UM MECANISMO DE EXTENSÃO DE LINGUAGENS E A LINGUAGEM EXTENSÍVEL LPSE

Luiz Carlos Montez Monte

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

Aprovada por:

José Lucas Mourão Rangel Netto
Presidente

Estevam Gilberto De Simone

Emmanuel Piseces Lopes Passos

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 1982

MONTE, LUIZ CARLOS MONTEZ

Um Mecanismo de Extensão de Linguagens e a Linguagem Extensível LPSE (Rio de Janeiro) 1982.

IX, 160p. 29,7cm (COPPE-UFRJ), M.Sc, Engenharia de Sistemas, 1982).

Tese - Universidade Federal do Rio de Janeiro, Programa de Engenharia de Sistemas da Coordenação dos Programas de Pós-Graduação em Engenharia.

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

dedicado a meus pais

AGRADECIMENTOS.

Desejo agradecer aos meus colegas da COBRA e do Fundão, todos nessa batalha pela criação e manutenção de tecnologia nacional em computadores. Expresso meu especial agradecimento aos que puderam me ajudar mais diretamente nesse trabalho: à Lila, por sua dedicação e carinho; à Rosana, Luiz Fernando e Ivan, meu companheiro inseparável nos estudos, por terem dobrado esforços na equipe LPS; ao Jorge, pela eficiente datilografia; ao Eugênio, pela valiosa ajuda na pesquisa bibliográfica.

Muito tenho que agradecer a Eduardo Lessa, pelas portas que me abriu, inclusive mais essa.

Ao Estevam, excelente professor e eterno orientador de meus estudos sobre compiladores.

Finalmente, agradeço a orientação criativa, segura e simpática do Rangel.

RESUMO

As linguagens de programação, mesmo as que possuem padrões publicados e registrados, estão sempre sendo alteradas pelos fabricantes de software, pesquisadores e até por seus usuários.

Seja qual for o mecanismo empregado, se a alteração na linguagem cria recursos para novas adaptações, criando novos tipos de declarações, em particular declarações de macros ou dispositivos equivalentes, essa alteração em si também é um mecanismo que permite novas modificações na linguagem.

A linguagem extensora "E" é aqui proposta como um mecanismo para a programação de extensões, facilmente adaptáveis às linguagens de programação usuais.

Essa linguagem permite que se programe extensões para outra linguagem a ela adaptada, através de declarações de substituição de textos. Nessa declaração, o programador fornece um padrão que, toda vez que for reconhecido no texto, é substituído por um outro trecho de texto também fornecido na declaração. Como esse padrão é descrito por uma fórmula sintática poderosa,essa linguagem extensora, funciona como um meio de se programar a tradução de uma linguagem em outra.

O compilador da linguagem extensora "E", que leia um fonte da linguagem alterada e produza outro da linguagem original, pode ser encarado como um pré-processador de fontes com recursos programáveis, ou seja, é uma única peça de software que manipula diversas alterações de uma linguagem.

ABSTRACT

The existing programming languages, either standardised or not, are frequently being altered by its designers or users.

The extensions proposed to a language may involve mechanisms of creating new features, or new types of declaration such as declarations of macros, or equivalent, which are in itself resources for including new extensions.

The language "E" is here proposed as an adequate mechanism for implementing extensions to a variety of programming languages.

Once a language has been adapted to include the language "E", it is possible to implement extensions to that language by means of declarations of text replacement to be performed. These declarations, programmed by the user, contain a pattern and the corresponding replacement text. The source program is searched for occurrences of the pattern, that are replaced in line by the replacement text. The patterns are written by means of powerful syntatic formulae, giving this language "E" the capability of translating one language into another.

The language "E" compiler, receiving a source file containing the extended language as input and producing an original language file as output, may be thought as a programmable source file pre-processor, i.e., an unique piece of software which can handle several alterations to a language.

I INTRODUÇÃO 1
I.1 Linguagens de programação alteradas 1
I.2 Mecanismos de alterações 2
I.3 A linguagem extensora "E" 3
I.4 Características da linguagem "E" e de seu processador 5
I.5 Descrição da linguagem e do processador 6
I.6 Notações 7

II MACROS E EXTENSIBILIDADE 9
II.1 Linguagem base, meta-linguagem e linguagem derivada 12
II.2 Técnicas de extensão 13
II.3 Declarações 16
II.4 Identificadores 19
II.5 Renomeações 22

III A LINGUAGEM EXTENSORA "E" E SUA APLICAÇÃO A LINGUAGEM LPS 23
III.1 Programa "E" 25
III.2 Declaração-de-substituição 27
III.3 Texto 30
III.4 Parâmetros 35
III.5 Alternativas 41
III.6 Conjunto-de-sintaxes 53
III.7 Sintaxe 55
III.8 Padrão-inicial 55
III.9 Teste-de-alternativa 59
III.10 Indicador-de-cláusula 59

IV IMPLEMENTAÇÃO NO COBRA-530 60
IV.1 Notações 62
IV.2 Reconhecedores 62
IV.3 Estruturas básicas 65
IV.4 Reconhecimento das estruturas básicas do LPSE 67
IV.4.1 Objetos usados no reconhecimento 69
IV.4.2 Sub-rotina LE.TOKEN 69
IV.4.3 Função TOKEN 69

IV.4.4 Sub-rotina ERRO 70
IV.4.5 Sub-rotina EXIJA 70
IV.4.6 Rotinas dos não-terminais 70
IV.4.7 Atributos 74
IV.5 Reconhecimento de Trechos-substituíveis 87
IV.5.1 Geração da meta-linguagem intermediária 88
IV.5.1.1 Trechos-substituíveis 89
IV.5.1.2 Sintaxe 90
IV.5.1.3 Ítem 90
IV.5.1.4 Cláusula 91
IV.5.1.5 Grupo 92
IV.5.1.6 Conjunto-de-sintaxes 93
IV.5.2 Implementação das árvores sintáticas 93
IV.5.3 Atributos sintáticos dos nós 100
IV.5.4 Tabela de declarações 101
IV.5.5 Algoritmo de reconhecimento de Trechos-substituíveis 103
IV.5.6 Pesquisa de início válido 109
IV.5.7 Término de parâmetro efetivo 111
IV.6 Substituição 117
IV.6.1 O SUBSTITUIDOR 119
IV.6.2 Instruções 121
IV.6.2.1 Entrada e Saída 121
IV.6.2.2 Trecho-substituível 122
IV.6.2.3 Endereços indiretos e Desvios 126
IV.6.2.4 Parâmetro 128
IV.6.2.5 Testes e Grupos 132

V RESUMO DA LINGUAGEM LPSE 135
V.1 sequência-de-símbolos-léxicos 135
V.1.1 símbolo-léxico 135
V.1.2 palavra 135
V.1.3 identificador-de-parâmetro 136
V.1.4 símbolo-reservado 136
V.2 Programa LPSE 136
V.2.1 Declaração-de-substituição 138
V.2.2 Texto 139
V.3 Padrão-inicial 140

V.3.1 Cláusula-sem-parâmetro 141
V.3.2 Indicador-de-parâmetro 141
V.3.3 Cláusula-com-parâmetro 142
V.3.4 Sintaxe 143
V.3.5 Conjunto-de-sintaxes 144
V.3.6 Alternativas 145
V.3.7 Grupo-obrigatório-não-repetitivo 146
V.3.8 Grupo-obrigatório-repetitivo 146
V.3.9 Grupo-opcional-não-repetitivo 147
V.3.10 Grupo-opcional-repetitivo 147
V.3.11 Terminador 149
V.4 Padrão-final 149
V.4.1 Elemento-insubstituível 150
V.4.2 Trecho-substituível 150
V.4.3 Referência-a-parâmetro 153
V.4.4 Nome-de-cláusula 154
V.4.5 Indicador-de-cláusula 154
V.4.6 Teste-de-alternativa 154
V.4.7 Default 155
V.4.8 Teste 156

VI CONCLUSÕES 157

Bibliografia 159

CAPÍTULO I

INTRODUÇÃO

I.1 Linguagens de programação alteradas

As linguagens de programação, mesmo as que possuem padrões publicados e registrados, estão sempre sendo alteradas pelos fabricantes de software, pesquisadores e até por seus usuários. Parte dessas alterações visa a adequar a linguagem à máquina específica para a qual o seu compilador gerará código, seja introduzindo novas características na linguagem, para que recursos da máquina possam ser utilizadas, ou cortando características incompatíveis ou ineficientes nessa máquina. Os fabricantes de compiladores também modificam as linguagens para que seus compiladores e os programas objetos por eles produzidos sejam eficientes.

As linguagens alteradas são dependentes dos compiladores que reconhecem essas alterações e das máquinas-alvos desses compiladores. Para que a portabilidade dos programas-fontes seja recuperada, outros compiladores são alterados para que suportem essas tais modificações.

As linguagens também sofrem alterações orientadas para uma classe de problemas que seus programas devem tratar. Se uma linguagem tem apenas características voltadas para uma classe de problemas (entrada de dados, tempo real, processamento comercial ou cálculo por exemplo), logo se faz notar a carência de outras características próprias de linguagens mais gerais. Se ela tenta conciliar o geral com o específico, nunca o consegue de um modo sintético, compactado, próprio para uma programação fácil e compilação eficiente. Tudo isso é fonte de extensões nas linguagens de programação.

O progresso no desenho das linguagens de programação também é um causador de modificações. Linguagens velhas recebem desenho novo para permitirem programações estruturadas, acesso a bancos

de dados, dados estruturados em registros, tipos abstratos etc. E a cada dia que passa, em menos tempo uma linguagem nova se torna velha.

Resumindo, praticamente todas as linguagens de programação, cedo ou tarde, são alteradas. A linguagem extensora "E" é aqui proposta como um mecanismo para a programação de extensões, facilmente adaptável às linguagens de programação usuais. A adaptação da linguagem extensora "E" à linguagem de programação "LPS", gerando a linguagem extensível "LPSE", e sua implementação no computador COBRA-530, também estão descritas aqui.

I.2 Mecanismos de alterações

O mecanismo mais radical para a alteração de uma linguagem é um novo compilador para essa linguagem alterada. Se, por uma lado, isso permite modificações audazes, essas modificações são permanentes, não servindo esse mecanismo para introduzir novas modificações adicionais. A feitura de compiladores não é um trabalho de programação simples. O possuidor de um sistema de geração de compiladores teria esse trabalho simplificado, mas ainda encontraria o inconveniente de ter que trocar uma peça de software considerável, como é um compilador, a cada vez que quisesse introduzir alterações na linguagem compilada.

O mecanismo menos radical nem chega a alterar linguagens, apenas disciplina seus usuários e fornece ferramentas dirigidas para classes de problema. Consiste na exploração do potencial de declarações de uma linguagem e na criação de bibliotecas com essas declarações visando a um propósito. O poder desse mecanismo depende da linguagem empregada. Se for FORTRAN, as bibliotecas poderão contar com funções, sub-rotinas, áreas COMMON etc. Mesmo que a linguagem não possua mecanismos de programação em separado, essas bibliotecas podem ser construídas se for disponível um editor de texto que permita a inserção de trechos arquivados nessa biblioteca, no texto do programa-fonte.

Com uma linguagem como ADA, por exemplo, extremamente geral, mas que permite uma super-exploração desse processo, criando-se bibliotecas de funções, sub-rotinas, pacotes, tasks, paco-

tes genéricos etc, esse mecanismo torna-se poderoso. Através da criação de bibliotecas, os usuários passam a ter ferramentas padronizadas e disponíveis para tratar determinadas classes de problemas. Dependendo da linguagem e do usuário, esse mecanismo pode ser o suficiente para adaptar a linguagem às necessidades do usuário. Entretanto, esse mecanismo, sempre disponível, até hoje não evitou que sejam feitas alterações nas linguagens.

Um dos principais incovenientes, que algumas linguagens de programação induzem nesse sistema de bibliotecas, aparece quando não se dispõe de chamadas a rotinas que produzam código "em linha", isto é, sem nenhum processamento adicional de passagem de parâmetros, desvios e retornos. Macros em linguagens montáveis, macros-sintáticos e rotinas "in-line" produzem esse tipo de código.

Um mecanismo de alteração intermediário é o pré-processamento do fonte. Neste método, uma peça de software transforma o fonte escrito na linguagem alterada em um fonte na linguagem original, apto a ser processado por um compilador padrão dessa linguagem. Esse procedimento é largamente usado das mais diversas maneiras. Existem desde pré-processadores específicos para uma linguagem alterada, até os que permitem a programação da tradução de uma linguagem qualquer para outra, evidentemente sujeitas a restrições. O ônus causado pelo pré-processamento do fonte é o principal inconveniente desse tipo de mecanismo. Como esse mecanismo mantém intacto o compilador, ele se apresenta vantajoso.

Seja qual for o mecanismo empregado, se a alteração na linguagem cria recursos para novas adaptações, criando novos tipos de declarações, em particular declarações de macros ou dispositivos equivalentes, essa alteração em si também é um mecanismo que permite novas modificações na linguagem.

I.3 A linguagem extensora "E"

A linguagem extensora "E" é um mecanismo para a programação de extensões, adaptável às linguagens de programação. Ela não é uma linguagem completamente definida, duas de suas classes gra-

maticais (não-terminais) têm definições em aberto. A adaptação da linguagem "E" à uma linguagem consiste em definir esses não-terminais "palavra" e "identificadores-de-parâmetro" de acordo com a linguagem hospedeira. O não-terminal "palavra" representa os elementos léxicos ("tokens") da linguagem hospedeira e o não-terminal "identificador-de-parâmetro" segue a sintaxe dos identificadores dessa linguagem, apenas diferindo deles por um caráter acrescentado no início desses identificadores.

A linguagem extensora "E" permite que se programe extensões para outra linguagem a ela adaptada, através de declarações de substituição de textos. Em cada uma dessas declarações, o programador fornece um padrão que, toda vez que for reconhecido no texto, é substituído por um outro trecho de texto, também fornecido na declaração. Como esse padrão é descrito por uma fórmula sintática poderosa,essa linguagem extensora, funciona como um meio de se programar a tradução de uma linguagem em outra.

O processador da linguagem extensora "E", que leia um fonte da linguagem alterada e produza outro da linguagem original, pode ser encarado como um pré-processador de fontes com recursos programáveis, ou seja, é uma única peça de software que traduz diversas alterações de uma linguagem. Porém, ele pode ser incorporado ao compilador da linguagem original,de modo a fornecer como saída não um fonte da linguagem alterada e sim a própria saída que o analisador léxico desse compilador fornecia. Desse modo, ao invés de um pré-processador de fontes, teria-se um processador de elementos-léxicos ("tokens").

De qualquer modo, o próprio analisador léxico original pode ser aproveitado na feitura do processador da linguagem extensora "E".

Um conjunto de declarações de substituições, que definam uma determinada alteração em uma linguagem, pode ser arquivado em uma biblioteca para ser utilizado por todos que quiserem compilar nessa linguagem alterada. Para construir uma nova alteração, bastaria produzir um novo conjunto de declarações para ser usado no lugar daquele ou complementando-o.

I.4 Características gerais da linguagem "E" e de seu processador

O esquema de tradução de fontes foi adotado aqui por ser desvinculado do compilador e permitir a programação de novas alterações. O fato do processador da linguagem "E" ser uma peça de software independente do compilador, apesar de seu relacionamento íntimo, permite otimizações e extensões na própria linguagem "E" e no seu processador. Entretanto, esse esquema de tradução é menos eficiente que a feitura de um novo compilador para a linguagem alterada. Por isso, foram diretrizes do projeto da linguagem "E" e do seu processador:

a) eficiência. Toda característica da linguagem ou do processador que comprometesse a eficiência deveria ser excluída.

b) versatilidade. As características da linguagem e do processador deveriam servir ao maior conjunto de aplicações possível.

c) facilidade de programação pelo usuário.

d) independência do compilador.

e) alterabilidade. A linguagem e o processador devem permitir extensões.

f) implementação não muito complexa.

Algumas características principais surgem dessas diretrizes:

a) o processador não deve ser induzido a realizar "back-tracking", isto é, ao tentar reconhecer um padrão e verificar que ele não pode ser reconhecido, o processador não deve voltar símbolos para tentar reconhecer outros padrões. Isto significa que a cada momento o processador só está tentando reconhecer no máximo um único padrão escolhido pelo seu primeiro

símbolo. Outros reconhecimentos poderiam estar suspensos esperando o término do reconhecimento desse padrão.

b) a tradução de uma linguagem alterada para a sua original deve ser eficiente. A tradução de uma linguagem qualquer para outra não é o objetivo primordial.

c) as fórmulas que descrevem os padrões a serem reconhecidos devem ser poderosas e legíveis.

d) as tabelas que dirijam o reconhecimento dos padrões devem ser produzidas diretamente da análise das fórmulas que descrevem esses padrões, sem cálculos adicionais para a obtenção dessas tabelas.

e) deve receber elementos léxicos como entrada e produzir elementos léxicos como saída.

f) deve ser implementado e adaptado ao compilador LPS no computador COBRA-530.

I.5 Descrição da linguagem e do processador

O capítulo II recapitula noções sobre Macros e Extensibilidade que facilitarão a compreensão dos demais capítulos.

O capítulo III discute a linguagem extensora "E" e sua adaptação à linguagem LPS.

O capítulo IV descreve a implementação do processador da linguagem "E" adaptado ao compilador LPS.

O capítulo V resume a linguagem extensível LPSE, que é a própria linguagem "E" adaptada à LPS. Esse capítulo é um breve manual de referência da LPSE.

O capítulo VI discute os resultados obtidos e analisa possibilidades de extensão.

I.6 Notações

As sintaxes livres-de-contexto das linguagens são descritas aqui através de uma simples variante da Forma Backus-Naur (BNF). Em particular:

(a) palavra composta de letras minúsculas, algumas vezes contendo hífens, denota categoria sintática do nível léxico, por exemplo

sequência-de-símbolos-léxicos
símbolo-léxico
palavra
símbolo-reservado
identificador-de-parâmetro
cadeia-de-caracteres

(b) palavra iniciada por letra maiúscula seguida de letras minúsculas, algumas vezes contendo hífens, denota outras categorias sintáticas, por exemplo

Elemento-insubstituível
Trecho-substituível
Texto
Cláusula-sem-parâmetro

(c) símbolos entre aspas representam os próprios símbolos como terminais, por exemplo

"ZERE.X"
"("
")"
"macro"
"..."
"||"

(d) uma barra vertical separa ítens alternativos, por exemplo

```
símbolo-léxico ::= palavra | símbolo-reservado |
                   identificador-de-parâmetro
```

(e) os sinais "*", "+" e "?" indicam que o ítem que o precede pode aparecer zero ou mais vezes, uma ou mais vezes e zero ou uma vez, respectivamente. Por exemplo

sequência-de-símbolos-léxicos ::= símbolo-léxico+

Elemento-insubstituível ::= "ORIGINAL"? palavra

Cláusula ::= palavra*

(f) qualquer categoria sintática sufixada por um travessão e um número ou outro símbolo é equivalente à categoria sintática sem sufixo correspondente. O sufixo apenas serve como diferenciador. Por exemplo

Ítem_1 e Ítem_2 são equivalentes a Ítem.

