ(ENT1,<*I3>,N,FOR J:=1 STEP 1 UNTIL N
                                              DO A[I,J]);
      PARA1 I DE 1 ATE N
      FACA READ(ENT1,<*I3>,P,FOR J:=1 STEP 1 UNTIL P
                                              DO B[I,J]);
      PARA1 I DE 1 ATE M
      FACA PARA1 J DE 1 ATE P
           FACA PARA1 K DE 1 ATE N
                FACA C[I,J] := C[I,J] + A[I,K] * B[K,J];
      WRITE(SAI1,<X5," ***  MATRIZ RESULTADO  *** ">);
      PARA1 I DE 1 ATE M
      FACA WRITE(SAI1,<X5,*I5>,P,FOR J:=1 STEP 1 UNTIL P
                                              DO C[I,J])
    END
  END
END.
```

```
        ================================================
        COMPILADOR EXPAND II        -        VERSAO 1,84
        ================================================
             BEGIN
               %
               %          ***  PROGRAMA 2  ***
               %
               % FUNCAO : - ILUSTRAR O USO DAS MACROS "MATI" E "PARA"
               %           - MULTIPLICACO DE MATRIZES (  (M,N) , (N,P) )
               %
               FILE ENT1(KIND = PACK),
                    SAI1(KIND = PACK);
               INTEGER N,M,P;
               READ(ENT1,<3I3>,N,M,P);
               BEGIN
                 ARQMACRO MATI,ENQUANTO;
ARQMACRO       MACRO DECL MATI IDEN I1 [ ARIT A1 , ARIT A2 ]
ARQMACRO       DEFINE DECLARE
ARQMACRO                 INTEGER I1#FISICO [(A1)*(A2)];
ARQMACRO                 MACRO VAR I1 [ARIT A3 ,ARIT A4]
ARQMACRO                 DEFINE I1#FISICO[(A3 - 1)*(A2) + (A4) ]
ARQMACRO                 ENDMACRO
ARQMACRO               END
ARQMACRO       ENDMACRO
ARQMACRO       MACRO COM ENQUANTO EXPR E1 FACA COM C1
ARQMACRO       DEFINE BEGIN
ARQMACRO                 LABEL L1;
ARQMACRO                 L1 : COND
ARQMACRO                        E1 => BEGIN
ARQMACRO                                C1;
ARQMACRO                                GOTO L1
ARQMACRO                              END;
ARQMACRO                         =>
ARQMACRO                      END
ARQMACRO             END
ARQMACRO       ENDMACRO
                 INTEGER I,J,K;
                 MATI A[M,N];
                 MATI B[N,P];
                 MATI C[M,P];
                 MACRO COM PARA1 VAR V1 DE ARIT A1 ATE ARIT A2 FACA COM C
                 DEFINE BEGIN
                          V1 := A1;
                          ENQUANTO V1 <= A2
                          FACA BEGIN
                                 C1;
                                 V1 := V1 + 1
                               END
                        END
                 ENDMACRO;
                 BEGIN
                   PARA1 I DE 1 ATE M
                   FACA READ(ENT1,<*I3>,N,FOR J:=1 STEP 1 UNTIL N
                                                        DO A[I,J]);
                   PARA1 I DE 1 ATE N
                   FACA READ(ENT1,<*I3>,P,FOR J:=1 STEP 1 UNTIL P
                                                        DO B[I,J]);
```

```
            PARA1 I DE 1 ATE M
            FACA PARA1 J DE 1 ATE P
                 FACA PARA1 K DE 1 ATE N
                      FACA C[I,J] := C[I,J] + A[I,K] * B[K,J];
            WRITE(SAI1,<X5," ***  MATRIZ RESULTADO  *** ">);
            PARA1 I DE 1 ATE M
            FACA WRITE(SAI1,<X5,*IS>,P,FOR J:=1 STEP 1 UNTIL P
                                                    DO C[I,J])
          END
        END
      END.
==========================================
NUMERO DE REGISTROS DO PROGRAMA FONTE =    65
NUMERO DE ERROS DETECTADOS            =     0
NUMERO DE REGISTROS DO PROGRAMA GERADO=   109
==========================================
```

*** RESULTADO DA TRADUCAO DO PROGRAMA 2 ***