As sintaxes dependentes-do-contexto (semântica estática) e os esquemas de tradução (semântica dinâmica) das linguagens são descritos aqui por gramáticas de atributos com notação baseada na notação anterior, acrescida das seguintes regras:

(g) atributos também são descritos por palavras compostas de letras minúsculas, algumas vezes contendo hífens, como no ítem (a). Por exemplo

substituído

nome-do-parâmetro

(h) atributos sintetizados por um não-terminal podem ser denotados precedidos pelo nome do não-terminal e pelo símbolo "^". Por exemplo

Sintaxe^tem-nome

Ítem^termina-com-palavra

Programa^substituído

(i) atributos herdados por uma não-terminal podem ser denotados precedidos pelo nome do não-terminal e pelo símbolo "↓". Por exemplo

Grupo↓obrigatório

Cláusula↓nome

CAPÍTULO II

MACROS E EXTENSIBILIDADE

A repetição de trechos em diversas partes de um programa é problema comum e tedioso para o programador de computadores. Uma das soluções empregada por eles é a declaração de rotinas, nas quais, aos parâmetros formais definidos em tempo de compilação, são atribuídos os parâmetros efetivos calculados em tempo de execução. Entretanto, essa solução não satisfaz se o trecho não suficientemente grande para compensar o prejuízo de tempo e espaço gastos nos mecanismos de chamada à rotina e passagem de parâmetros.

Exemplo II.1:

Em um programa escrito em LPS, que envolva o uso repetido de pilhas, frequentemente há necessidade de se colocar um ítem na pilha e incrementar o ponteiro para o topo. Nessa ocasião pode ser necessário escrever:

```
PILHA(TOPO) := ITEM;
TOPO := TOPO + 1;
```

E em outra:

```
PILHA1(TOPO1) := X;
TOPO1 := TOPO1 + 1;
```

Poder-se-ia ter uma rotina que manipulasse as pilhas, porém, isto poderia ser bem ineficiente, devido a passagem de parâmetros e trocas de contexto. O necessário, neste caso, é um método simples para que quando um comando do tipo

```
EMPILHE (PILHA, TOPO, ITEM);
```

fosse encontrado pelo compilador, produzisse o mesmo códi-

go que produziria o trecho

```
PILHA (TOPO) := ITEM;
TOPO := TOPO + 1;
```

se o programador o tivesse escrito neste ponto. Se esse comando fosse uma chamada-a-rotina, não serviria, pois geraria um processamento adicional na chamada, no retorno da rotina e na passagem de parâmetros.

Problemas como este deram origem às idéias dos processadores de macros e das linguagens extensíveis. Com o tempo estas idéias evoluíram para bem longe deste início.

A idéia de usar-se processadores de macros como complementos das linguagens de programação já tem uma idade considerável. Greenwald, em 1959, e McIlroy, em 1960, entre outros, publicaram artigos sobre o assunto, mas as macros já estavam em uso pelos fabricantes de computadores, principalmente em conjunção com as linguagens montáveis.

Diversas peças de software foram então desenvolvidas e chamadas de processadores de macro. Por isso, é difícil dar-se um definição formal de processador de macros que sirva para todas elas. A idéia de substituição de trechos de texto, entretanto, é uma propriedade inerente aos processadores de macros.

Brown, em 1969, definiu processador de macros como "uma peça de software desenhada para permitir ao usuário adicionar facilidades desenhadas por ele próprio a uma peça de software existente". Esta definição é tão geral que não cabe apenas para processador de macros, serve para qualquer mecanismo que permita estender o conjunto de facilidades proporcionadas por uma peça de software. Processadores de uma linguagem de programação, por exemplo, são peças de software, em geral mais de uma, que executam instruções contidas em textos, que lhes servem de entrada, contando que estes textos pertençam à linguagem de programação tratada pelo processador (compilador, interpretador, carregador, bibliotecas de rotinas intrínsecas, editor de ligações, etc). Para acrescentar novas facilidades a esse processador é necessário alterar-se a sua linguagem de programação, pois ela lhe for-

nece os comandos a serem executados.

Extensibilidade é a capacidade que usuários têm para definir novas características numa linguagem. Uma linguagem é dita extensível se ela for composta por uma linguagem base e por facilidades de definição que permitam ao usuário criar novas notações, novas operações, novas estruturas de dados etc.

Seu usuário pode criar extensões na linguagem bem adaptadas para sua área de aplicação, além de permitir a escrita de algoritmos concisos, claros e livres de detalhes de baixo-nível. Para tal, a implementação de mecanismos de extensibilidade deve ser eficiente e eficaz.

II.1 Linguagem base, meta-linguagem e linguagem derivada

Na abertura do Simpósio para Linguagens Extensíveis de maio de 1969, Christensen caracterizou extensibilidade pelos seus objetivos:

"O objetivo ideal e final das linguagens extensíveis é simples e atraente. Um sistema de programação universal e único é postulado e deve se incluir como apoio de software para todo computador de propósito geral. Este sistema de programação não é limitado a uma linguagem de programação particular, como PL/I. Melhor, ele incui uma linguagem básica e uma meta-linguagem. Um programa neste sistema consiste de, primeiro, comandos na meta-linguagem que expandam, contraiam ou modifiquem as definições da linguagem básica para produzir uma linguagem derivada e, segundo, comandos na linguagem derivada que constituem a parte executável do programa.

Desse modo o sistema inclui facilidades para definir e depois programar numa variedade sem limites de linguagens de programação - linguagens usadas para aplicações comerciais, ciências ou outras aplicações, que podem ser simples ou complexas."

II.2 Técnicas de extensão

Standish dá alguns exemplo para ilustrar diversas técnicas de extensão:

Paráfrase é uma forma de definirmos uma nova entidade por intermédio de regras de substituição dessa entidade por outras entidades conhecidas (ou que serão conhecidas após futuras definições). Isto é usado comumente nas linguagens naturais para definir o significado de novas palavras (exemplo: Galactoscópio = instrumento com o que se examina a pureza do leite). Paráfrase é usada abundantemente como a principal técnica para linguagens extensíveis, e ela toma várias formas, com resultados finais que podem ser diferentes em cada caso. Por exemplo:

(a) macros numa linguagem montável:

```
MACDEF  SOME A, B, C
LOAD    A
ADD     B
STORE   C
ENDDEF
```

(b) declarações de rotinas em linguagens algébricas:

```
function FATORIAL (N:INTEGER) return INTEGER is
   begin
     if N = 0 then
        return 1;
     else
        return N * FATORIAL (N-1);
     end if;
   end FATORIAL;
```

(c) macros sintáticas:

```
statement-macro WHILE (B: boolean expression )
                      DO (S: compound statement )
define  create new label (L1) in
                L1 : if B then
                        begin
                          S;
                          goto L1
                        end
```

(d) definições de tipos de dados. Criação de tipos de dados compostos a partir de tipos de dados atômicos (integer, character, real):

```
type RACIONAL is
   record
     NUMERADOR   : INTEGER;
     DENOMINADOR : INTEGER;
   end record;

MATRIZ : array 1.. N of VETOR;
```

(e) definições de operações.

```
function "+" (A, B : RACIONAL) return RACIONAL;
```

(f) extensões das estruturas de controle. Aqui confere-se atributos aos processos em tempo de sua definição. Usado para definir processos que ajam como co-rotinas, que executem concorrentemente, que monitorem ocorrências de certos eventos etc.

Ortofrase permite adicionar características ortogonais à linguagem. Uma características ortogonal é uma que cai fora do espaço das características expressíveis por paráfrase. Sua definição não pode ser expressa na linguagem. Adicionar um sistema de arquivamento, um relógio de tempo-real ou passagem de parâmetros por referência são exemplos de características que muitas

vezes não podem ser definidas por paráfrase se as bases para defini-las não estiverem na linguagem.

Metafrase, ao contrário da paráfrase e da ortofrase, que adicionam novas capacidades sem alterar as que já existiam, consiste em alterar as regras de interpretação de uma linguagem para que ela processe velhas expressões por um novo modo. Mudanças nas políticas de definição do alcance de variáveis, mudança no siginificado da avaliação de parâmetros e a mudança do sentido das atribuições e desvios para permitir a um programa rodar de trás para frente são exemplos de Metafrase.

Nos casos da ortofrase e da metafrase, normalmente tem-se que modificar a descrição do processador da linguagem. Essas descrições são complexas e em geral só devem ser feitas por pessoal especificamente qualificado.

II.3 Declarações

Nas linguagens de programação existem maneiras de se definir novas entidades a partir de outras, através de declarações. Portanto, pode ser dito que declarações estendem linguagens por paráfrase.

Declaração é o modo com o qual praticamente a totalidade das linguagens de programação permite ao programador estabelecer novas fórmulas gramaticais (produções) em extensão à gramática inicial da linguagem. Esta fórmula associa uma sintaxe a uma semântica estática e/ou dinâmica.

Exemplo II.2: (LPS)

```
. . .
procedure ZERE.X (integer I);
   begin
     X(I):=0;
   end ZERE.X;
. . .
ZERE.X(A+B);
. . .
```

A declaração de rotina ZERE.X produz, para o compilador, o mesmo efeito que seria produzido se a produção sintática

Chamada-a-subrotina ::= "ZERE.X" "(" Expressão ")"

fosse adicionada pelo compilador à gramática que define a linguagem LPS ao reconhecer a Declaração-de-rotina correspondente. Essa produção sintática tem uma semântica associada.

Exemplo II.3: (LPS)

```
. . .
constant ENDERECO.FINAL = #2FFFF;
. . .
constant TAMANHO = ENDERECO.FINAL + 1;
. . .
```

O compilador "adiciona", com as respectivas semânticas,

Identificador-de-constante ::= "ENDERECO.FINAL"
Identificador-de-constante ::= "TAMANHO"

Exemplo II.4: (LPS)

```
integer procedure Z;
begin
  . . .
  Z := 0;
  . . .
end Z;
  . . .
X := Z;
```

O compilador acrescenta a produção

Identificador-de-variável ::= "Z"

válida apenas no interior da Declaração-de-subrotina, e

Chamada-a-função ::= "Z"

válida após a Declaração-de-função até o fim do Bloco que possui essa Declaração.

As fórmulas gramaticais acrescentadas pelo compilador têm alcance limitado, definido pelas regras de alcance (escopo) da linguagem. De qualquer modo o alcance máximo que uma declaração pode ter é o programa na qual ocorreu, não afetando outros programas que venham a ser compilados. Portanto, declarações estendem a linguagem temporariamente.

Algumas linguagens, ADA, por exemplo, permitem a um programa reconhecer declarações de um outro programa previamente especificado. Se a linguagem permitir isso, uma biblioteca de declarações é uma maneira de estender a linguagem.

II.4 Identificadores

Em geral, a sintaxe introduzida por uma declaração tem, pelo menos uma palavra "inventada" pelo programador e que identificará a entidade declarada, isto é, a difenciará de outras sintaxes semelhantes. Se houver apenas uma dessas palavras, esta será o IDENTIFICADOR da entidade declarada.

No exemplo II.2, o identificador era ZERE.X.

No exemplo II.3, o identificador era ENDERECO.FINAL na primeira declaração e TAMANHO na segunda;

No exemplo II.4, o identificador era Z nas duas entidades declaradas, não havendo confusão por que as duas têm alcance disjuntos.

Exemplo II.5: (ADA)

```
TABELA: array (1..10,1..100) of INTEGER;
```

É acrescentada a produção

```
Indexed-component
     ::="TABELA" "("expression(",",expression)*")"
```

cuja sintaxe foi delineada pela linguagem ADA na própria declaração (Object-declaration). Isto sem mencionar que os atributos semânticos dos não-terminais também foram delineados na Object-declaration que é o nosso exemplo.

Podemos então dizer que uma linguagem de programação contém uma meta-linguagem com a qual é estendida a linguagem de programação original durante o processamento de um programa dessa linguagem.

Uma das principais limitações quanto a esse tipo de extensibilidade nas linguagens de programação é a sintaxe rígida introduzida pelas declarações. O programador pouco pode interferir nessa sintaxe. Em geral pode apenas escolher o identificador fi-

dor ficando praticamente todo resto da sintaxe estabelecido rigidamente pela linguagem. E o pior é que os elementos originais da linguagem (comandos, declarações, rotinas) não têm essa mesma forma, exatamente por ser limitada. É preferida para elas uma estrutura de frase.

A linguagem ADA apresenta uma série de inovações que buscam diminuir essas limitações, tais como parâmetros de rotinas opcionais, rotulados e não-posicionais. Entretanto essa mesma linguagem sente a necessidade de um comando "case" no qual um pedaço de sua sintaxe pode ocorrer um número indeterminado de vezes no mesmo comando. E essa linguagem não admite extensões que tenham esse mesmo problema, ou seja, uma rotina não pode ter um parâmetro que possa ocorrer um número indeterminado de vezes numa mesma chamada.

Exemplo II.6: (LPS)

```
      . . .
integer procedure MIN (integer P1, P2);
      begin
           MIN := if P1 < P2 then P1 else P2;
      end;
      . . .
integer X, Y, Z, MENOR;
      . . .
&
& calculo do menor valor entre X, Y e Z:
&
      MENOR := MIN (X,MIN(Y,Z));
      . . .
```

Apesar da função MIN ser associativa, podendo ser definida para uma conjunto com um número qualquer de elementos, foram necessárias duas chamadas a ela para calcular o menor valor entre X, Y e Z. Não é permitido chamar essa função com mais de dois parametros (MIN(X, Y, Z)).

As linguagens de programação também costumam ter comandos com cláusulas alternativas, mas as chamadas de rotinas não.

Exemplo II.7: (LPS)

```
for X:=0 to 10 do
    . . .
for X:=10 downto 1 do
    . . .
```

A linguagem ADA tem o recurso de fornecer valores para os parâmetros formais na falta (default) dos respectivos parâmetros efetivos. Isso, porém, implica que no corpo da rotina não se tenha idéia sobre a ausência ou não de um parâmetro efetivo, não podendo então essa ausência ter um valor semântico que pudesse ser testado. Essa linguagem, com todos os progressos, não liberou as chamadas a rotinas dos parênteses envólucro dos parâmetros.

Exemplo II.8: (LPS)

```
X := MIN(A,MAX(B,MIN(C,D)));
```

Há uma verdadeira cascata de parênteses ao usar funções nos parâmetros efetivos.

Exemplo II.9: (LPS)

```
if X = Y then
     if X = Z then
          if X = W then
               x := 0;
```

Em LPS, assim como em ALGOL, o ";" é um separador de comandos. Nesse exemplo ele termina todos os comandos IFs, não havendo cascata de parênteses devido a estrutura de frases.

II.5 Renomeação

Exemplo II.10: (LPS)

fonte original	fonte equivalente
. . .	
byte X;	byte X;
. . .	. . .
constant C1=10+C;	
. . .	. . .
procedure Z;	
begin	
X:=C1;	
Y:=X+Y;	
end;	
. . .	. . .
X:=100;	X:=100;
Y:=C1;	Y:=10+C;
Z;	X:=10+C;
	Y:=X+Y;
. . .	. . .

A declaração da constante C1 e a declaração da subrotina Z podem ser eliminadas substituindo-se as referências a esses identificadores. A variável X tem uma declaração que não pode ser eliminada dessa maneira, pois ela também estabelece um "objeto variável byte", e não existe modo de substituir as referências a X através de outras construções da linguagem LPS.

Declarações que possam ser eliminadas substituindo-se as referências a essa declaração por construções da própria linguagem equivalente serão chamadas de renomeações.

CAPÍTULO III

A LINGUAGEM EXTENSORA "E"

E SUA APLICAÇÃO À LINGUAGEM LPS

Nesse capítulo é definida a linguagem extensora "E", que permite a escrita de textos em uma outra linguagem (LPS, por exemplo), acrescida de extensões. Essa linguagem extensora será dotada de um único tipo de declaração, que por sua vez, produzirá apenas renomeações. Essa declaração será uma fórmula que dirá como substituir trechos de texto por outros.

Alguns aspectos da definição da linguagem extensora E ficam em aberto para que possam ser definidos de acordo com a linguagem hospedeira - linguagem que receberá extensões.

Desse modo não é definida a sintaxe das PALAVRAS, que são os constituintes dos textos. A idéia é que, alterando-se a definição sintática de palavra, seria possível usar a linguagem extensora "E" em textos de linguagens de programação diferentes. Por exemplo, para usarmos a linguagem "E" para estender a linguagem LPS, definiremos como palavra tudo que for um "token" LPS, ou seja: número, cadeia-de-caracteres, identificador, identificador-reservado e símbolo-especial. Ou seja, tudo que um "scanner" LPS produz como saída é considerado como PALAVRA para o LPSE (LPS + E). Para outras linguagens faríamos o mesmo.

Um programa "E" é uma sequência-de-símbolos-léxicos, embora nem toda sequência-de-símbolos-léxicos seja um programa "E".

Os símbolos-léxicos são: palavras, símbolos-reservados e identificadores-de-parâmetros.

```
sequência-de-símbolos-léxicos ::= símbolo-léxico+
símbolo-léxico ::= palavra | símbolo-reservado |
                   identificador-de-parâmetro
```

As palavras e os identificadores-de-parâmetro não terão suas sintaxes completamente definidas, para que possam se adaptar a linguagem hospedeira.

As PALAVRAS serão os "tokens" da linguagem hospedeira que não forem reservados para a linguagem "E" (símbolos-reservados e identificadores-de-parâmetros).

Os IDENTIFICADORES-DE-PARÂMETRO devem seguir a sintaxe dos identificadores da linguagem hospedeira, mas devem ser precedidos por um "$", para diferenciá-los destes.

Exemplo III.1:

Identificadores LPS	Identificadores-de-parâmetro LPSE
CODIGO	$CODIGO
T...	$T...
N1INF	$N1INF

Os símbolos-reservados têm significado especial na linguagem, não podendo serem usados como palavras não reservadas. Servem basicamente, para montar declarações. Símbolos-reservados que difiram apenas no uso de letras maiúsculas ou minúsculas correspondentes são considerados os mesmos.

símbolo-reservado ::= "macro" | "define" | "endmacro" | "opend"

| "stend" | "original" | "..." | "{" | "}"

| "[" | "]" | "|" | "||" | "$"

III.1 Programa "E"

Exemplo III.2: (LPSE)

```
macro INTEIRO
define
    INTEGER
endmacro

BEGIN
   INTEIRO X,Y,Z;
   X:=Y+Z;
END
```

Declaração-de-substituição

Texto

Programa

Programa^substituído:

```
BEGIN
    INTEGER X,Y,Z;
    X:=Y+Z;
END
```

No exemplo III.2, o Programa é um Texto precedido por uma única Declaração-de-substituição. Programa^substituído representa o Programa após a execução das substituições declaradas.