```
BEGIN
  FILE  ENT1(KIND = PACK),
        SAI1(KIND = PACK);
  INTEGER N,M,P;
  READ(ENT1,<3I3>,N,M,P);
  BEGIN
    INTEGER I,J,K;
    INTEGER  ARRAY AFISICO[1:M*N];
    INTEGER  ARRAY BFISICO[1:N*P];
    INTEGER  ARRAY CFISICO[1:M*P];
    BEGIN
      BEGIN
        I:=1;
        BEGIN
          LABEL L1;
          L1:IF I<=M
             THEN BEGIN
                    BEGIN
                      READ(ENT1,<*I3>,N,FOR J:=1 STEP 1 UNTIL N DO
                      AFISICO[(I-1)*N+J]);
                      I:=I+1;
                    END;
                    GO TO L1;
                  END
             ELSE ;
        END;
      END;
      BEGIN
        I:=1;
        BEGIN
          LABEL L1;
          L1:IF I<=N
             THEN BEGIN
                    BEGIN
                      READ(ENT1,<*I3>,P,FOR J:=1 STEP 1 UNTIL P DO
                      BFISICO[(I-1)*P+J]);
                      I:=I+1;
                    END;
                    GO TO L1;
                  END
             ELSE ;
        END;
      END;
      BEGIN
        I:=1;
        BEGIN
          LABEL L1;
          L1:IF I<=M
             THEN BEGIN
                    BEGIN
                      BEGIN
                        J:=1;
```

```
                        BEGIN
                          LABEL L1;
                          L1:IF J<=P
                              THEN BEGIN
                                     BEGIN
                                       BEGIN
                                         K:=1;
                                         BEGIN
                                           LABEL L1;
                                           L1:IF K<=N
                                               THEN BEGIN
                                                      BEGIN
                                                      CFISICO[(I-1)*P+J]:=
                                                      CFISICO[(I-1)*P+J]+
                                                      AFISICO[((I-1)*N+K)]*
                                                      BFISICO[((K-1)*P+J)];
                                                      K:=K+1;
                                                      END;
                                                      GO TO L1;
                                                      END
                                                  ELSE ;
                                         END;
                                       END;
                                       J:=J+1;
                                     END;
                                     GO TO L1;
                                   END
                               ELSE ;
                          END;
                        END;
                        I:=I+1;
                      END;
                      GO TO L1;
                    END
                ELSE ;
        END;
      END;
      WRITE(SAI1,<X5," ***  MATRIZ RESULTADO  *** ">);
      BEGIN
        I:=1;
        BEGIN
          LABEL L1;
          L1:IF I<=M
              THEN BEGIN
                     BEGIN
                       WRITE(SAI1,<X5,*I5>,P,FOR J:=1 STEP 1 UNTIL P DO
                       CFISICO[(I-1)*P+J]);
                       I:=I+1;
                     END;
                     GO TO L1;
                   END
              ELSE ;
        END;
      END;
    END;
  END;
END.
```

```
1
BEGIN
  %
  %            *** PROGRAMA 3 ***
  %
  % FUNCAO - ILUSTRAR O USO DA MACRO "SE" , DO COMANDO
  %           "RESULT" E DO BLOCO "RESULT".
  %
  FILE ENT1(KIND = PACK) , SAI1(KIND = PACK);
  ARQMACRO SE;
  REAL A,B,C;
  READ(ENT1,/,A,B,C);
  COND BEGIN
         SE A > B
         ENTAO RESULT A
         SENAO RESULT B
       END<C =>WRITE(SAI1,<"MAIOR=",F6.2>,C);
              =>WRITE(SAI1,<"MAIOR=",F6.2>,BEGIN
                                             SE A > B
                                             ENTAO RESULT A
                                             SENAO RESULT B
                                           END)
  END
END.
```