Exemplo III.3: (LPSE)

```
macro PROG
  define
    BEGIN
  endmacro

macro FIM
  define
      END
  endmacro

macro TABELA
  define
      INTEGER (10)
  endmacro

PROG
   TABELA T1,T2;
   T1(0):=T2(0);
FIM
```

Declaração-de-substituição

Declaração-de-substituição

Declaração-de-substituição

Texto

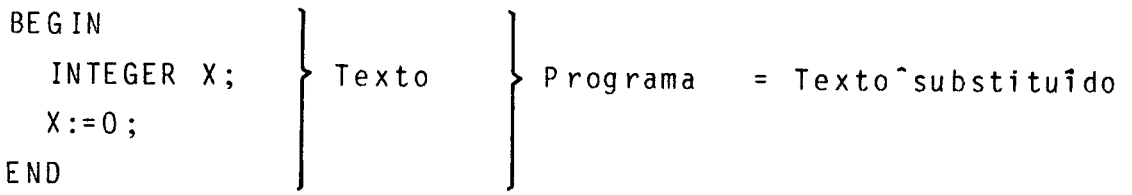
Programa

Programa substituído:

```
BEGIN
    INTEGER (10) T1,T2;
    T1(0):=T2(0);
END
```

No exemplo III.3, o Programa é composto de um Texto precedido por três Declarações-de-substituição. No exemplo III.4, o Programa compõem-se apenas do Texto, não havendo Declarações-de-substituição.

Exemplo III.4: (LPSE)

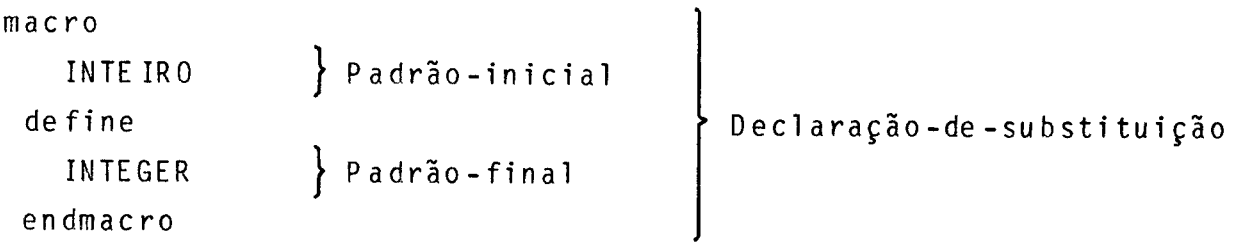


Programa é um Texto precedido por Declarações-de-substituição, que indicam as substituições a serem feitas nesse Texto.

Programa ::= Declaração-de-substituição* Texto

III.2 Declaração-de-substituição

Exemplo III.5: (LPSE)



Exemplo III.6: (LPSE)

```
macro                                          ┐
   TROQUE X COM Y     } Padrão-inicial         │
 define                                        │
   BEGIN                ┐                     │
     INTEGER Z;         │                     } Declaração-de-
     Z:=X;              } Padrão-final        } Substituição
     X:=Y;              │                     │
     Y:=Z;              │                     │
   END                  ┘                     │
 endmacro                                      ┘
BEGIN
   INTEGER X,Y;
   X:=1; Y:=2;
   TROQUE X COM Y
END
```

Programa substituído:

```
BEGIN
   INTEGER X,Y;
   X:=1; Y:=2;
   BEGIN
      INTEGER Z;
      Z:=X;
      X:=Y;
      Y:=Z;
   END
END
```

Uma Declaração-de-substituição inicia com o símbolo-reservado "MACRO" e termina com "ENDMACRO". É composta por um Padrão-inicial e um Padrão-final separados pelo símbolo-reservado "DEFINE".

Declaração-de-substituição ::=
"MACRO" Padrão-inicial "DEFINE" Padrão-final "ENDMACRO"

Esta declaração acrescenta uma regra gramatical à linguagem "E". Essa regra vale à partir do "DEFINE" da própria Declaração-de-Substituição e permanece válida até o fim do Programa.

O Padrão-inicial é uma fórmula que fornece a sintaxe da regra acrescentada.

O Padrão-final indica como será feita a substituição dos trechos de texto que se encaixarem com a sintaxe do Padrão-inicial.

Sempre que essa sintaxe for reconhecida num trecho de texto, será feita a substituição desse trecho conforme indicado no Padrão-final.

III.3 Texto

Exemplo III.7: (LPSE)

```
macro
   VETOR DE INTEIRO
 define
   INTEGER (*)     & vetor com dimensão em aberto
 end macro
BEGIN                                          }
   VETOR DE INTEIRO X = 10:0;                  }
   VETOR DE INTEIRO Y = 5:0;                   }  Texto
   X(3) := Y(1)                                }
END                                            }
```

Texto

Elemento-insubstituível	BEGIN
Trecho-substituível	VETOR DE INTEIRO
Elemento-insubstituível	X
Elemento-insubstituível	=
Elemento-insubstituível	10
Elemento-insubstituível	:
Elemento-insubstituível	0
Elemento-insubstituível	;
Trecho-substituível	VETOR DE INTEIRO
Elemento-insubstituível	Y
Elemento-insubstituível	=
Elemento-insubstituível	5
Elemento-insubstituível	:
Elemento-insubstituível	0
Elemento-insubstituível	;
Elemento-insubstituível	X
Elemento-insubstituível	(
Elemento-insubstituível	3
Elemento-insubstituível	)
Elemento-insubstituível	:=
Elemento-insubstituível	Y
Elemento-insubstituível	(
Elemento-insubstituível	1
Elemento-insubstituível	)
Elemento-insubstituível	END

Exemplo III.8: (LPSE)

```
macro
   INTEIRO
 define
   INTEGER
 endmacro
macro
   .
 define
   ;
 endmacro
BEGIN          ┐
   INTEIRO X.  │
   INTEIRO Y.  } Texto
   X:=Y        │
END            ┘
```

Texto	Elemento-insubstituível	BEGIN
	Trecho-substituível	INTEIRO
	Elemento-insubstituível	X
	Trecho-substituível	.
	Trecho-substituível	INTEIRO
	Elemento-insubstituível	Y
	Trecho-substituível	.
	Elemento-insubstituível	X
	Elemento-insubstituível	:=
	Elemento-insubstituível	Y
	Elemento-insubstituível	END

Um Texto é uma sequência, às vezes vazia, de Trechos-substituíveis e Elementos-insubstituíveis.

Texto ::= (Trecho-substituível | Elemento-insubstituível)*

As regras gramaticais para Trecho-substituível são geradas pelas Declarações-de-substituição.

Um Elemento-insubstituível é uma palavra que, ou não faz parte da sintaxe de qualquer Trecho-substituível, ou vem precedida pelo símbolo-reservado "ORIGINAL". Este símbolo indica que a palavra que o segue não deve ser considerada como fazendo parte da sintaxe de qualquer substituição.

Elemento-insubstituível ::= "ORIGINAL"? palavra

Exemplo III.9: (LPSE)

```
macro INTEIRO
  define INTEGER
  endmacro
BEGIN                                  }
   INTEIRO original INTEIRO;           } Texto
   original INTEIRO:=0;                }
END                                    }
```

Texto	Elemento-insubstituível	BEGIN
	Trecho-substituível	INTEIRO
	Elemento-insubstituível	original INTEIRO
	Elemento-insubstituível	;
	Elemento-insubstituível	original INTEIRO
	Elemento-insubstituível	:=
	Elemento-insubstituível	0
	Elemento-insubstituível	END

```
Texto substituído: BEGIN
                      INTEGER INTEIRO;
                      INTEIRO := 0;
                   END
```

Exemplo III.10: (LPSE)

```
macro VARIAVEL INTEIRA
 define INTEGER
 endmacro
BEGIN
   VARIAVEL INTEIRA original VARIAVEL, INTEIRA;  } Texto
   . . .
```

```
                 { Elemento-insubstituível      BEGIN
                 | Trecho-substituível          VARIAVEL INTEIRA
      Texto      { Elemento-insubstituível      original VARIAVEL
                 | Elemento-insubstituível      ,
                 | Elemento-insubstituível      INTEIRA
                 { Elemento-insubstituível      ;
                                                    . . .
```

```
                        BEGIN
Texto^substituído:      INTEGER VARIAVEL, INTEIRA;
                        . . .
```

Tanto Trecho-substituível, como Elemento-insubstituível, são formados por palavras. As ambiguidades são resolvidas pela seguinte regra: quando uma palavra puder fazer parte da sintaxe de um Trecho-substituível, ela fará parte desse Trecho a não ser que venha precedida por "ORIGINAL", nesse caso ela formará um Elemento-insubstituível.

Exemplo III.11: (LPSE)

```
macro
  SOME X COM Y        } Cláusula-sem-parâmetro } Padrão-inicial
 define
    X:=X+Y
 endmacro
BEGIN                              ⎫
  INTEGER X = 1, Y = 10;           ⎪
  SOME X COM Y;                    } Texto
  Y:=X                             ⎪
END                                ⎭
```

Uma Cláusula-sem-parâmetro é uma sequência de palavras num Padrão-inicial e indica que essas palavras aparecem em sequência na sintaxe atribuída a esse Padrão-inicial.

Cláusula-sem-parâmetro ::= palavra+

Toda Cláusula-sem-parâmetro tem um nome, que é a primeira palavra dessa cláusula.

No exemplo III.11, "SOME X COM Y", que aparece no Padrão-inicial, é uma Cláusula-sem-parâmetro cujo nome é "SOME". Essa mesma sequência aparece no Texto, sendo reconhecida como fazendo parte da sintaxe da regra introduzida pela Declaração-de-substituição.

Exemplo III.12: (LPSE)

```
macro SOME X COM Y define X:=X+Y endmacro
BEGIN INTEGER X,Y;   ⎫
  SOME Y COM X       } Texto
END                  ⎭
```

A sequência "SOME Y COM X" que aparece no Texto não é reconhecida como fazendo parte da sintaxe de um Trecho-substituível. Para tal, a sequência deveria ser idêntica à Cláusula-sem-parâmetro correspondente.

III.4 Parâmetros

Um Indicador-de-parâmetro ou é um identificador-de-parâmetro, ou é o símbolo-reservado "$".

Indicador-de-parâmetro ::= identificador-de-parâmetro | "$"

Exemplo III.13: (LPSE)

```
macro SOME $A COM $B;
  define
       $A := $A + $B;
  endmacro
BEGIN
  INTEGER I,J;
  SOME I COM J;
  SOME J COM I*3;
END
```

Padrão-inicial	Cláusula-com-parâmetro	SOME $A
	Cláusula-com-parâmetro	COM $B
	Cláusula-sem-parâmetro	;

Texto substituído:

```
BEGIN
  INTEGER I,J;
  I := I + J;
  J := J + I*3;
END
```

Uma Cláusula-com-parâmetro difere de uma sem parâmetro por ter um Indicador-de-parâmetro no final. Na posição correspondente na sintaxe gerada pelo Padrão-inicial será criado um parâmetro-formal. Graças às Cláusulas-com-parâmetro tem-se substitui-

ções parametrizadas.

Cláusula-com-parâmetro ::= palavra+ Indicador-de-parâmetro

Do mesmo modo que a Cláusula-sem-parâmetro, a primeira palavra da Cláusula-com-parâmetro é o seu próprio nome. Entretanto, uma Cláusula-com-parâmetro tem um outro atributo: o nome-do-parâmetro.

Exemplo III.14: (LPSE)

```
DOBRE $P

    Cláusula-com-parâmetro^nome = "DOBRE"
    Cláusula-com-parâmetro^nome-do-parâmetro = "P"
```

Exemplo III.15: (LPSE)

```
SOME $A

    Cláusula-com-parâmetro^nome = "SOME"

    Cláusula-com-parâmetro^nome-do-parâmetro = "A"
```

Exemplo III.16: (LPSE)

```
macro QUAD ($)
  define
     ($QUAD) * ($QUAD)
  endmacro
BEGIN
  INTEGER I,J;
  I:= 2 + QUAD (J+1);
  J:= QUAD (0)
END
```

```
                        { Cláusula-com-parâmetro     QUAD ( $
Padrão-inicial          {
                        { Cláusula-sem-parâmetro     )
```

```
Cláusula-com-parâmetro^nome                 = "QUAD"

Cláusula-com-parâmetro^nome-do-parâmetro = "QUAD"

Texto^Substituído: BEGIN
                       INTEGER I,J;
                       I := 2 + (J+1) * (J+1);
                       J := (0) * 0
                     END
```

Quando o Indicador-de-parâmetro for um identificador-de-parâmetro, o nome-do-parâmetro virá desse identificador-de-parâmetro; se for um "$", o nome-do-parâmetro será igual ao nome da cláusula.

Portanto, "$" é usado como Indicador-de-parâmetro, herdando o nome da cláusula para ser o seu nome-do-parâmetro.

Exemplo III.17: (LPSE)

```
macro COMP ($A, $B, $C)

                    Cláusula-com-parâmetro_1    COMP ( $A
                    Cláusula-com-parâmetro_2    , $B
  Padrão-inicial
                    Cláusula-com-parâmetro_ 3   , $C
                    Cláusula-sem-parâmetro_ 4   )

Cláusula-com-parâmetro_1 ^nome: "COMP"

Cláusula-com-parâmetro_2 ^nome: ","

Cláusula-com-parâmetro_3 ^nome: ","
```

Exemplo III.18: (LPSE)

```
macro SOME ($ ,$B )
  define
     $SOMA := $SOMA + $B          Padrão-final
  endmacro
BEGIN
  INTEGER I,J;
  SOMA ( I,J )                    Trecho-substituível
END
```

Padrão-final	Referência-a-parâmetro	$SOMA
	Texto	:=
	Referência-a-parâmetro	$SOMA
	Texto	+
	Referência-a-parâmetro	$B

Trecho-substituível		SOMA
		(
	Parâmetro-efetivo_$SOMA	I
		,
	Parâmetro-efetivo_$B	J
		)

```
Trecho-substituído:   BEGIN
                        INTEGER I,J;
                        I := I + J
                      END
```

Cada Cláusula-com-parâmetro acrescenta uma regra gramatical à linguagem "E", criando uma nova sintaxe para Referência-a-parâmetro. Essa regra será válida apenas no Padrão-final

da própria Declaração-de-substituição que contém essa Cláusula-com-parâmetro.

A sintaxe contida nessa regra sempre se refere a um identificador-de-parâmetro de mesmo nome-do-parâmetro que o da Cláusula-com-parâmetro.

Em cada substituição feita, um Trecho-substituível é trocado por um Padrão-final. As Referências-a-parâmetro que apareçam nesse Padrão-final são também substituídas pelos Parâmetros-efetivos correspondentes que apareçam nesse Trecho-substituível.

III.5 Alternativas

Exemplo III.19: (LPSE)

```
macro SE $ ENTAO $   [SENAO $];
  define
    BEGIN
      LABEL LSENAO,LFIM;
      IF NOT ($SE) THEN GOTO LSENAO;
      $ENTAO;
        {SENAO define GOTO LFIM;}
    LSENAO:
        {SENAO define $SENAO;}
    LFIM:
    END;
endmacro
BEGIN
  INTEGER I,J;
          SE I = J ENTAO I:=I+1;
          SE I = J ENTAO J:=0 SENAO J:=1;
END
```

No exemplo III.19, temos

```
Grupo-opcional:          [SENAO $]

Teste_1:                 {SENAO define GOTO LFIM;}

Teste_2:                 {SENAO define $SENAO;    }

Trecho-substituível_1:   SE I=J ENTAO I:= I+1;

Trecho-substituível_2:   SE I=J ENTAO J:=0 SENAO J:=1;

Trecho-substituível_1^Substituído:
                         BEGIN
                            LABEL LSENAO, LFIM;
                            IF NOT (I=J) THEN GOTO LSENAO;
                            I:= I+1;
                         LSENAO:
                         LFIM:
                         END;

Trecho-substituível_2^Substituído:
                         BEGIN
                            LABEL LSENAO, LFIM;
                            IF NOT (I=J) THEN GOTO LSENAO;
                            J:=0;
                            GOTO LFIM;
                         LSENAO:
                            J:=1;
                         LFIM:
                         END;
```

Um Grupo-opcional aparece num Padrão-inicial para indicar trechos opcionais na sintaxe desse Padrão-inicial. Um Trecho-substituível referente a esse Padrão-inicial pode ter ou não esses trechos opcionais

```
Grupo-opcional                ::= Grupo-opcional-não-repetitivo
                                | Grupo-opcional-repetitivo
Grupo-opcional-não-repetitivo::= "[" Alternativas "]"
```

Através de um Teste podemos ter pedaços do Padrão-final fazendo parte ou não de uma substituição, de acordo com a presença ou ausência de um trecho opcional definido num Grupo-opcional.

No exemplo III.19 aparecem dois Testes que definem pedaços do Padrão-final que farão parte de substituições em Trechos-substituíveis que contenham a cláusula de nome "SENAO". Em Trechos-substituíveis sem essa cláusula, esses pedaços não entram na substituição.

Exemplo III.20: (LPSE)

```
macro VARIANDO $ [DESCENDO] ATE $ FAZER $;
   define
      WHILE $VARIANDO
           { DESCENDO define >= |
                      define <= } $ATE DO
      BEGIN
          $FAZ;
          $VARIANDO := $VARIANDO
              { DESCENDO define - | define +}  1
      END;
endmacro
BEGIN
   INTEGER I,J;
   VARIANDO I ATE 10 FAZER J:=J+1;     Trecho-substituível
   VARIANDO I DESCENDO ATE 1 FAZER;    Trecho-substituível
END

   Teste:                  { DESCENDO define >= | define <= }
   Teste:                  { DESCENDO define -  | define +  }
   Trecho-substituível_1:   VARIANDO I ATE 10 FAZER J:=J+1;

   Parâmetro-efetivo-$VARIANDO:   I
   Parâmetro-efetivo-$ATE:        10
   Parâmetro-efetivo-$FAZER:      J:=J+1;

Trecho-substituível_1^substituído: WHILE I <= 10 DO
                                       BEGIN
                                         J:= J+1;
                                         I:= I+1;
                                       END
Trecho-substituível_2:   VARIANDO I DESCENDO ATE 1 FAZER;
Trecho-substituível_2^substituído: WHILE I >= 1 DO
                                       BEGIN
                                         I:=I-1;
                                       END;
```

No exemplo III.20, apareceu duas vezes um outro tipo de Teste, definindo pedaços alternativos; um fará parte da substituição, caso a cláusula de nome "DESCENDO" apareça no Trecho-substituível, já o outro pedaço alternativo fará parte apenas se essa cláusula não aparecer no Trecho-substituível.

Exemplo III.21: (LPSE)

```
macro PEG $ [FIQUE];
   define
       PILHA ($PEG);
      { FIQUE define | define $PEG := $PEG - 1  ; }
   endmacro
. . .
I := PEG J;
I := PEG J FIQUE;

Trecho-substituível:                PEG J;
Parâmetro-efetivo_$PEG:             J

Trecho-substituível_1^substituído: PILHA (J);
                                    J := J - 1;

Trecho-substituível_2:              PEG J FIQUE;

Parametro-efetivo_$PEG:             J

Trecho-substituível^substituído:    PILHA (J)
```

O exemplo III.21 parece com o III.20. Apenas um dos pedaços alternativos é vazio. O efeito é que só haverá substituição causada por esse teste quando não ocorrer a cláusula "FIQUE".

Exemplo III.22: (LPSE)

```
macro CASO $ [QUANDO $=>$C;]... FIMCASO; define... endmacro
. . .
CASO I                          ┐
  QUANDO 0   => X:=Y;           │
  QUANDO 3   => X:=Z;           ├ Trecho-substituível_1
  QUANDO N+7 => Z:=Y;           │
FIMCASO;                        ┘
. . .
  CASO A+B*Z FIMCASO;           ┐ Trecho-substituível_2
. . .
CASO I + J                      ┐
  QUANDO 1 => X:=X+1;           ├ Trecho-substituível_3
FIMCASO;                        ┘
. . .
```

Grupo-opcional-repetitivo: [QUANDO $ =>$C;]...

Enquanto um Grupo-opcional-não-repetitivo indica que um trecho opcional pode aparecer nenhuma ou uma vez num Trecho-substituível, um Grupo-opcional-repetitivo indica que o trecho pode aparecer nenhuma, uma ou mais vezes.

Grupo-opcional-repetitivo ::="[" Alternativas "]" "..."

No exemplo III.22 o trecho correspondente ao Grupo-opcional-repetitivo aparece 3 vezes no Trecho-substituível_1, nenhuma no Trecho-substituível_2 e uma vez no Trecho-substituível_3.

Exemplo III.23: (LPSE)

```
macro CASO $ [QUANDO $ => $C ; ]...FIMCASO;
    define
      { QUANDO define                                    } Teste
            IF $CASO = $QUANDO THEN $C ELSE} ;           }
    endmacro

Trecho-substituível_1^substituído: IF I = 0 THEN
                                      X:= Y ELSE
                                   IF I = 3 THEN
                                      X:= Z ELSE
                                   IF I = N+7 THEN
                                      Z:= Y ELSE;

Trecho-substituível_2^substituído:      ;


Trecho-substituível_3^substituído: IF I+J = 1 THEN
                                      X:=X+1  ELSE;
```

Nesse exemplo vemos que o Teste define um pedaço do Padrão-final que é repetido para cada ocorrência do trecho opcional no Trecho-substituível. Se o Grupo-opcional-repetitivo possui Indicadores-de-parâmetro, cada Referência-a-parâmetro que corresponda a algum deles, será substituída em cada repetição pelo respectivo parâmetro-efetivo da respectiva repetição do trecho opcional no Trecho-substituível.

Observe-se que essa definição do funcionamento de um Teste independe do fato dele se referir a um grupo repetitivo ou não, pois, para este, só haverá uma única ocorrência do pedaço correspondente ao Grupo. Portanto, a definição anterior de Teste para Grupos não-repetitivos, dada após o exemplo III.19, é um caos particular desta nova definição, que abrange Testes para Grupos repetitivos ou não-repetitivos.

Exemplo III.24: (LPSE)

```
macro SOME $ [EM $].../;
  define {EM define $EM:=$EM+$SOME;
          |  define $SOME:=$SOME+$SOME;}
  endmacro
. . .
SOME X EM Y EM Z EM W;  }Trecho-substituível_1
. . .
SOME R;                 }Trecho-substituível_2
. . .

Trecho-substituível_1^substituído: Y:=Y+X;
                                   Z:=Z+X;
                                   W:=W+X;

Trecho-substituível_2^substituído: R:=R+R;
```

Nesse exemplo, como nos outros, é dado no Teste um "default", pedaço que será colocado na substituição apenas se o Grupo não ocorrer no Trecho-substituível.

Exemplo III.25: (LPSE)

```
macro EM $ {SOME $}.../;
  define
      $EM:=$EM   {SOME define + $SOME};
  endmacro
```

A diferença entre um Grupo obrigatório e um opcional é que no obrigatório o trecho definido pelo grupo tem que aparecer pelo menos uma vez no Trecho-substituível. São usadas chaves "{}" no lugar de colchetes "[ ]".

Grupo-obrigatório-repetitivo ::= "{" Alternativas "}" "..."

Os testes correspondentes a Grupo-obrigatório nao devem ter "default" pois estes nunca serão escolhidos pelo processador da linguagem "E".

Exemplo III.26: (LPSE)

```
macro FOR $ {UPTO $ | DOWNTO $}  DO $;
  define
    . . .
  endmacro
 . . .
  FOR I UPTO 10 DO X(I):=0;
 . . .
  FOR J DOWNTO I DO Z(J):=X(J)-J;
 . . .
```

Como o nome diz, Alternativas fornecem sintaxes alternativas para os Trechos-substituíveis. Elas só podem aparecer dentro de um Grupo e podem ser composta de uma única alternativa (Conjunto-de-sintaxes) ou mais de uma separadas por "|".

Alternativas::=Conjunto-de-sintaxes("|" Conjunto-de-sintaxes)*

O exemplo III.26 possui um Grupo-obrigatório-não-repetitivo que indica que a sintaxe introduzida por este Grupo não pode ser repetida, nem omitida, no mesmo Trecho-substituível. Ou seja, um Grupo-obrigatório-não-repetitivo é apenas uma sintaxe encerrada entre parenteses. No Exemplo III.26 uma das duas Alternativas tem que aparecer no Trecho-substituível, e apenas uma vez.

Grupo-obrigatório-não-repetitivo ::= "{" Alternativas "}"

Exemplo III.27: (LPSE)

```
macro FOR $ {UPTO $ | DOWNTO $}  DO $
  define
     WHILE $FOR {UPTO define <=$UPTO
                 |DOWNTO define >= $DOWNTO}
     DO BEGIN
          $DO;
          $FOR := $FOR {DOWNTO define - |UPTO define +}  1
        END;
  endmacro
. . .
```

Neste exemplo III.27 notamos que um Teste pode ser composto por mais de um Teste-de-alternativa. A ordem desses Testes-de-alternativa não interessa. Entretanto, a mesma Alternativa não deve ser testada mais de uma vez no mesmo Teste, já que, uma vez feita a substituição correspondente a um Teste-de-alternativa, os testes que o seguem não são realizados.

Exemplo III.28: (LPSE)

```
macro INCR {UM | DOIS};
  define
     X:= X + 1 {DOIS define + 1};
  endmacro
. . .
```

Pelo exemplo III.28 vemos que nem todas Alternativas do Grupo precisam ser testadas.

Exemplo III.29: (LPSE)

```
macro PARA $ [INCR $ | DECR $] ATE $ FAZ $;
  define
     . . .
     $PARA := PARA {INCR define + $INCR
                   |DECR define - $DECR
                   |define + 1          }
     . . .
  endmacro
. . .
```

No exemplo III.29 temos um Teste com mais de um Teste-de-alternativa e com "default", pois refere-se a um Grupo-opcional. O "default" é usado na substituição caso nenhuma das Alternativas desse Grupo tenha ocorrido.

Exemplo III.30: (LPSE)

```
macro PARA $ [INCR $ | DECR $] ATE $ FAZ $;
  define
     . . .
     $PARA := $PARA {INCR define + $INCR
                    |DECR define - $DECR};
     . . .
  endmacro
. . .
```

Nesse exemplo III.30, nota-se que, mesmo referindo-se a um Grupo-opcional, o Teste não precisa ter "default", isto é, o "default" é uma substituição vazia.

Exemplo III.31: (LPSE)

```
macro ANDE  [0 | - 1] ;
  define
     X:=X  { - define - 1 | define + 1} ;
  endmacro
. . .
ANDE;       }Trecho-substituível_1
ANDE 0;     }Trecho-substituível_2
ANDE -1;    }Trecho-substituível_3
. . .

Trecho-substituível_1^substituído: X:=X+1;

Trecho-substituível_2^substituído: X:=X;

Trecho-substituível_3^substituído: X:=X-1;
```

O importante para ser notado no exemplo III.31 é a substituição do Trecho-substituível_2. Como vemos, o "default" não foi usado, pois uma das alternativas estava presente. Como não havia Teste-de-alternativa para a Alternativa 0, foi feita uma substituição vazia. Em resumo, o Teste

```
{ - define - 1 | define + 1}
```

é equivalente ao Teste

```
{ 0 define | - define - 1 | define + 1}
```

```
Teste ::= "{" Teste-de-alternativa ("|" Teste-de-alternativa)*
              Default? "}"

Default ::= "|" "DEFINE" Definição
```

Exemplo III.32: (LPSE)

```
macro MAIS [S | B = $ | C ]...;
  define
     Z:=Z + { B define $B | define 1} ;
  endmacro
. . .
MAIS A B=3 B=5 C A A C B=7;  }Trecho-substituível_1
. . .
MAIS ;                       }Trecho-substituível_2
. . .

Trecho-substituível_1^substituído: Z:=Z+3+5+7;

Trecho-substituível_2^substituído: Z:=Z+1;
```

O exemplo III.32 tenta mostrar que, mesmo para Grupos-opcionais-repetitivos, o "default" só é usado se o grupo não aparece no Trecho-substituível. Caso contrário, para cada aparição do Grupo, os Testes-de-alternativa são consultados.

III.6 Conjunto-de-sintaxes

Exemplo III.33: (LPSE)

```
macro COBRE  {CUSTOS = $ || LUCRO = $ || IMP = $ } ;
  define
     . . .
  endmacro
. . .
COBRE IMP = 10 CUSTO = 50000 LUCRO = 12;
COBRE LUCRO = 33 IMP = 5 CUSTO = 100000;
. . .
```

O Grupo-obrigatório-não-repetitivo do Exemplo III.33 é composto por uma única alternativa, que é o Conjunto-de-sintaxes

formado por 3 Sintaxes separadas por "||". Este símbolo indica que a ordem das sintaxes não importa no Trecho-substituível.

Exemplo III.34: (LPSE)

```
macro FAZ  [TUDO | ESQ || DIR] ;
  define
    . . .
  endmacro
. . .
  FAZ;
  FAZ TUDO;
  FAZ ESQ DIR;
  FAZ DIR ESQ;
. . .
```

O Exemplo III.34 mostra que um Conjunto-de-sintaxe faz parte de Alternativas.

Exemplo III.35: (LPSE)

```
macro FAZ [ TUDO | ESQ || DIR ]...;
  define ... endmacro
. . .
  FAZ TUDO ESQ DIR DIR ESQ DIR ESQ TUDO ESQ DIR;
  FAZ;
. . .
```

Conjunto-de-sintaxes ::= Sintaxe ("||" Sintaxe)*

III.7 Sintaxe

Exemplo III.36: (LPSE)

```
macro FAZ { [COMPRIME] || [UMA | DUPLA] FACE || DISCO = $};
  define... endmacro
. . .
  FAZ DISCO = 3 UMA FACE;
  FAZ DUPLA FACE COMPRIME DISCO = 5;
. . .
```

O exemplo mostra que uma Sintaxe é composta por Grupos e Cláusulas.

```
Sintaxe ::= (Cláusula | Grupo)+
Cláusula::=Cláusula-sem-parâmetro | Cláusula-com-parâmetro
Grupo ::= Grupo-obrigatório | Grupo-opcional
Grupo-obrigatório ::= Grupo-obrigatório-não-repetitivo
                    | Grupo-obrigatório-repetitivo
Grupo-opcional ::= Grupo-opcional-não-repetitivo
                 | Grupo-opcional-repetitivo
```

III.8 Padrão-inicial

Examinaremos o comando LPS

```
Comando-if ::= "IF" Expressão "THEN" Comando ("ELSE" Comando)?
```

Como o próprio Comando-if é um Comando, podemos ter

```
IF A = B THEN
     X := 0
ELSE
     IF ERRADO THEN
          GOTO FIM;
```

Notemos que ";" termina simultaneamente o Comando-goto e os dois Comandos-if, sem fazer parte da sintaxe de nenhum deles.

Como na linguagem "E" podemos ter um Trecho-substituível dentro de um parâmetro efetivo de outro Trecho-substituível, uma declaração com a mesma propriedade de auto-envolvimento dos comandos LPS ou ALGOL exige novas facilidades da linguagem "E". Isso é discutido nos próximos exemplos.

Exemplo III.37: (LPSE)

```
macro IF $ THEN $
  define ... endmacro
. . .
  IF A = B THEN X:=0;
  IF ERRADO THEN GOTO FIM;
. . .
```

Teríamos, no exemplo III.37, um Trecho-substituível começando com o primeiro "IF" e só terminando no fim do programa, tendo o segundo "IF" começado um segundo Trecho-substituível dentro do parâmetro-efetivo $THEN do primeiro Trecho-substituível. Não é esse o sentido usado nas linguagens tipo ALGOL.

Exemplo III.38: (LPSE)

```
macro IF $ THEN $ [ELSE $];
  define ... endmacro
. . .
  IF A = B THEN
      IF ERRADO THEN
         GOTO FIM;
  GOTO RPT;
. . .
```

No exemplo III.38, notamos a necessidade de dois ";" para terminar os dois Trechos-substituíveis IF. Senão as palavras "GOTO" e "RPT" também fariam parte do parâmetro-efetivo-$THEN do primeiro "IF".

Para solucionar esses problemas a linguagem "E" fornece dois conjuntos de Terminadores: um para comandos - STEND - e outro para expressões - OPEND. Esses Terminadores são usados no Padrão-inicial para indicar que a sintaxe do Trecho-substituível termina com qualquer palavra do conjunto indicado e essa palavra não faz parte desta sintaxe. Esses dois conjuntos dependem da linguagem hospedeira.

Exemplo III.39: (LPSE)

```
macro IF $ THEN $ [ ELSE $]  stend
  define ... endmacro
. . .
  IF A = B THEN
      IF ERRADO THEN
        GOTO FIM
      ELSE
        X := 0
  ELSE
      IF X = 0 THEN
        X := 1;
. . .
```

No exemplo III.39, os Trechos-substituíveis IF serão entendidos da mesma maneira que os Comandos-if da LPS, se "ELSE" e ";" pertencerem ao STEND. Efetivamente, na LPSE, essas palavras pertencem ao conjunto STEND de terminadores de comando.

Terminador ::= "STEND" | "OPEND"

O Padrão-inicial de uma Declaração-de-substituição é composto de uma Sintaxe seguida opcionalmente por um Terminador.

Padrão-inicial ::= Sintaxe Terminador?

Entretanto esta Sintaxe tem algumas restrições:

1) Deve iniciar com uma Cláusula, cujo nome será o nome da substituição declarada. Não se pode, portanto, iniciar o Padrão-inicial com um Grupo.

2) Se o Padrão-inicial não possui um Terminador, a sua Sintaxe deve terminar com uma Cláusula-sem-parâmetro, não podendo terminar nem com um Grupo, nem com uma Cláusula-com-parâmetro.

O motivo da existência dessas restrições é a diminuição que elas causam na complexidade das substituições, em comparação com a reduzida aplicabilidade, considerando-se a existência dos Terminadores "STEND" e "OPEND".

III.9 Teste-de-alternativa

Exemplo III.40: (LPSE)

```
macro RELATORIO  { NOME = $ | { LIN = $}...}  stend
    define
        . . .
        {{ LIN define $LIN}  | NOME define INTEGER $NOME}
        . . .
    endmacro
```

Como podemos ter Grupos dentro de Grupos, para podermos testar a ocorrência de uma Alternativa que é um Grupo, que por sua vez tem suas Alternativas, um Teste-de-Alternativas pode ser composto por um Teste.

```
Teste-de-alternativa ::= Indicador-de-cláusula
                         "DEFINE" Padrão-final
                       | Teste
```

III.10 Indicador-de-cláusula

Exemplo III.41: (LPSE)

```
macro SOME [EM $ ]  opend
     define
     ... { $EM define 1 }...
         { EM define 1 }
     endmacro
```

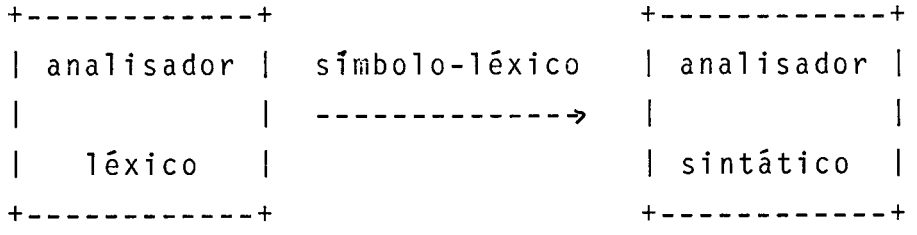
Um Indicador-de-cláusula pode ser o nome de uma Cláusula ou o nome-do-parâmetro dessa Cláusula. Os dois usos são idênticos. No exemplo III.41, os dois Testes mostrados têm o mesmo significado.

```
Indicador-de-cláusula ::= Nome-de-cláusula |
                          Referência-a-parâmetro
```

CAPÍTULO IV

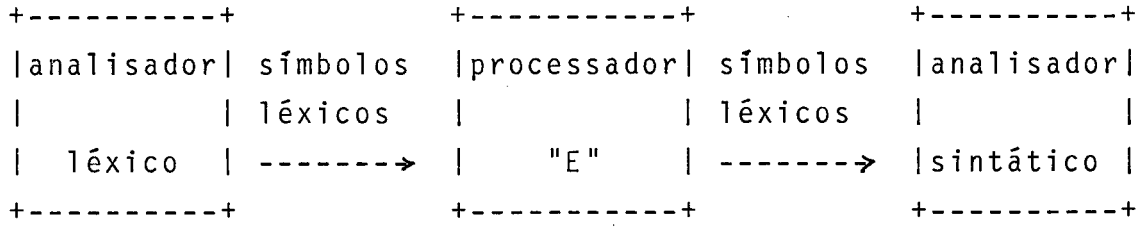
IMPLEMENTAÇÃO NO COBRA-530

Esse capítulo descreve o processador da linguagem extensora "E" e a implementação do compilador LPSE, produto da incorporação desse processador ao compilador LPS. A saída desse processador é uma sequência de símbolos-léxicos. Para tal a análise léxica também é papel desse processador. O analisador léxico do compilador LPS foi aproveitado para fazer essa análise. Isso foi simples proque o relacionamento entre os analisadores léxicos e sintáticos, no compilador LPS, é independente de realimentação, ou seja, as informações fluem em um único sentido, do léxico para o sintático, não precisando a análise léxica de nenhuma informação da análise sintática. O esquema abaixo representa esse relacionamento.



relacionamento independente de realimentação

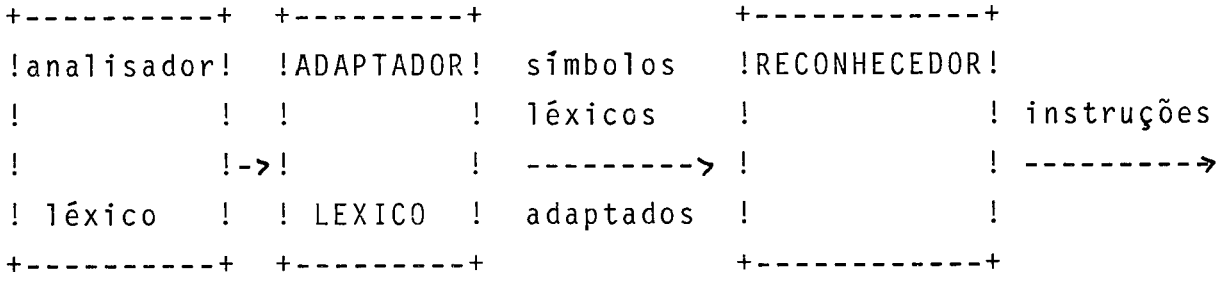
Esse mesmo tipo de relacionamento foi mantido entre o processador "E" e os dois analisadores o seguinte esquema:



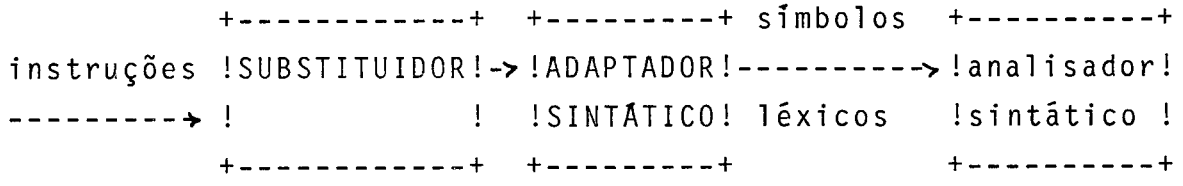
O processador "E" é composto de quatro módulos: ADAPTADOR-LÉXICO, RECONHECEDOR, SUBSTITUIDOR e ADAPTADOR-SINTÁTICO.