```
=============================================================
COMPILADOR EXPAND II        -        VERSAO 1.84
=============================================================
      BEGIN
        %
        %             *** PROGRAMA 3 ***
        %
        % FUNCAO - ILUSTRAR O USO DA MACRO "SE" , DO COMANDO
        %           "RESULT" E DO BLOCO "RESULT".
        %
        FILE ENT1(KIND = PACK) , SAI1(KIND = PACK);
        ARQMACRO SE;
ARQMACRO    MACRO COM SE EXPR E1 ENTAO COM C1 SENAO COM C2
ARQMACRO    DEFINE COND
ARQMACRO              E1 => C1;
ARQMACRO                 => C2
ARQMACRO            END
ARQMACRO    ENDMACRO
        REAL A,B,C;
        READ(ENT1,/,A,B,C);
        COND BEGIN
               SE A > B
               ENTAO RESULT A
               SENAO RESULT B
             END<C =>WRITE(SAI1,<"MAIOR=",F6.2>,C);
                  =>WRITE(SAI1,<"MAIOR=",F6.2>,BEGIN
                                                 SE A > B
                                                 ENTAO RESULT A
                                                 SENAO RESULT B
                                               END)
        END
      END.
=============================================================
NUMERO DE REGISTROS DO PROGRAMA FONTE =    29
NUMERO DE ERROS DETECTADOS            =     0
NUMERO DE REGISTROS DO PROGRAMA GERADO=    24
=============================================================
```

*** RESULTADO DA TRADUCAO DO PROGRAMA 3 ***

```
BEGIN
  REAL ARRAY OJVFTEMP[1:01];
  FILE  ENT1(KIND = PACK) , SAI1(KIND = PACK);
  REAL A,B,C;
  REAL PROCEDURE FUNXYZ1001;
  BEGIN
    FUNXYZ1001:=0;
    IF A>B
    THEN FUNXYZ1001:=A
    ELSE FUNXYZ1001:=B;
  END;
  READ(ENT1,/,A,B,C);
  IF FUNXYZ1001<C
  THEN WRITE(SAI1,<"MAIOR=",F6.2>,C)
  ELSE BEGIN
         BEGIN
           OJVFTEMP[01]:=0;
           IF A>B
           THEN OJVFTEMP[01]:=A
           ELSE OJVFTEMP[01]:=B;
         END;
         WRITE(SAI1,<"MAIOR=",F6.2>, OJVFTEMP[01]);
       END;
END.
```

```
3
BEGIN
   %
   %              ***    PROGRAMA    4     ***
   %
   %  FUNCAO   :  =ILUSTRAR O USO DA MACRO "SE"  E A ACAO
   %               DO RECUPERADOR DE ERROS.
   %
   MACRO COM SE EXPR E1 ENTAO COM C1 SENAO COM C2
   DEFINE COND E1 => C1;
                  => C2
          END
   ENDMACRO;
   MACRO COM ENQUANTO EXPR E2 FACA COM C3
   DEFINE BEGIN
            NEW L1; LABEL L1;
            L1 : COND E2 => BEGIN
                              C3;
                              GOTO L1
                            END;
                       =>
                END
         END
  ENDMACRO;
  MACRO COM SSE EXPR E3 ENTAO COM C4
  DEFINE COND E3 => C4;
                 =>
         END
  ENDMACRO;
  A;=8;
  SSE A > B
  ENTAO BEGIN
          AUX;= A
          A;=B;
          B;=AUX
        END;
  AUX;=1;
  SE A > B
  ENTAO MAIOR = A
  SENAO MAIOR := B;
  AUX;=2;
  ENQUANTO A > B
  ENTAO A;=A - 1;
  WRITE(SAI,/,A,B)
END.
```

```
==========================================================
COMPILADOR EXPAND II        -        VERSAO 1.84
==========================================================
     BEGIN
        %              ***   PROGRAMA   4    ***
        %
        %  FUNCAO   :  -ILUSTRAR O USO DA MACRO "SE"  E A ACAO
        %               DO RECUPERADOR DE ERROS.
        %
        MACRO COM SE EXPR E1 ENTAO COM C1 SENAO COM C2
        DEFINE COND E1 => C1;
                       => C2
               END
        ENDMACRO;
        MACRO COM ENQUANTO EXPR E2 FACA COM C3
        DEFINE BEGIN
                 NEW L1; LABEL L1;
                 L1 : COND E2 => BEGIN
                                   C3;
                                   GOTO L1
                                 END;
                              =>
                    END
             END
      ENDMACRO;
      MACRO COM SSE EXPR E3 ENTAO COM C4
      DEFINE COND E3 => C4;
                     =>
             END
      ENDMACRO;
      A:=B;
      SSE A > B
      ENTAO BEGIN
              AUX:= A
              A:=B;