O ADAPTADOR-LÉXICO ajusta a saída do analisador léxico, para que o restante do processador "E" trabalhe com objetos apropriados (palavras, símbolos-reservados e identificadores-de-parâmetros). O ADAPTADOR-SINTÁTICO transforma de volta esses objetos, para serem consumidos pelo analisador sintático. O RECONHECEDOR é detalhado nos ítens IV.1 a IV.4 e o SUBSTITUIDOR no ítem IV.5.

Resumidamente, o RECONHECEDOR faz a análise de um Programa LPSE, decompondo-o em um conjunto de instruções que representam as substituições que devem ser realizadas. O SUBSTITUIDOR executa essas instruções, sintetizando, assim, o Programa substituído, que é fornecido como entrada para o analisador sintático. O relacionamento independente de realimentação é essencial nesse caso, pois, como o SUBSTITUIDOR é bem menor que um analisador léxico normal, um esquema multi-passo pode ser, e foi, empregado.



PASSO 1



PASSO 2

O SUBSTITUIDOR pode ser encarado como uma UCP executando as tais instruções. O RECONHECEDOR é bem mais complicado, pois tem que, entre outras coisas, analisar os Padrões declarados nas Declarações-de-substituições e reconhecer os Trechos-substituíveis

baseados nesses Padrões. Para tal, a análise de um Padrão-inicial gera uma árvore, que sintetiza a sintaxe descrita nesse Padrão-inicial e serve de guia para o reconhecimento de Trechos-substituíveis que sigam esse Padrão. Este é o assunto do ítem IV.4.

IV.1 Notações

Para descrever as diferentes transformações produzidas pelo processador "E", a notação para gramáticas de atributos, fornecida no ítem I.6, deve ser completada com as seguintes regras:

(j) símbolos de ação podem fazer parte das regras gramaticais e são denotados por números sublinhados, como em

2 Declaração-de-substituição* 3 Definição

(k) as ações correspondentes aos símbolos de ação são descritas após a regra gramatical, sendo precedidas pelo símbolo de ação e o separador "==:". Exemplo:

1 ==: Teste-de-alternativas^de-cláusula := falso;

(l) a ação composta pelo símbolo "iff" e por uma expressão lógica indica que um erro deve ser emitido caso a expressão seja avaliada como falsa. O símbolo "iff" pode ser lido como "se e somente se". Por exemplo:

2 ==: iff not Grupo↓obrigatório;

IV.2 Reconhecedores

A principal dificuldade no reconhecimento de um programa LPSE reside, precisamente, na investigação dos Trechos-substituíveis. Além de terem as suas sintaxes introduzidas por Declarações, como diversas entidades em possivelmente todas as linguagens de programação, essa sintaxes são tão complexas quanto queira o programador LPSE, ao contrário da maioria das linguagens de programação. Essa versatilidade é que a torna um verdadeiro mecanismo da extensão.

Com efeito, os reconhecedores de linguagens de programação não encaram as declarações de entidades como extensões à linguagem. Reconhecem as estruturas sintáticas padrões geradas pelas declarações, deixando o reconhecimento específico da extensão (conferência de identificadores, quantidade e qualidade dos parâmetros etc) para outro nível de compilação: o da semântica estática.

Desse modo, o compilador da linguagem de programação LPS implementado no COBRA-530 reconhece a produção

```
Chamada-a-rotina
      ::= identificador
       |  identificador "(" Parâmetro-efetivo
                          ("," Parâmetro-efetivo)* ")"
```

Como descrevendo totalmente o não-terminal Chamada-a-rotina, não sendo acrescentada novas alternativas a essa produção durante a compilação. A conferência do número de parâmetros não é feita através das rotinas de reconhecimento sintático desse compilador.

Portanto, o compilador da linguagem extensora LPSE teria duas opções a seguir:

a) utilizar as mesmas rotinas que reconheceriam a estrutura básica do LPSE para reconhecer os Trechos-substituíveis. Isso equivale a ir acrescentando novas produções à linguagem básica durante o próprio reconhecimento;

b) reconhecer os Trechos-substituíveis num nível após o reconhecimento sintático da linguagem básica, não usando as mesmas rotinas que este. Ou seja, a parte semântica ficaria encarregada do reconhecimento sintático das substituições a serem feitas.

Essa questão foi decidida observando-se que o reconhecimento das estruturas da linguagem básica implicava em ações estritamente particulares a cada uma dessa estruturas e o reconhecimento dos Trechos-substituíveis, pela maleabilidade de suas sintaxes, era muito mais complexo, porém regular.

A complexibilidade citada exigia um reconhecedor sintático apropriado para tratar os Trechos-substituíveis.

A adequação desse reconhecedor de estruturas sintáticas genéricas, baseadas em expressões regulares, às estruturas básicas do LPSE, cada qual sendo um caso particular, pareceria artificial e isso tiraria a nitidez ganha com a separação dos reconhecimentos.

Portanto a opção (b) foi adotada

IV.3 Estruturas básicas

Definimos como sendo as estruturas Básicas do LPSE a linguagem gerada pela gramática original após eliminação das produções nas quais o lado direito seria função do próprio processamento dos programas.

Os não-terminais com esse tipo de lado direito são: Trecho-substituível, Referência-a-parâmetro e Nome-de-cláusula. Esses não-terminais apareciam nas produções:

```
(1) Padrão-final ::= (Elemento-insubstituível |
                      Trecho-substituível |
                      Referência-a-parâmetro | Teste)*

(2) Indicador-de-cláusula ::= Nome-de-cláusula |
                              Referência-a-parâmetro

(3) Texto ::= (Elemento-insubstituível | Trecho-substituível)*
```

Como as produções acrescentáveis a gramática para Nome-de-cláusula ou Referência-a-parâmetro sempre teriam no seu lado direito, respectivamente, uma palavra ou um identificador-de-parâmetro, alteramos a produção (2) para:

```
(2) Indicador-de-cláusula ::= palavra |
                              identificador-de-parâmetro
```

Como, num Padrão-final, um Trecho-substituível sempre seria uma combinação de palavras, Referência-a-parâmetro e Testes, e como temos

```
(4) Elemento-insubstituível ::= palavra | "ORIGINAL" palavra
```

alteramos (1) para

(1) Padrão-final ::= ("ORIGINAL" palavra | palavra |
identificador-de-parâmetro | Teste)*

Como, num Texto, um Trecho-substituível sempre seria uma combinação de palavras, alteramos (3) para:

(3) Texto ::= ("ORIGINAL" palavra | palavra)*

Foi eliminada a produção (4) por que o não-terminal Elemento-insubstituível passou a ser inatingível a partir do não-terminal inicial (Programa).

IV.4 Reconhecimento das estruturas básicas do LPSE

A linguagem correspondente a essas Estruturas Básicas é pequena e de fácil reconhecimento por qualquer método. Mais complicadas são as ações "semânticas" a serem tomadas durante a sua compilação. Em virtude disso foi escolhido um método simples, que permite fácil inserção das rotinas semânticas e que alterações na linguagem possam ser feitas sem grande transtorno. A técnica de tradução descendente recursiva (Bauer et al. 74) preenche esses requisitos, exigindo, apenas, uma linguagem de programação recursiva. A linguagem de programação de sistemas da linha COBRA-530, LPS, em parte possui essa características. Nela foi programado o compilador LPSE.

Dizemos "em parte" pois, embora permita rotinas recursivas, sua alocação de dados na memória é estática. Isto importa principalmente às rotinas semânticas que precisam guardar os atributos dos símbolos não-terminais. Como cada não-terminal nesse método tem sua produção expressa numa rotina, esses atributos são variáveis internas a essa rotina. Ora, existem não-terminais recursivos, cuja análise pode implicar na análise recursiva dele mesmo. O ideal, nesse caso, seria uma alocação dinâmica de variáveis, de modo que uma chamada recursiva não destruísse os valores anteriores dos atributos, que devem ser preservados para após o retorno dessa chamada.

A solução encontrada foi simular a alocação dinâmica de variáveis nas rotinas de seguinte modo:

a) em cada rotina recursiva que necessite de variáveis é definido um vetor cuja primeira posição contém seu próprio tamanho. As variáveis necessárias são redefinições de campos desse vetor.

b) o primeiro comando da rotina recursiva deve ser uma chamada à rotina SALVA, com esse vetor como parâmetro efetivo. Essa rotina empilhará as variáveis contidas nesse vetor em uma área própria.

c) antes do retorno da rotina recursiva deve ser chamada a rotina RESTAURA com o mesmo vetor como parâmetro. Essa rotina desempilhará os antigos valores das variáveis internas.

Isso permite a programação de rotinas recursivas como se as variáveis fossem alocadas dinâmicamente, embora necessite de mais memória, pois variáveis são salvas antes de qualquer utilização.

Exemplo IV.1 (LPS):

```
procedure SINTAXE:
   begin
      byte(5) VARIAVEIS = (4,0,0,0,0);
        word ENDER.ARVORE.CLAUSULA.CORRENTE pos VARIAVEIS + 1;
        word ENDER.ARVORE.PRIMEIRA.CLAUSULA pos VARIAVEIS + 3;
      SALVA(VARIAVEIS);
      ...
      ...
      RESTAURA(VARIAVEIS);
   end SINTAXE;
```

No método descendente recursivo, a gramática da linguagem a ser compilada pode ser usada com a função de um fluxograma. As rotinas correspondem às produções, havendo um grupo de outras rotinas formando o esqueleto do compilador. São rotinas de "scanning" do texto fonte, manuseio de erros, etc.

A seguir estão descritas as variáveis usadas nesse método, algumas rotinas do esqueleto e, de uma maneira geral, as rotinas dos não-terminais.

IV.4.1 Objetos usados no reconhecimento

O analisador léxico ao ler um "token" fornece na variável TIPO.DO.TOKEN um tipo-de-token, isto é, um valor que informa se o "token" é "MACRO", "DEFINE", "ENDMACRO", "OPEND", "STEND", "ORIGINAL", "...", "{", "}", "|", "|", "|", "||", identificador-de-parâmetros, "$", palavra ou fim-de-arquivos. Caso o "token" seja um identificador-de-parâmetro ou uma palavra, o analisador léxico fornece na variável NUMERO.DO.TOKEN um atributo, isto é, um valor inteiro, que representa bi-univocamente essa palavra ou identificador-de-parâmetro.

Ou seja, os "tokens" da sequência "X:=X+Y" são reconhecidos pelo analisador-léxico como palavras. Entretanto, são distinguidos por inteiros distintos que representam univocamente cada palavra. Obviamente, os dois "X" recebem o mesmo atributo, pois não são palavras distintas.

Também existe a variável NUMERO.DO.TOKEN.ANTERIOR, que contem o atributo do "token" anterior ao corrente.

IV.4.2 Sub-rotina LE.TOKEN

Faz a analise léxica do texto fonte preenchendo, a cada vez que é chamada, as variáveis TIPO.DO.TOKEN, NUMERO.DO.TOKEN e NUMERO.DO.TOKEN.ANTERIOR.

IV.4.3 Função TOKEN

Recebe um tipo-de-token como parâmetro e informa se este é o TIPO.DO.TOKEN corrente. Se for, imediatamente lê o próximo "token" pela rotina LE.TOKEN, onde é preenchida a variável NUMERO.DO.TOKEN.ANTERIOR.

IV.4.4 Sub-rotina ERRO

Imprime uma mensagem de erro.

IV.4.5 Sub-rotina EXIJA

Recebe um tipo-de-token como parâmetro e acusa erro se não for esse o TIPO.DO.TOKEN corrente.

IV.4.6 Rotinas dos não-terminais

Baseando-se na gramática da linguagem LPSE poderiam ser escritas as rotinas dos não-terminais. Entretanto, essa gramática foi alterada para outra equivalente, de modo a reduzir o tamanho do compilador e facilitar a inserção das rotinas semânticas.

Exemplo IV.2:

- produção na gramática LPSE.
  Conjunto-de-sintaxes ::= Sintaxe ("||" Sintaxe)*

- produção na gramática utilizada:
  Conjunto-de-sintaxes ::= Sintaxe ("||" Conjunto-de-sintaxes)?

- rotina (LPS):

```
procedure CONJUNTO.DE.SINTAXES;
   begin
     ...
     SINTAXE;
     ...
     if TOKEN(BARRA.BARRA) then
        begin
          CONJUNTO.DE.SINTAXES;
          ...
        end
     ...
end CONJUNTO.DE.SINTAXES;
```

Essa alteração na gramática foi feita para melhor encaixe das rotinas semânticas correspondentes ao "||", como veremos adiante.

Um dos artifícios empregados foi reunir diversos não-terminais parecidos em um só, usando variáveis como atributos desse não-terminal resultante para fazer a distinção entre os não-terminais originais.

Exemplo IV.3:

- produções na gramática LPSE:

```
Programa ::= Declaração-de-substituição * Texto
Declaração-de-substituição
     ::="MACRO" Padrão-inicial "DEFINE" Padrão-final "ENDMACRO"

Padrão-final ::= ("ORIGINAL" palavra | palavra |
                  identificador-de-parâmetro | Teste)*

Texto ::= ("ORIGINAL" palavra | palavra )*
```

- produções transformadas:

```
Definição ::= ("ORIGINAL" palavra | palavra |
               1 (identificador-de-parâmetro | Teste ))*
        1 ==: iff not texto;

Programa ::= 2 Declaração-de-substituição* 3 Definição
       2 ==: texto := falso;
       3 ==: texto := vero;

Declaração-de-substituição
     ::= "MACRO" Padrão-inicial "DEFINE" Definição "ENDMACRO"
```

Um Padrão-final é uma sequência de elementos de um conjunto com 4 componentes. Texto era uma sequência de elementos de um subconjunto daquele conjunto.

Foi criado o não-terminal Definição que substitui os dois outros. Duas das alternativas desse não-terminal só são permitidas caso o atributo texto não esteja ativo (vero). Ou seja, o não-terminal Definição com o atributo texto desligado corresponde ao Padrão-final. Com ele ligado, corresponde ao Texto.

O atributo texto é vinculado ao não-terminal Programa. Inicialmente é desligado. Só é ligado ao terminarem as Declarações-de-substituição. Com efeito, o não-terminal Texto só existia na gramática na produção do não-terminal Programa, seguindo as Declarações-de-substituição.

Outro artifício empregado foi o de separar os terminais iniciadores de não-terminais, dos próprios. Desse modo as rotinas dos não-terminais não precisam ser funções recursivas, difíceis de implementar em LPS.

Exemplo IV.4:

- Produções na gramática LPSE:

```
Programa ::= Declaração-de-substituição * Texto
Declaração-de-substituição
     ::= "MACRO" Padrão-inicial "DEFINE" Padrão-final "ENDMACRO"
```

- Produções Alteradas

```
Programa
     ::= 2 ("MACRO" Declaração)* 3 Definição
  2 ==: texto := falso;
  3 ==: texto := vero;

Declaração ::= Padrão-inicial "DEFINE" Definição "ENDMACRO"
```

-Como se vê, a rotina do não-terminal Declaração só precisará ser chamada quando o "token" "MACRO" aparecer. Desse modo a rotina acusará erro se a sintaxe de Declaração não for reconhecida. Do outro modo a rotina seria uma função que informaria se a Declaração-de-substituição foi reconhecida ou não.

- rotina para o não-terminal Programa:

```
procedure PROGRAMA
   begin
     byte TEXTO;
     ...
     TEXTO := FALSO;
     while TOKEN(TMACRO) do
        DECLARACAO;
     ...
     TEXTO := VERO;
     DEFINICAO;
   end PROGRAMA;
```

IV.4.7 Atributos

Como vimos, a gramática efetivamente empregada no reconhecedor LPSE foi uma gramática alterada, onde não-terminais foram agrupadas em um, sendo diferenciados por atributos desse não-terminal. Assim, quando a rotina do novo não-terminal Definição herdasse o atributo TEXTO ligado, ela reconheceria um Texto e não um Padrão-final, que seria o caso quando o atributo TEXTO viesse desligado.

Se olharmos as rotinas sintáticas ignorando a manipulação dos atributos, veremos que o reconhecedor reconhece uma outra linguagem diferente da LPSE. Os atributos e as restrições que eles impõem estariam no âmbito da semântica estática, que nada mais seria que regras sintáticas disfarçadas.

Esse é um expediente sempre utilizado na confecção de compiladores. A semântica estática sempre contém regras sintáticas que, se expressas através de uma gramática sem atributos, seriam mais complicadas do que descritas por uma gramática de atributos. O nosso reconhecedor, confeccionado manualmente, utiliza razoavelmente esse expediente com esse fim.

A questão é: como implementar um reconhecedor descendente recursivo baseado em uma gramática de atributos? Neste reconhecedor um não-terminal corresponde a uma rotina sintática; um atributo que esse não-terminal herde poderia corresponder a um parâmetro de entrada dessa rotina; um outro que fosse sintetizado seria um parâmetro de saída.

Essa implementação significa uma custosa troca de parâmetros entre rotinas. Muitas vezes os atributos recebidos por um não-terminal é passado adiante para outros não-terminais sem qualquer transformação. Na implementação sugerida acima o parâmetro-atributo deveria ser recebido por uma rotina e passado para outras. Isto seria desnecessário se o atributo fosse uma variável global a essas rotinas. Na rotina onde esse atributo não fosse transformado, nem consultado, não haveria qualquer computação com essa variável.

A dificuldade agora está na recursividade das rotinas não-terminais. Ela exige que os atributos de um não-terminal sejam salvos e restaurados conforme o esquema descrito no ítem

IV.4. Ou seja, os atributos devem ser variáveis locais às rotinas, o que implica que as rotinas dos não-terminais que recebam ou enviem esses atributos sejam declaradas no interior da rotina onde esses atributo está declarado.

Exemplo IV.5:

(1) Programa ::= 1 ("MACRO" Declaração)* 2 Definição
1 ==: texto := falso;
2 ==: texto := vero;

(2) Declaração ::= Padrão-inicial "DEFINE" Definição "ENDMACRO"

- Programa é o não-terminal inicial. Pela produção (1), definimos que as rotinas DECLARACAO e DEFINICAO serão declaradas no interior da rotina PROGRAMA, juntamente com o atributo TEXTO. Pela produção (2) concluímos que na rotina DECLARACAO há uma chamada da rotina DEFINICAO. Isto nos fixa na forma:

```
procedure PROGRAMA;
   begin
     byte TEXTO;
     procedure DEFINICAO;
        begin
          ...
        end;
     procedure DECLARACAO;
        begin
          ...
        end;
     ...
   end;
```

- esta abordagem do arranjo das declarações das rotinas dos não-terminais é suficiente para resolver o problema dos atributos na reconhecedor LPSE.

Algumas regras sintáticas da LPSE não foram colocadas na gramática que definiu essa linguagem. Isso também foi resolvido com a gramática de atributos empregada.

Exemplo IV.6:

Padrão-inicial ::= Sintaxe Terminador?

A Sintaxe de um Padrão-inicial deve iniciar com uma Cláusula, cujo nome será o nome da substituição declarada, nunca com um Grupo. Se o Padrão-inicial não possui um Terminador, a sua Sintaxe deve terminar com uma Cláusula-sem-parâmetro, não podendo terminar nem com um Grupo, nem com uma Cláusula-com-parâmetro.

Estas duas regras implicam na criação de dois atributos sintetizados pelo não-terminal Sintaxe: um que informa que a Sintaxe inicia com Cláusula e outro que indica que ela termina com uma Cláusula-com-parâmetro.

```
Padrão-inicial ::= Sintaxe 1 (Terminador | 2)
                   1 ==: iff Sintaxe^tem-nome
                   2 ==: iff Sintaxe^termina-com-palavra
```

Dizer que uma Sintaxe começa com uma Cláusula e não com um Grupo é dizer que ela inicia com uma palavra, que será o nome da Sintaxe e não com " " ou " ". Dizer que ela termina apenas com uma Cláusula-com-parâmetro é dizer que ela termina com uma palavra e não com um Indicador-de-parâmetro, " ", " " ou "...".

Na implementação que descrevemos, foi desnecessário o atributo tem-nome, já que o teste de que a Sintaxe começa com uma palavra pode ser feito apenas perguntando se o TOKEN era palavra antes de chamar-se a rotina SINTAXE.

- Produção empregada:

```
Declaração ::= Sintaxe 1 ("STEND"|"OPEN"| 2) "DEFINE"
                   Definição "ENDMACRO"
          1 ==: iff Sintaxe^tem-nome;
          2 ==: iff Sintaxe^termima-com-palavra;
```

- Rotina DECLARACAO:

```
procedure DECLARACAO;
    begin
      byte TERMINA.COM.PALAVRA;
      procedure SINTAXE;
         begin
           ...
         end SINTAXE;
      if TIPO.DO.TOKEN = TPALAVRA then
         begin
           ...
           SINTAXE;
           if TOKEN (TSTEND) then
                ...
           else if TOKEN (TOPEND) then
                ...
           else
             if not TERMINA.COM.PALAVRA then
                begin
                  ERRO (FALTSIMB)
                  ...
                end;
           EXIJA (TDEFINE);
           ...
           DEFINICAO;
           EXIJA (TENDMACRO);
           ...
      end
      else
         begin
           ERRO (FALTNOMMCR);
           ...
         end;
    end DECLARACAO;
```

Exemplo IV.7:

- produções originais:

```
Sintaxe ::= (Cláusula | Grupo)+

Cláusula ::= Cláusula-sem-parâmetro | Cláusula-com-parâmetro

Cláusula-sem-parâmetro ::= palavra+

Cláusula-com-parâmetro ::= palavra+ Indicador-de-parâmetro

Indicador-de-parâmetro ::= "$" | identificador-de-parâmetro

Grupo ::= Grupo-obrigatório | Grupo-opcional

Grupo-obrigatório ::= Grupo-obrigatório-não-repetitivo
                    |   Grupo-obrigatório-repetitivo

Grupo-obrigatório-não-repetitivo ::= "{" Alternativas "}"

Grupo-obrigatório-repetitivo ::= "{" Alternativas "}" "..."

Grupo-opcional ::= Grupo-opcional-não-repetitivo
                 |   Grupo-opcional-repetitivo

Grupo-opcional-não-repetitivo ::= "[" Alternativas "]"

Grupo-opcional-repetitivo ::= "[" Alternativas "]" "..."

Alternativas ::=Conjunto-de-sintaxes ("|" Conjunto-de-sintaxes)*
```

- produções empregadas:

```
Sintaxe ::= Ítem 1 (Ítem 1) *
      1 ==: Sintaxe^termina-com-palavra :=
                    Ítem^termina-com-palavra

Ítem ::= "{" 1 Grupo | "[" 2 Grupo | palavra Cláusula 3
   1 ==: Ítem^termina-com-palavra := falso;
         Grupo↓obrigatório := vero;
   2 ==: Ítem^termina-com-palavra := falso;
         Grupo↓obrigatório := falso;
   3 ==: Ítem^termina-com-palavra :=
               Cláusula^termina-com-palavra

Cláusula ::= palavra* ("$" 1 |identificador-de-parâmetro 1 | 2)
       1 ==: Cláusula^termina-com-palavra ::= falso
       2 ==: Cláusula^termina-com-palavra ::= vero

Grupo ::= Conjunto-de-sintaxes ("|" Conjunto-de-sintaxes)*
                              (1 "}" | 2 "]") ("..."|)
    1 ==: iff Grupo↓obrigatório;
    2 ==: iff not Grupo↓obrigatório;
```

- Rotina Sintaxe:

```
byte TERMINA.COM.PALAVRA;
procedure SINTAXE;
   begin
     byte procedure ITEM;
       begin
         ...
       end ITEM;
     ...
     if not ITEM then
       EXIJA (PALAVRA); & PROVOCA ERRO
     ...
     while ITEM do
       begin
         ...
       end;
     ...
end SINTAXE;
```

- Observação: inicialmente a função ITEM é chamada. Se retornar FALSO é porque o TOKEN corrente não é válido para iniciar um Ítem. Portanto, EXIJA (PALAVRA) provocará erro, já que PALAVRA é um início válido de Ítem.

- rotina Item

```
byte procedure ITEM;
   begin
     byte OBRIGATORIO;
     procedure GRUPO;
        begin
          ...
        end GRUPO;
     procedure CLAUSULA;
        begin
          ...
        end CLAUSULA;
     ...
     if TOKEN (ABRE.CHAVES) then
        begin
          OBRIGATORIO := VERO;
          GRUPO;
          TERMINA.COM.PALAVRA := FALSO;
          ITEM := VERO;
        end
     else if TOKEN (ABRE.COLCHETES) then
        begin
          OBRIGATORIO := FALSO;
          GRUPO;
          TERMINA.COM.PALAVRA := FALSO;
          ITEM := VERO;
        end
     else if TOKEN (PALAVRA) then
        begin
          CLAUSULA;
          ITEM:= VERO;
        end
     else
          ITEM := FALSO;
     ...
end ITEM;
```

- rotina Cláusula:

```
procedure CLAUSULA;
    begin
      ...
      while TOKEN (PALAVRA) do
          ...;
      if TOKEN (DOLAR) then
          begin
            TERMINA.COM.PALAVRA := FALSO;
            ...
          end
      else if TOKEN (IDENTIFICADOR.DE.PARAMETRO) then
          begin
            TERMINA.COM.PALAVRA := FALSO;
            ...
          end
      else
          begin
            TERMINA.COM.PALAVRA := VERO;
            ...
          end;
      ...
end CLAUSULA;
```

- rotina Grupo:

```
byte OBRIGATORIO;
procedure GRUPO;
   begin
     byte GRUPO.OBRIGATORIO; & ATRIBUTO HERDADO
     ...
     procedure CONJUNTO.DE.SINTAXES;
        begin
          ...
        end CONJUNTO.DE.SINTAXES;
     ...
     GRUPO.OBRIGATORIO := OBRIGATORIO;
     ...
     CONJUNTO.DE.SINTAXES;
     ...
     while TOKEN (BARRA) do
        begin
          CONJUNTO.DE.SINTAXES;
          ...
        end;
     EXIJA (if GRUPO.OBRIGATORIO then FECHA.CHAVES else
             FECHA.COLCHETES);
     if TOKEN (T...) then
        ...
     else
        begin
          ...
        end;
        ...
end GRUPO;
```

Exemplo IV.8:

- produções originais :

```
Teste-de-alternativas ::= Indicador-de-cláusula "DEFINE"
                          Padrão-final
                        | Teste

Indicador-de-cláusula ::= Nome-de-cláusula
                        | Referência-a-parâmetro

Teste ::= "{" Teste-de-alternativas
          ("|" Teste-de-alternativa)* Default? "}"
```

- produções empregadas:

```
Teste-de-alternativa
  ::= ("{" Teste 1 |(identificador-de-parâmetro|palavra) 2)
          ( 3 "DEFINE" Definição | )
1 ==: Teste-de-alternativa^de-cláusula := falso;
2 ==: Teste-de-alternativa^de-cláusula := vero;
3 ==: iff Teste-de-alternativa^de-cláusula;

Teste ::= Teste-de-alternativas ("|" Teste-de-alternativa)*
          Default? "}"
```

- rotina Teste-de-alternativa

```
procedure TESTE.DE.ALTERNATIVA;
    begin
      byte DE.CLAUSULA;
      ...
      if TOKEN (ABRE.CHAVES) then
         begin
           TESTE;
           ...
           DE.CLAUSULA := FALSO;
         end
      else
         begin
           if TOKEN (IDENTIFICADOR.DE.PARAMETRO) then
              ...
           else
              begin
                EXIJA (PALAVRA);
                ...
              end;
           DE.CLAUSULA := VERO;
         end;
      ...
      if DE.CLAUSULA then
         begin
           EXIJA (TDEFINE);
           DEFINICAO;
         end;
      ...
end TESTE.DE.ALTERNATIVA;
```

IV.5 Reconhecimento de Trechos-substituíveis

Pronto o reconhecedor das Estruturas Básicas do LPSE, fica faltando construir um reconhecedor do restante da linguagem: Trechos-substituídos, Referência-a-parâmetro e Nomes-de-cláusula. A sintaxe desses não-terminais deve ser o produto do reconhecimento dos Padrões-iniciais, pois são especificadas através deles. O mais complexo é o reconhecimento dos Trechos-substituíveis, que têm possibilidades sintáticas bem amplas, pois essa é a principal característica da LPSE.

O não-terminal Trecho-substituível aparece na gramática LPSE nas seguintes produções:

```
Padrão-final ::=(Elemento-insubstituível | Trecho-susbtituível|
                    Referência-a-parâmetro | Teste)*
Texto ::= (Elemento-insubstituível | Trecho-substituível)*
```

Como vimos na (Exemplo 4.3) essa duas produções foram sintetizadas em um única:

```
Definição ::= ("ORIGINAL" palavra | palavra |
              1 (identificador-de-parâmetro | Teste ))*
        1 ==: iff not texto;
```

Nessa produção não mais surge o não-terminal Trecho-substituível, embora seja sabido que eles são formados dentro das Definições.

Portanto, dentro do reconhecedor de Definição é que devem ser reconhecidos os Trechos-substituíveis.

O Padrão-inicial é uma meta-linguagem com a qual pode-se especificar Sintaxe de Trechos-substituível. Porém, é uma meta-linguagem voltada para o ser humano que a programará.

Não é apropriada para o processamento automático do reconhecimento de Trechos-substituíveis. Devido a isso, definiremos uma outra meta-linguagem, mais sintética, para ser a base desse reconhecedor. As produções nessa meta-linguagem serão geradas durante o reconhecimento dos Padrões-iniciais, dos quais elas serão uma representação.

IV.5.1 Geração da meta-linguagem intermediária

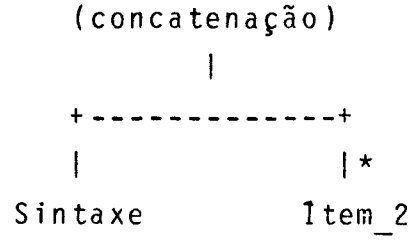
Neste ítem, os símbolos de ação podem representar a entrada de uma produção na gramática LPSE. Estas produções são denotadas como convencionado no ítem I.6. Porêm, o lado direito de uma produção é representado por uma árvore, cujo significado é pormenorizado no ítem IV.5.5. Os nós dessas árvores são facilmente visualisados por serem símbolos entre parênteses. Por exemplo:

```
(STEND)
(concatenação)
(||)
```

As folhas das árvores podem ser não-terminais, que representam as árvores geradas durante o reconhecimento desses não-terminais.

Essas árvores são n-árias, isto é, cada nó pode ter um número qualquer de filhos. Quando uma folha contiver o sinal "*" no seu galho, isto significará que ali podem existir qualquer número de folhas daquele tipo.

Exemplo:



IV.5.1.1 Trechos-substituíveis

Na gramática empregada no reconhecimento das Estruturas Básicas, o não-terminal Padrão-inicial foi diluído na especificação do não-terminal Declaração:

```
Declaração ::= Sintaxe ("STEND" | "OPEND" ) "DEFINE" Definição
                                                "ENDMACRO"
```

O não-terminal Sintaxe especifica sintaxe de Trechos-substituíveis. Os terminais "STEND" e "OPEND" acrescentam regras de finalização a essa sintaxes. O não terminal Definição em nada influi na especificação da Sintaxe dos Trechos-substituíveis. Abaixo reescrevemos a produção com a transformação para a outra meta-linguagem

```
Declaração ::= Sintaxe ("STEND" 1 | "OPEND" 2 | 3)
               "DEFINE" Definição "ENDMACRO"

        1 ==: Trecho-substituível ::= (STEND) ;
                                        |
                                      Sintaxe

        2 ==: Trecho-substituível ::= (OPEND) ;
                                        |
                                      Sintaxe

        3 ==: Trecho-substituível ::= Sintaxe;
```

A meta-linguagem proposta é uma gramática de produções com árvores no lado-direito. Elas simplesmente sintetizam a sentença da meta-linguagem original.

IV.5.1.2 Sintaxe

```
Sintaxe ::= Ítem_1 1 (Ítem_2 2 )*

      1 ==: Sintaxe ::= Ítem_1 ;

      2 ==: Sintaxe ::= (concatenação)
                              |
                        +-----------+
                        |           |*
                    Sintaxe     Ítem_2
```

O não-terminal Sintaxe especifica um Ítem de sintaxe, ou a concatenação de mais de um Ítem.

IV.5.1.3 Ítem

```
Ítem ::= "{" Grupo_1 1 | "|" Grupo_2 2 | palavra Cláusula 3

   1 ==: Ítem ::= Grupo_1;

   2 ==: Ítem ::= Grupo_2;

   3 ==: Ítem ::= Cláusula;
         Cláusula↓nome := (palavra);
```

Um Ítem pode ser um Grupo ou uma Cláusula.

IV.5.1.4 Cláusula

```
Cláusula (↓nome) ::= palavra*
                     ("$" 1 | identificador-de-parâmetro 1 | 2)

              1 ==: Cláusula ::= (cláusula-com-parâmetro) ;
                                          |
                                    +-----------+
                                    |           |*
                              Cláusula↓nome  (palavra)

              2 ==: Cláusula ::= (cláusula-sem-parâmetro) ;
                                          |
                                    +-----------+
                                    |           |*
                              Cláusula↓nome  (palavra)
```

Uma Cláusula é composta do seu nome, concatenado, ou não, com outras palavras, podendo ou não ter parâmetro.

IV.5.1.5 Grupo

```
Grupo ::= Conjunto-de-sintaxes 1
          ("|" Conjunto-de-sintaxes 2) ("}"|"]")
          ("..." 1 | 2 )

1 ==: if Grupo obrigatório then

            Grupo ::=  (obrigatório-repetitivo)
                                |
                       +------------------------+
                       |                        |*
            Conjunto-de-sintaxes_1 Conjunto-de-sintaxes_2
      else

            Grupo ::= (opcional-repetitivo)                ;
                                |
                       +--------------------+
                       |                    |*
            Conjunto-de-sintaxes_1 Conjunto-de-sintaxes_2

2 ==: if Grupo obrigatório then

            Grupo ::=     (obrigatório)
                                |
                       +--------------------+
                       |                    |*
            Conjunto-de-sintaxes_1 Conjunto-de-sintaxes_2
      else
            Grupo ::=      (opcional)                      ;
                                |
                       +--------------------+
                       |                    |*
            Conjunto-de-sintaxes_1 Conjunto-de-sintaxes_2
```

Um Grupo pode ser (obrigatório), (opcional), (obrigatório-repetitivo) ou (opcional-repetitivo), e representa alternativas de Conjuntos-de-sintaxes.