     *** SEGUIDOR DE IDENTIFICADOR INVALIDO
              B:=AUX
            END;
      AUX:=1;
      SE A > B
      ENTAO MAIOR = A

     *** FALTOU SINAL DE ATRIBUICAO (:=)
      SENAO MAIOR := B;
      AUX:=2;
      ENQUANTO A > B
      ENTAO A:=A - 1;

     *** ERRO NA UTILIZACAO DE UMA MACRO
      WRITE(SAI,/,A,B)
     END.
==========================================================
NUMERO DE REGISTROS DO PROGRAMA FONTE =     44
NUMERO DE ERROS DETECTADOS            =      3
NUMERO DE REGISTROS DO PROGRAMA GERADO=      0
==========================================================
```

APENDICE 6

MENSAGENS DE ERRO

CODIGO	MENSAGEM
001	* TAMANHO DO IDENTIFICADOR NAO DEVE EXCEDER 64 CARACTERES
003	* CARATER NAO PERTENCENTE A LINGUAGEM
004	*** CATEGORIA DA MACRO NAO ESPECIFICADA
005	*** NOME DA MACRO DEVE SER UM IDENTIFICADOR
006	*** NAO E PERMITIDA A OCORRENCIA DE PARAMETROS CONSECUTIVOS
007	*** OS PARAMETROS DA MACRO DEVEM SER DISTINTOS ENTRE SI
008	*** IDENTIFICADOR DE PARAMETRO INVALIDO
009	*** NUMERO DE PARAMETROS DA ESTRUTURA NAO COINCIDE COM O NUM. DA DEFINICAO
010	*** USO INDEVIDO DE IDENTIFICADOR DE PARAMETRO
011	*** SINTAXE DA DEFINICAO DA MACRO ESTA INCORRETA
012	*** CONFLITO SHIFT-REDUCE
013	*** CONFLITO REDUCE-REDUCE
014	*** A PARTIR DESTE PONTO O PROGRAMA FOI IGNORADO POR HAVER MACRO ERRADA
015	*** A ANALISE DO BLOCO ACIMA NAO FOI CONCLUIDA POR HAVER MACRO ERRADA
016	*** MACRO NAO EXISTENTE NO DIRETORIO
017	*** ESPERAVA-SE , OU ; NA DECLARACAO ARQMACRO
018	*** DECLARACAO DE MACRO INCOMPLETA
019	*** AREA DE CODIGO CHEIA (NODOS)
100	*** ESPERAVA-SE UM BEGIN
101	*** FINAL DE PROGRAMA ERA ESPERADO

```
102       *** ESPERAVA-SE . DE FIM DE PROGRAMA
103       *** ESPERAVA-SE DECLARACAO OU COMANDO
104       *** FALTOU DELIMITADOR ( ; OU END )
105       *** ESPERAVA-SE ;
106       *** FALTOU SINAL DE ATRIBUICAO (:=)
107       *** ESPERAVA-SE UM IDENTIFICADOR
108       *** EXPRESSAO INVALIDA
109       *** SEGUIDOR DE IDENTIFICADOR INVALIDO
110       *** ESPERAVA-SE UM (
111       *** ESPERAVA-SE IDENT. OU PROCEDURE
112       *** DECLARACAO INVALIDA
113       *** COMANDO INVALIDO
114       *** ESPERAVA-SE END
115       *** ESPERAVA-SE =>
116       *** SEGUIDOR DE EXPRESSAO INVALIDO
117       *** FALTANDO ESPECIFICACAO DE ARQUIVO E FORMATO
118       *** ESPERAVA-SE ; OU #
119       *** ESPERAVA-SE )
120       *** FALTANDO !
121       *** ESPERAVA-SE ,
122       *** ESPERAVA-SE := OU #
123       *** ESPERAVA-SE STEP
124       *** ESPERAVA-SE UNTIL
125       *** ESPERAVA-SE DO
126       *** ESPERAVA-SE #
127       *** ESPERAVA-SE OP. DE MULTIPLICACAO
128       *** ESPERAVA-SE UM ) OU UMA ,
129       *** DELIMITADOR (, ; OU END ) ESPERADO
130       *** ERRO NA UTILIZACAO DE UMA MACRO
```

131 *** FINAL DE PROGRAMA INESPERADO

132 *** SEGUIDOR DE BLOCO INVALIDO

133 *** SEGUIDOR DE VARIAVEL INVALIDO

134 *** SEGUIDOR DE CONSTANTE INVALIDO

135 *** SEGUIDOR DE BLOCO RESULT INVALIDO

136 *** ESPERAVA-SE + - ; OU END

137 *** ESPERAVA-SE # ; OU END

138 *** ESPERAVA-SE END

139 *** ESPERAVA-SE + - ; OU)

200 *** NUMERO DE MACROS MAIOR QUE O PERMITIDO
- AUMENTAR NMAXMACRO

201 *** NUMERO DE PARAMETROS MAIOR QUE O PERMITIDO
- AUMENTAR NMAXPAR

202 *** NUMERO DE ELEMENTOS MAIOR QUE O PERMITIDO
- AUMENTAR NMAXELEM

203 *** NUMERO DE FOLLOW'S MAIOR QUE O PERMITIDO
- AUMENTAR NMAXFOL

204 *** NUMERO DE ESTADOS MAIOR QUE O PERMITIDO
- AUMENTAR NMAXEST

205 *** NUMERO DE SHIFT'S MAIOR QUE O PERMITIDO
- AUMENTAR NMAXSHIFT

206 *** NUMERO DE CONSTANTES MAIOR QUE O PERMITIDO
- AUMENTAR NMAXCONST

207 *** NUMERO DE IDENTIFICADORES MAIOR QUE O PERMITIDO
- AUMENTAR NMAXIDEN

208 *** NUMERO DE ESPECIFICACOES DE ARQUIVO-FORMATO
MAIOR QUE O PERMITIDO - AUMENTAR NMAXARQFOR

209 *** NUMERO DE NODOS MAIOR QUE O PERMITIDO
- AUMENTAR NMAXNODO

210 *** ESTOURO DA PILHA SINTATICA - AUMENTAR MAXTOPO

211 *** NUMERO DE NIVEIS MAIOR QUE O PERMITIDO
- AUMENTAR NMAXNIV

212 *** ESTOURO DO POCO DE NUMEROS - AUMENTAR MAXTAMCONST

213 *** ESTOURO DO POCO DE ESPECIFICACOES DE ARQUIVO-FORMATO

- AUMENTAR MAXTAMAF

214 *** ESTOURO DO POCO DE IDENTIFICADORES
- AUMENTAR MAXTAMIDEN

215 *** NUMERO DE ESTADOS DE UMA LISTA EXCEDEU O MAXIMO
- AUMENTAR MAXTAMLISTA

216 *** NUMERO DE PALAVRAS RESERVADAS EXCEDEU O MAXIMO
- AUMENTAR NMAXCOL

APENDICE 7

RELACAO DOS NODOS

NODO	CODIGO	PRODUCAO CORRESPONDENTE
PONTO (.)	1	1
BE	2	2
LC	3	3
LDC	4	4
ATRIB (:=)	6	6
NGOTO	7	7
COND	8	8
ROT (:)	10	10
RESULT	11	11
CALL	12	12
LEIA	13	13
ESCREVA	14	14
WLIT	15	15
VAZIO	16	16
LD	17	17
BRKT (+!)	20	20
MAIS (+)	21	21
MENOS (-)	22	22
JV (#)	24	24
LCOND	25	25
VERD (=>)	26	26
FALS (=>)	27	27
LES	28	28
NFOR	31	31
NSTEP	9	31
NUNTIL	18	31
NDO	19	31
INTEG	32	32
REALL	33	33
CI	34	34
CR	35	35
LAB	36	36
PROC	37	37
IP	38	38
RP	39	39
DECEND	40	40
NFILE	41	41
ENDMACRO	43	43
NNEW	44	44
VEZES (*)	45	45
DIVIS (/)	46	46
NMOD (MOD)	47	47
NDIV (DIV)	48	48
ID	50	50
MA (>)	51	51
ME (<)	52	52

IG	(=)	53	53
MM	(<>)	54	54
MEI	(<=)	55	55
MAI	(>=)	56	56
LVAR		58	58
BR		60	60
MEU	(-)	65	65
MAU	(+)	66	66
DIG		67	67

NODOS ESPECIAIS

ARQFOR (CODIGO 5) => ASSOCIADO AOS COMANDOS DE ENTRADA E SAIDA.

TEMP (CODIGO 68) E LCAUX (CODIGO 69) => NODOS ASSOCIADOS AO USO DE BLOCO RESULT EM EXPRESSOES ARITMETICAS PERTINENTES AOS COMANDOS DE ATRIBUICAO , COMANDOS RESULT E COMANDOS WRITE.

FUN (CODIGO 98) E NFUN (CODIGO 99) => NODOS ASSOCIADOS AO USO DE BLOCOS RESULT EM CONDICOES PERTINENTES AO COMANDO COND.

REP (CODIGO 88) => NODO USADO PARA DUPLICACAO DE SEGMENTOS DE PROGRAMA , ASSOCIADO AOS PROCEDIMENTOS DE MACRO EXPANSAO.