IV.5.1.6 Conjunto-de-sintaxes

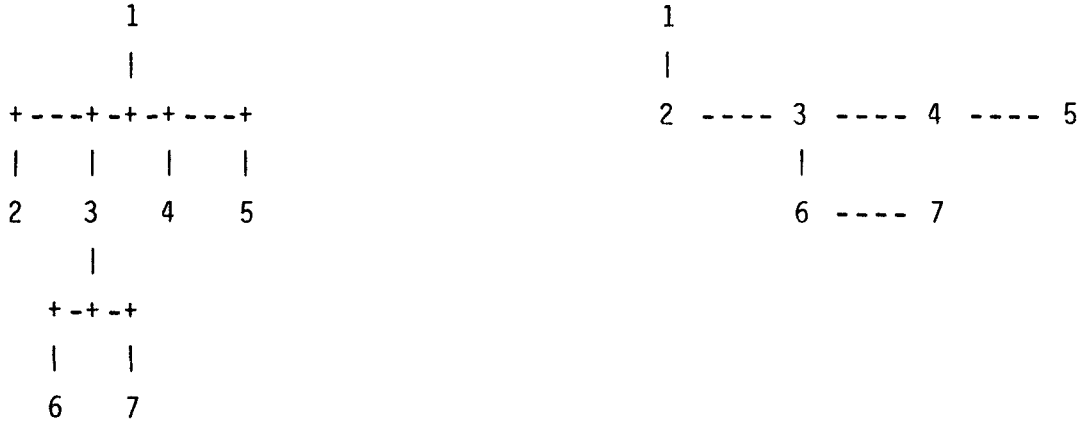
```
Conjunto-de-sintaxes ::= Sintaxe
                         ("||" Conjunto-de-sintaxes 1 | 2)
```

```
1 ==: Conjunto-de-sintaxes ::=    (||)  ;
                                    |
                                +---------+
                                |         |
                             Sintaxe  Conjunto-de-sintaxes
```

```
2 ==: Conjunto-de-sintaxes ::= Sintaxe;
```

IV.5.2 Implementação das árvores sintáticas

As árvores sintáticas definidas anteriormente são n-árias, isto é, seus nós podem ter qualquer número de filhos. Por isso foram implementadas por uma estrutura de dados "linkada" onde cada nó da árvore aponta para seu filho mais velho (mais à esquerda) e para seu irmão adjacente mais próximo.
A figura representa essa estrutura.



árvore n-ária — a mesma árvore

Na implementação descrita, um nó é uma estrutura que, além dos ponteiros para o Filho e Irmão acima descritos, possue um

campo para conter o Tipo-de-nó, um outro para conter um Atributo daquele nó, conforme será descrito mais adiante, e outro para um ponteiro auxiliar denominado Próximo.

```
Árvore ::= Nó+
Nó ::= Tipo-do-nó Filho Irmão Atributo Próximo
Tipo-do-nó ::="STEND"|"OPEND"|"concatenação"|"palavra"|
              "Cláusula-com-parâmetro"|"cláusula-sem-parâmetro"
              |"obrigatório-repetitivo"|"opcional-repetitivo"
              |"obrigatório"|"opcional"|"||"
Filho ::= Ponteiro-para-árvore | Nulo
Irmão ::= Ponteiro-para-árvore | Nulo
Próximo ::= Ponteiro-para-árvore | Nulo
Ponteiro-para-árvore ::= Word
Atributo ::= Word
Word ::= Bit 16
Bit ::= "0" | "1"
Nulo ::= "0000_0000_0000_0000"
```

As árvores são criadas durante o reconhecimento das Declarações e não são destruídas, nem reduzidas, enquanto durar o reconhecimento do Programa. Para criação dessas árvores foi então definida a função CRIA, que cria um nó do Tipo-de-nó fornecido como parâmetro e informa o endereço desse nó. Essa rotina resume-se a escolher consecutivamente um nó num "tabuleiro-de-nós" (pool).

Os campos de um nó são representados por variáveis referenciáveis ("based"). Durante o reconhecimento de um Padrão-inicial, essa rotina é chamada e essas variáveis são preenchidas para formar a árvore sintática que servirá de base para o reconhecimento dos Trechos-Substituíveis correspondentes a essa Declaração. São também necessários atributos para os não-terminais se comunicarem, informando onde estão as sub-árvores que eles formaram. A variável ENDER.ARVORE, ao término de qualquer rotina de não-terminal, aponta para a raiz da árvore que essa rotina formou. A variável ENDER.ARVORE.DECLARACAO contém o endereço da árvore gerada por essa Declaração.

Exemplo IV.9:

- produção empregada

```
Declaração ::= Sintaxe ("STEND" 1 | "OPEND" 2 |3) "DEFINE"
               Definição "ENDMACRO"

        1 ==: Trecho-substituível ::= (STEND)  ;
                                        |
                                      Sintaxe

        2 ==: Trecho-substituível ::= (OPEND)  ;
                                        |
                                      Sintaxe

        3 ==: Trecho-substituível ::= Sintaxe ;
```

- rotina Declaração

```
procedure DECLARACAO;
   begin
     ...
     word ENDER.ARVORE.DECLARACAO;
     word ENDER.ARVORE;
     ...
     procedure SINTAXE;
        begin
          ...
        end SINTAXE;
     ...
     if TIPO.DO.TOKEN = PALAVRA then
        begin
          ...
          SINTAXE;
          ENDER.ARVORE.DECLARACAO := ENDER.ARVORE;
          if TOKEN (TSTEND) then
(1)          (ENDER.ARVORE.DECLARACAO:=CRIA(NO.STEND)) ^
(1)                        FILHO := ENDER.ARVORE
          else if TOKEN (TOPEND) then
             (ENDER.ARVORE.DECLARACAO:=CRIA(NO.OPEND)) ^
                        FILHO := ENDER.ARVORE
          else ...;
          ...
        end
     else ...;
   ...
end DECLARACAO;
```

- observação: destaque para a linha (1)

CRIA(NO.STEND) aloca um nó de tipo "STEND"
ENDER.ARVORE.DECLARACAO := endereço desse nó
(...) ^ FILHO := ENDER.ARVORE; significa que o campo Filho desse nó recebe o endereço da árvore gerada pelo não-terminal Sintaxe

Exemplo IV.10:

- produção empregada

```
Sinxate ::= Ítem_1 1 (Ítem_2 2 )*
      1 ==: Sintaxe ::= Ítem_1;
      2 ==: Sintaxe ::= (concatenação) ;
                                  |
                          +-----------+
                          |           |*
                      Sintaxe      Ítem_2
```

- rotina Sintaxe;

```
   Word ENDER.ARVORE;
   procedure SINTAXE;
      begin
        word ENDER.ARVORE.ITEM.CORRENTE;
        word ENDER.ARVORE.PRIMEIRO.ITEM;
        byte TEM.CONCATENACAO;
        ...
        if not ITEM then
           EXIJA (PALAVRA);
(1)  ENDER.ARVORE.ITEM.CORRENTE :=
         ENDER.ARVORE.PRIMEIRO.ITEM := ENDER.ARVORE;
(2)  TEM.CONCATENACAO := FALSO;
        while ITEM do
           begin
(3)          ENDER.ARVORE.ITEM.CORRENTE :=
             ENDER.ARVORE.ITEM.CORRENTE^IRMAO := ENDER.ARVORE;
(4)          TEM.CONCATENACAO := VERO;
           end;
(5)  if TEM.CONCATENACAO then
           (ENDER.ARVORE := CRIA(NO.CONCATENACAO))^FILHO :=
            ENDER.ARVORE.PRIMEIRO.ITEM;
   end SINTAXE;
```

- observações:

linha (1) ENDER.ARVORE contém o endereço da árvore gerada pela rotina do não-terminal ITEM; esse valor será o ENDER.ARVORE.PRIMEIRO.ITEM e o ENDER.ARVORE.ITEM.CORRENTE.

linha (2) indica que até agora a Sintaxe só tem um Ítem até agora, não havendo concatenação de Ítens.

linha (3) após o reconhecimento de outro ítem, eles são encadeados. O ponteiro Irmão do Item-corrente aponta para a árvore recém-formada pelo não-terminal Ítem. E a variável ENDER.ARVORE.ITEM.CORRENTE é atualizada apontando, também, para essa árvore.

linha (4) é indicado que já houve concatenação.

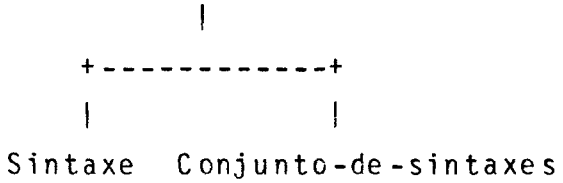
linha (5) se houve concatenação, todos os ítens estão ligados pelo ponteiro Irmão e essa cadeia está apontada por ENDER.ARVORE.PRIMEIRO.ITEM. É criado um nó (concatenação) cujo filho aponta para a cadeia de Irmãos citada.

Exemplo IV.11:

- produção empregada

Conjunto-de-Sintaxes ::= Sintaxe ("||"Conjunto-de-sintaxes 1|2)

1 ==: Conjunto-de-sintaxes ::= (||) ;



2 ==: Conjunto-de-sintaxes ::= Sintaxe ;

- rotina Conjunto-de-sintaxes:

```
procedure CONJUNTO.DE.SINTAXES;
   begin
     word FILHO.ESQUERDO;
     ...
     SINTAXE;
(1)  FILHO.ESQUERDO := ENDER.ARVORE;
     if TOKEN (BARRA.BARRA) then
     begin
       CONJUNTO.DE.SINTAXES;
(2)    FILHO.ESQUERDO^IRMAO := ENDER.ARVORE;
(3)    (ENDER.ARVORE := CRIA(NO.BARRA.BARRA))^FILHO.ESQUERDO;
     end;
     ...
end CONJUNTO.DE.SINTAXES;
```

- observações:

linha (1) a variável FILHO.ESQUERDO aponta para a árvore gerada pelo não-terminal Sintaxe.

linha (2) ao ser gerada uma nova árvore pelo não-terminal Conjunto-de-Sintaxes ela é apontada pelo campo Irmão da árvore anterior (FILHO.ESQUERDO).

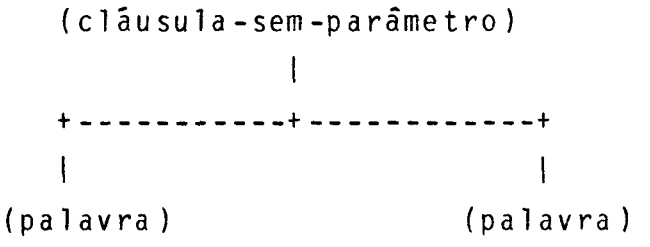
linha (3) é gerado um nó (||) com Filho apontando para o FILHO.ESQUERDO. Para sair da rotina atualizada, a variável ENDER.ARVORE tem que apontar para esse nó.

IV.5.3 Atributos sintáticos dos nós

Considerando a seguinte declaração:

```
macro TROQUE TUDO define
   ...
endmacro
```

observa-se que a árvore gerada conterá apenas uma Cláusula-sem-parâmetro e será da forma:



o que indica que a Sintaxe dos Trechos-substituíveis por essa declaração é composta da concatenação de duas palavras. Não quaisquer duas palavras, apenas "TROQUE" e "TUDO", nessa ordem. Este fato ainda não está representado na árvore. Para representá-lo, os nós (palavras) possuem um atributo, cada, contendo o NUMERO.DO.TOKEN da palavra correspondente.

Outros tipos de nós também possuem atributos, mas estes são usados no processo de substituição, não no reconhecimento dos Trechos-substituíveis.

IV.5.4 Tabela de declarações

Os Trechos-substituíveis devem ser encontrados no interior das Definições. Ao ser detetado um início de Trecho-substituível, o reconhecimento deste deve ser feito dirigido pela árvore associada à Declaração correspondente. Porém, como detetar esse início? Uma alternativa era ter-se um algoritmo que, percorrendo todas as árvores, buscasse as palavras que iniciassem os Trechos-substituíveis correspondentes a essas árvores, para que a palavra corrente no reconhecimento de uma Definição fosse comparada com aquelas palavras.

Entretanto, como foi visto, cada Sintaxe possui um palavra que lhe serve de nome. Esta é a única palavra que inicia o reconhecimento através dessa Sintaxe. Resumindo, cada Declaração-de-substituição tem uma única palavra que pode iniciar os Trechos-substituíveis por essa Declaração. A solução empregada foi a criação de uma tabela que associasse cada nome de substituição à árvore correspondente a essa substituição. Isto, além de evitar o percorrimento de cada árvore, permitiu que outras informações referentes às Declarações fossem colocadas nessa mesma tabela.

Essa tabela é manipulada através de duas rotinas:

DECORE.DECLARACAO: insere uma entrada na tabela, acusando erro se já houver alguma outra entrada com o mesmo nome. Isto proíbe que duas declarações possuam o mesmo nome.

INICIO.DE.TRECHO.SUBSTITUIVEL: função que informa se a palavra corrente serve como início de um Trecho.substituível, isto é, se existe uma entrada na tabela com essa palavra no campo nome. Se existir, essa entrada é copiada para a área DADOS.DA.DECLARACAO.

Exemplo IV.12:

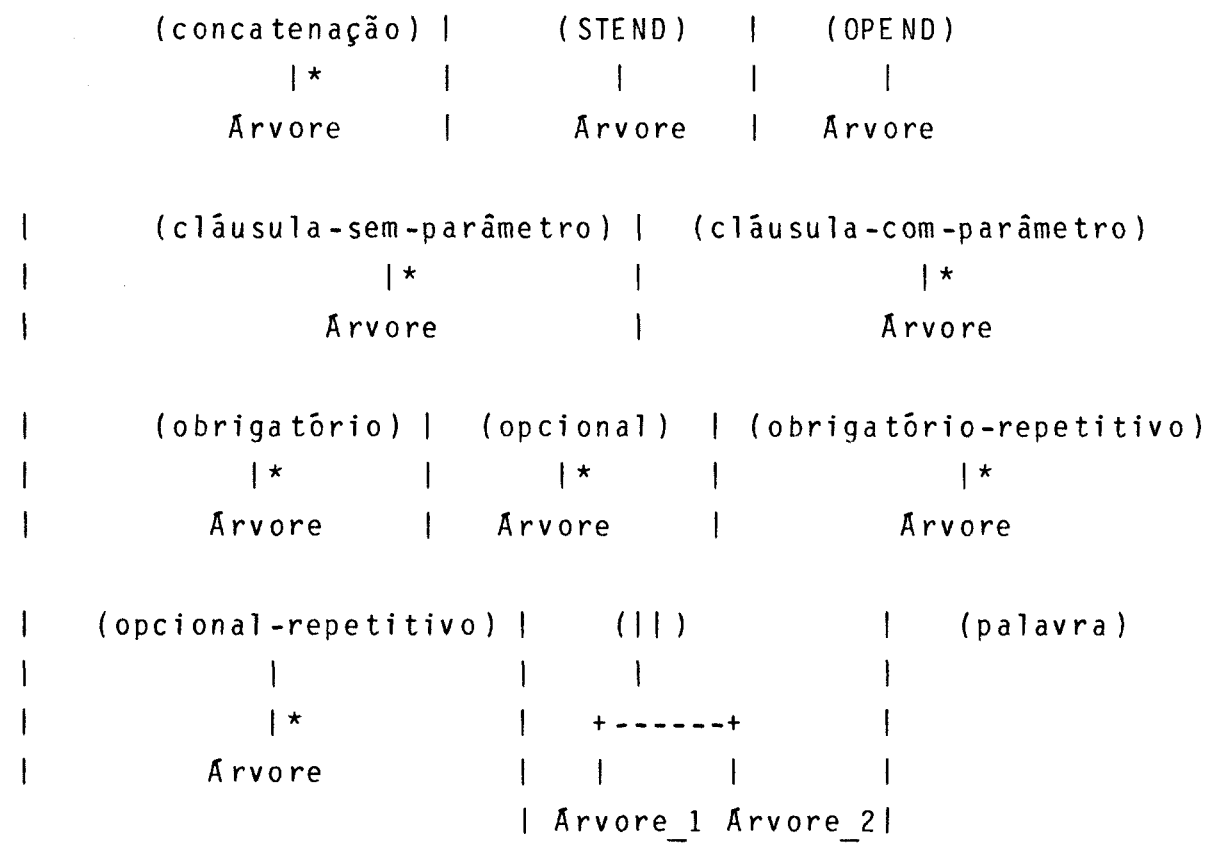
```
procedure DECLARACAO;
   begin
     word NOME.DA.DECLARACAO;
     ...
     word ENDER.ARVORE.DECLARACAO;
     ...
     if TIPO.DO.TOKEN = PALAVRA then
        begin
          ...
          NOME.DA.DECLARACAO := NUMERO.DO.TOKEN;
          ...
          SINTAXE;
          ENDER.ARVORE.DECLARACAO := ENDER.ARVORE;
          if TOKEN (TSTEND) then
              (ENDER.ARVORE.DECLARACAO := CRIA(NO.STEND))...
          else if TOKEN (TOPEND) then
              (ENDER.ARVORE.DECLARACAO := CRIA(NO.OPEND))...
          else ...;
          ...
          DECORE.DECLARACAO(NOME.DA.DECLARACAO,
                            ENDER.ARVORE.DECLARACAO,...)
          ...
        end
     else ...;
     ...
end DECLARACAO;
```

IV.5.5 Algoritmo de reconhecimento de Trechos-substituíveis

Ao ser detetado um início de Trecho-substituível, o reconhecimento deste deve ser feito dirigido pela árvore correspondente. Esta árvore contém as informações necessárias a esse reconhecimento.

As árvores sintáticas geradas obedecem à gramática simplificada abaixo, embora nem todas as árvores que respeitem essa gramática possam ser geradas por uma Declaração.

Árvore ::=



Se a raiz da árvore que dirige o reconhecimento de um Trecho-substituível for um nó (concatenação), o reconhecimento através desta árvore será composto do reconhecimento em sequência através de cada uma das sub-árvores filhas deste nó (concatenação).

Se a raiz for um nó (STEND) ou (OPEND), a sub-árvore dirigirá o reconhecimento inicialmente e, após isto, será verificado

se a palavra corrente pertence ao conjunto STEND ou OPEND.

Se for um nó (cláusula-sem-parâmetro), o reconhecimento será idêntico ao de árvore com raiz (concatenação). Se for (cláusula-com-parâmetro), após o reconhecimento através das sub-árvores, deve ser reconhecido um parâmetro efetivo. Este pode conter tudo que uma Definição pode ter, inclusive novos Trechos-substituíveis. Então basta fazer uma chamada para a rotina DEFINICAO.

Estes são os nós que podem ser raízes das árvores das Declarações. Com excessão do (STEND) e (OPEND), também podem ser raízes de sub-árvores, que serão reconhecidas do mesmo modo. Devido a isso, a rotina que reconhece Trecho-substituível é recursiva e reconhece também árvores com raízes distintas das acima mencionadas.

Ou seja, se a raiz for (palavra), o reconhecedor exigirá a palavra cujo numero-de-token está guardado como atributo do nó (palavra).