```
4"6C00000E61C4" , 4"6C00000E61C4" , 4"6C00000E61C4",    %  78  79  80
4"6C00000E61C4" , 4"6C00000E4144" , 4"6C00000E4144",    %  81  82  83
4"6C00000E4144" , 4"6C00000E4144" , 4"770000000000",    %  84  85  86
4"6D0000000000" , 4"6D0000000000" , 4"680000000000",    %  87  88  89
4"780000000000" , 4"6D0000000000" , 4"790000000000",    %  90  91  92
4"800000000000" , 4"6B0000004140" , 4"6C00000E61C4",    %  93  94  95
4"760000000000" , 4"6C00000E61C4" , 4"760000000000",    %  96  97  98
4"640000000004" , 4"700000001000" , 4"680000000000",    %  99 100 101
4"690000000000" , 4"740000000000" , 4"740000000000",    % 102 103 104
4"740000000000" , 4"740000000000" , 4"740000000000",    % 105 106 107
4"740000000000" , 4"740000000000" , 4"740000000000",    % 108 109 110
4"740000000000" , 4"740000000000" , 4"740000000000",    % 111 112 113
4"740000000000" , 4"740000000000" , 4"740000000000",    % 114 115 116
4"6C00000E6DC4" , 4"6C00000E6DC4" , 4"680000000000",    % 117 118 119
4"810000000000" , 4"880000000000" , 4"640000020004",    % 120 121 122
4"880000000000" , 4"640000020004" , 4"680000000000",    % 123 124 125
4"6C00000EE3C4" , 4"800000000000" , 4"800000000000",    % 126 127 128
4"880000000000" , 4"6B0000004100" , 4"800000000000",    % 129 130 131
4"680000000000" , 4"680000000000" , 4"720000000000",    % 132 133 134
4"680000000000" , 4"6C00000E69C4" , 4"7A0000000000",    % 135 136 137
4"680000000000" , 4"800000000000" , 4"6C00000E61C4",    % 138 139 140
4"7B0000000000" , 4"6C00000E61C4" , 4"7C0000000000",    % 141 142 143
4"6C00000E61C4" , 4"7D0000000000" , 4"6C00000E61C4",    % 144 145 146
4"800000000000" ;                                       % 147
FILL REDUCAO [*] WITH
                    4"000000000001",                    %   0
                    4"000001000002",                    %   1
                    4"000002000003",                    %   2
                    4"000003000003",                    %   3
                    4"000003000003",                    %   4
                    4"000003000001",                    %   5
                    4"000004000003",                    %   6
                    4"000004000002",                    %   7
                    4"000004000003",                    %   8
                    4"000004000001",                    %   9
                    4"000004000003",                    %  10
                    4"000004000002",                    %  11
                    4"000004000002",                    %  12
                    4"000004000006",                    %  13
                    4"000004000006",                    %  14
                    4"000004000004",                    %  15
                    4"000004000000",                    %  16
                    4"000005000003",                    %  17
                    4"000005000001",                    %  18
                    4"000006000001",                    %  19
                    4"000006000004",                    %  20
                    4"000007000003",                    %  21
                    4"000007000003",                    %  22
                    4"000007000001",                    %  23
                    4"000008000003",                    %  24
                    4"000008000001",                    %  25
                    4"000009000005",                    %  26
                    4"000009000002",                    %  27
                    4"00000A000003",                    %  28
                    4"00000A000001",                    %  29
                    4"00000B000001",                    %  30
                    4"00000B00000A",                    %  31
```