Se a raíz for (||). a árvore terá 2 filhos (Árvore_1 e Árvore_2). Se a palavra corrente for um início válido da Árvore_1, serão reconhecidas Árvore_1 e Árvore_2, nesta ordem. Senão, a ordem será invertida.

Se for (obrigatório), será pesquisada de qual sub-árvore a palavra corrente é um início válido. Esta sub-árvore será reconhecida. Se, ao contrário, a palavra não for início válido das Sub-árvores, será acusado um erro, já que o nó (obrigatório) indica a obrigatoriedade da existência no Trecho-substituível de alguma das alternativas.

Se for (opcional), esse erro não será assinalado quando a palavra não servir como início das sub-árvores. Para (opcional-repetitivo), enquanto for possível o reconhecimento de alguma sub-árvore, este será feito. No (obrigatório-repetitivo) é necessário que pelo menos um reconhecimento seja feito.

O reconhecedor de Trechos-substituível terá a forma:

```
procedure TRECHO.SUBSTITUIVEL (word ARVORE);
   begin
     procedure NO.GRUPO;
        begin ... end;
     procedure NO.BARRA.BARRA;
        begin ... end;
     procedure CONCATENA;
        begin ... end;
     case ARVORE^TIPO.DO.NO of
        begin
          & NO.PALAVRA:
                ...;        & EXIJA PALAVRA ARVORE^ATRIBUTO
          & NO.STEND:
              begin
                CONCATENA;
                PERTENCA(CONJUNTO.STEND);
              end;
          & NO.OPEND:
              ...   & SEMELHANTE A NO.STEND
          & NO.OPCIONAL;
              NO.GRUPO;
          & NO.OBRIGATORIO:
              NO.GRUPO;
          & NO.OPCIONAL.REPETITIVO:
              NO.GRUPO;
          & NO.OBRIGATORIO.REPETITIVO:
              NO.GRUPO;
          & NO.BARRA.BARRA:
              NO.BARRA.BARRA;
          & NO.CLAUSULA:
              CONCATENA;
          & NO.CLAUSULA.PARAMETRO:
              begin
                 CONCATENA;
                 DEFINICAO;
              end;
        end;
end TRECHO.SUBSTITUIVEL;
```

```
procedure CONCATENA;
   begin
     word FILHO.CORRENTE;
     ...
     FILHO.CORRENTE := ARVORE^FILHO;
     repeat
          TRECHO.SUBSTITUIVEL (FILHO.CORRENTE)
     until (FILHO.CORRENTE := FILHO.CORRENTE^IRMAO) = NULO;
     ...
   end CONCATENA;

procedure NO.BARRA.BARRA;
   begin
     if INICIO.VALIDO (ARVORE^FILHO) then
        begin
          TRECHO.SUBSTITUIVEL (ARVORE^FILHO);
          TRECHO.SUBSTITUIVEL (ARVORE^FILHO^IRMAO);
        end
     else
        begin
          TRECHO.SUBSTITUIVEL (ARVORE^FILHO^IRMAO) ;
          TRECHO.SUBSTITUIVEL (ARVORE^FILHO);
        end;
end NO.BARRA.BARRA;
```

A rotina NO.GRUPO sintetiza os nós de grupo e não foi realizada como a intuição pediria devido a necessidade de inserirem-se as chamadas às rotinas semânticas.

```
procedure NO.GRUPO;
   begin
     word ENDER.ARV.ALTERNATIVA.CORRENTE;
     word ENDER.ARV.ALTERNATIVA.CERTA;
     byte NAO.RECONHECEU;
     byte TENTA.RECONHECER.MAIS;
     NAO.RECONHECEU := TENTA.RECONHECER.MAIS := VERO;
     repeat  & RECONHECIMENTO
        begin
          ENDER.ARV.ALTERNATIVA.CORRENTE := ARVORE^FILHO;
          ENDER.ARV.ALTERNATIVA.CERTA    := NULO;
          repeat
             if INICIO.VALIDO(ENDER.ARV.ALTERNATIVA.CORRENTE)
                                                      then
                ENDER.ARV.ALTERNATIVA.CERTA :=
                             ENDER.ARV.ALTERNATIVA.CORRENTE
             else ...
          until (ENDER.ARV.ALTERNATIVA.CORRENTE :=
                  ENDER.ARV.ALTERNATIVA.CORRENTE^IRMAO)=NULO;
          if ENDER.ARV.ALTERNATIVA.CERTA = NULO then
             TENTA.RECONHECER.MAIS := FALSO
          else begin ...
             TRECHO.SUBSTITUIVEL(ENDER.ARV.ALTERNATIVA.CERTA)
             NAO.RECONHECEU := FALSO;
             end;
          if ARVORE^TIPO.DO.NO = OPCIONAL     or
             ARVORE^TIPO.DO.NO = OBRIGATORIO then
               TENTA.RECONHECER.MAIS := FALSO;
        end
        until not TENTA.RECONHECER.MAIS;
     if NAO.RECONHECEU then
        begin
          if ARVORE^TIPO.DO.NO= NO.OBRIGATORIO or
             ARVORE^TIPO.DO.NO= NO.OBRIGATORIO.REPETITIVO
          then ERRO(CHAMCR INV);
        end
     else ...;
end NO.GRUPO;
```

Essa rotina consiste de duas partes. A primeira é um laço de reconhecimento controlado pela variável TENTA.RECONHECER.MAIS, inicialmente ligada e que é desligada quando não for mais possível reconhecer nenhuma sub-árvore ou se o tipo-de-nó não era repetitivo.

A segunda parte acusa um erro, se o nó era do tipo obrigatório e não houve nenhum reconhecimento na primeira parte. Ela usa a variável NAO.RECONHECEU, que é desligada quando há um reconhecimento de alguma sub-árvore.

O laço da primeira parte também se compõe de duas partes.

Numa é procurada a sub-árvore do qual a palavra corrente pode ser um início válido. A variável ENDER.ARV.ALTERNATIVA.CERTA contém o endereço dessa árvore ou NULO, se nenhuma servir. Na outra parte é feito o reconhecimento e desligada a variável TENTA.RECONHECER.MAIS quando devido.

IV.5.6 Pesquisa de início válido

O algoritmo para verificação de que uma palavra possa ser um início válido da sintaxe descrita por uma árvore também é interessante. A rotina INICIO.VALIDO, que faz esse trabalho, é chamada para todos os tipos de árvore, exceto os de raiz STEND ou OPEND .

Se a árvore for composta apenas de um nó (palavra), o resultado da rotina será o resultado da comparação entre a palavra investigada e o atributo desse nó (palavra).

Se a raiz da árvore for um nó (concatenação), (cláusula-sem-parâmetro) ou (cláusula-com-parâmetro), a rotina pesquisará se a palavra é um início válido da sub-árvore mais à esquerda.

Se for um nó de grupo, pesquisará se a palavra é um início de qualquer sub-árvore.

Portanto a rotina INICIO.VALIDO é uma função recursiva.

```
byte procedure INICIO.VALIDO (word ARVORE);
   begin
     word FILHO.CORRENTE;
     procedure INICIOS.ALTERNATIVOS;
        begin
          FILHO.CORRENTE := ARVORE^FILHO;
          repeat
             RESULT := INICIO.VALIDO (FILHO.CORRENTE)
          until (FILHO.CORRENTE := FILHO.CORRENTE^IRMAO) =
                NULO or RESULT;
        end;
          case ARVORE^TIPO.DO.NO of
              begin
               & NO.PALAVRA:
                   RESULT:=(ARVORE^ATRIBUTO =
                                PALAVRA.TESTADA);
               & NO.STEND:
                   ;
               & NO.OPEND:
                   ;
               & NO.OPCIONAL:
                   INICIOS.ALTERNATIVOS;
               & NO.OBRIGATORIO:
                   INICIOS.ALTERNATIVOS;
               & NO.OPCIONAL.REPETITIVO:
                   INICIOS.ALTERNATIVOS;
               & NO.OBRIGATORIO.REPETITIVO:
                   INICIOS.ALTERNATIVOS;
               & NO.BARRA.BARRA;
                   INICIOS.ALTERNATIVOS;
               & NO.CONCATENACAO;
                   RESULT := INICIO.VALIDO (ARVORE^FILHO);
               & NO.CLAUSULA.SEM.PARAMETRO:
                   RESULT := INICIO.VALIDO (ARVORE^FILHO);
               & NO.CLAUSULA.PARAMETRO:
                   RESULT := INICIO.VALIDO (ARVORE^FILHO);
          end;
end INICIO.VALIDO;
```

IV.5.7 Término de parâmetro efetivo

No reconhecimento de trechos dirigidos por árvores com raiz (cláusula-com-parâmetro) (vide IV.5.5), ficou estabelecido que os parâmetros efetivos são reconhecidos pela própria rotina DEFINICAO. Para tanto, essa rotina necessita reconhecer quando uma palavra provoca o fim do reconhecimento de um parâmetro efetivo, para que ela possa retornar. A descoberta do conjunto de palavras que terminam um parâmetro exige uma investigação na árvore, que não se limita a sub-árvore da (cláusula-com-parâmetro). Quem determina que trechos devem ser investigados é o nó pai dessa sub-árvore.

Se ele for um nó (concatenação) a atitude será muito distinta da se ele for (obrigatório-repetitivo).

Um ponteiro para árvores adicional, chamado PROXIMO, faz parte de cada nó da árvore, de modo a facilitar esse trabalho. Ele é usado da seguinte maneira:

Para verificar se uma palavra termina a sintaxe de uma sub-árvore, olha-se para quem o ponteiro PROXIMO da raiz dessa sub-árvore aponta:

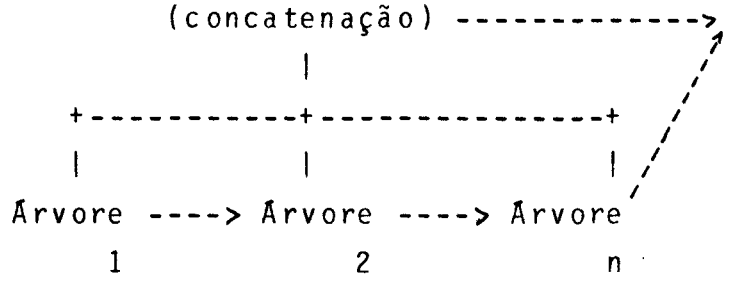
- se for para um nó (STEND) ou (OPEND), basta verifica se a palavra faz parte do CONJUNTO.STEND ou CONJUNTO.OPEND;
- se for um nó (concatenação), (cláusula-sem-parâmetro), (cláusula-com-parâmetro), (obrigatório), (obrigatório-repetitivo) ou (palavra), basta verificar se a palavra é um início válido da sub-árvore cuja raiz é esse nó.
- se for para um nó (opcional) ou (opcional-repetitivo), verifica-se se a palavra é início válido da sub-árvore cuja raiz é esse nó. Se for, a palavra termina a sintaxe. Se não for, deve-se verificar se a palavra termina a sintaxe dessa sub-árvore, já que esta é opcional.

```
byte procedure TERMINADOR (word ARVORE);
  begin
    ...
    RESULT := if ARVORE^PROXIMO^TIPO.DO.NO = NO.OPCIONAL or
                 ARVORE^PROXIMO^TIPO.DO.NO =
                                        NO.OPCIONAL.REPETITIVO then
                 if INICIO.VALIDO(ARVORE^PROXIMO) then
                      VERO
                 else TERMINADOR (ARVORE^PROXIMO)
               else if ARVORE^PROXIMO^TIPO.DO.NO = NO.STEND
                      then PERTENCE (CONJUNTO.STEND)
               else if ARVORE^PROXIMO^TIPO.DO.NO = NO.OPEND
                      then PERTENCE (CONJUNTO.OPEND)
               else          INICIO.VALIDO(ARVORE.PROXIMO);
    TERMINADOR := RESULT;
    ...
end TERMINADOR;
```

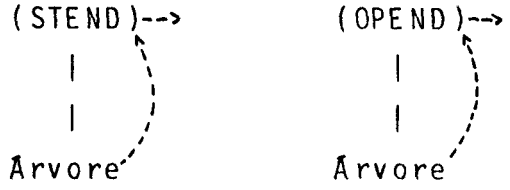
Ao terminar o reconhecimento do Padrão-inicial de uma Declaração-de-substituição, quando se tem a árvore sintática gerada, é chamada a rotina PREENCHE.PROXIMO, que percorrerá essa árvore, da raiz até as folhas, gerando esses ponteiro.

Inicialmente todos os ponteiros PROXIMOS da árvore são NULOS. A rotina PREENCHE.PROXIMO é chamada para a raiz da árvore.

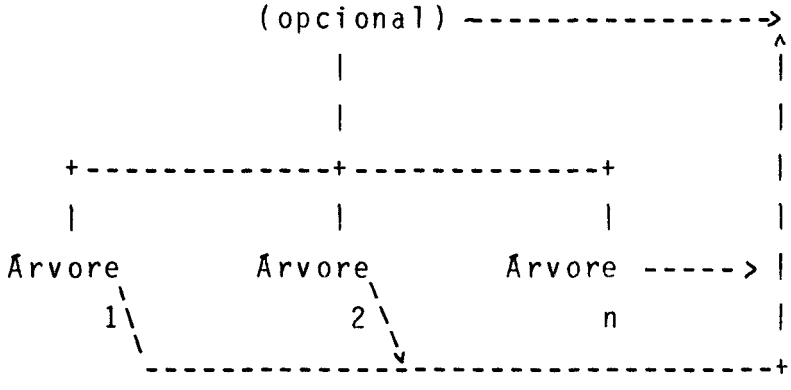
Quando a rotina PREENCHE.PROXIMO for chamada para uma árvore (ou subárvore) de raiz (concatenação), o ponteiro PROXIMO de cada sub-árvore desse nó apontará para o seu irmão da direita. O ponteiro PROXIMO do filho mais à direita herdará o mesmo valor do PROXIMO do seu pai.



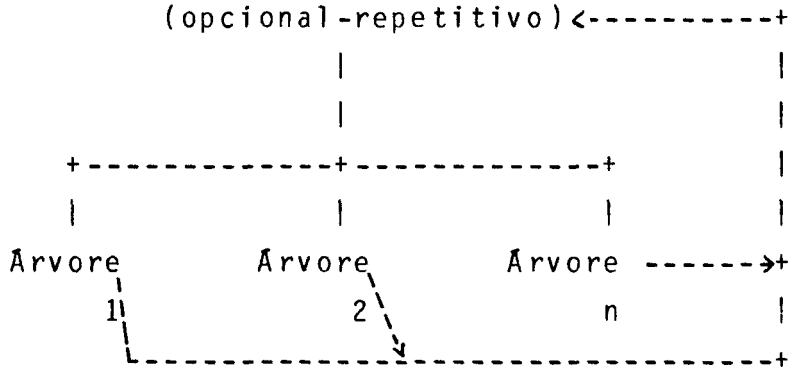
Se a PREENCHE.PROXIMO for chamada para uma árvore de raiz (STEND) ou (OPEND), o ponteiro PROXIMO da raiz da sub-árvore filha apontará para o seu prôprio pai. Será feita uma chamada recursiva para a sub-árvore filha.



Se ela for chamada para uma raiz (opcional) ou (obrigatório), o PROXIMO dessa raiz será copiado para cada sub-árvore filha, para qual será feita uma chamada recursiva.

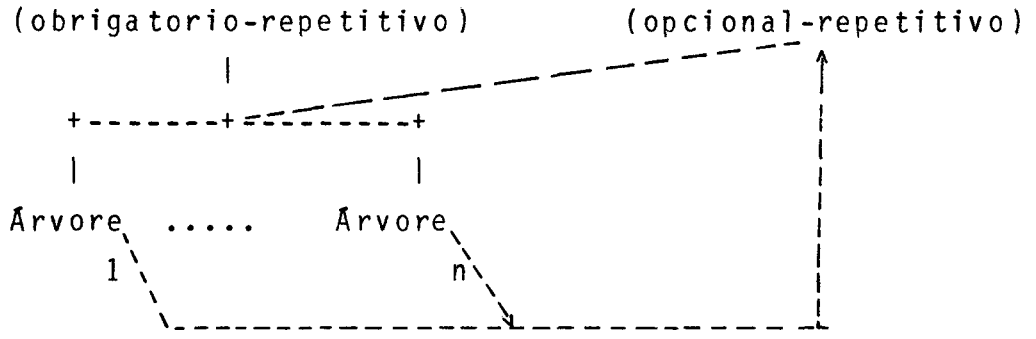


Se a raiz for (opcional-repetitivo), o prôximo de cada filha apontará para essa raiz e será feita uma chamada recursiva para cada uma delas.



Se for (obrigatório-repetitivo) ou (||), será gerado um nô falso, fora da árvore, de tipo (opcional-repetitivo), cujos pon-

teiros para o FILHO mais à esquerda e para o PROXIMO serão cópias da raiz. Cada sub-árvore apontará para esse nó falso e será percorrida recursivamente.



Esse artifício é mais fácil de compreender se considerarmos que

{W}... equivale a W [W]...

e que

W || V equivale a {W | V}..., onde cada alternativa só poderá ser escolhida uma única vez.

Se a raiz da árvore for (palavra), (cláusula-com-parâmetro) ou (cláusula-sem-parâmetro) a rotina PREENCHE.PROXIMO simplesmente retorna, pois não precisa propagar nada para os filhos destes nós.

A conjunção dos ponteiros, PROXIMOS com a rotina TERMINADOR permite que, dado um nó (cláusula-com-parâmetro), as palavras que terminam o parâmetro dessa cláusula fiquem bem determinadas.

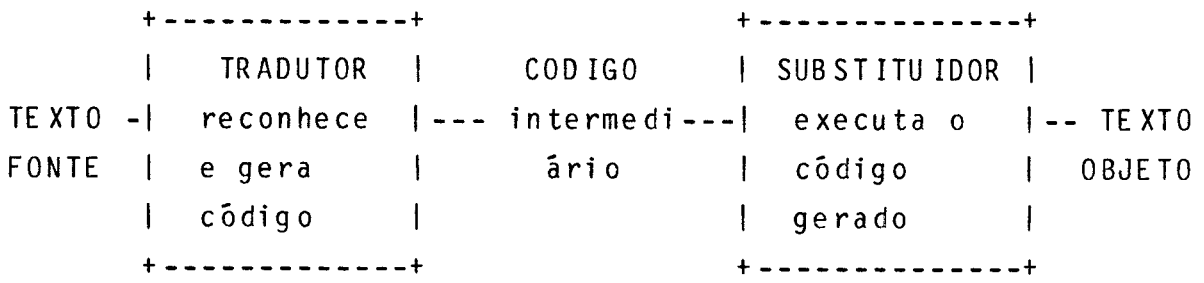
```
procedure PREENCHE.PROXIMO (word ARVORE)
    begin
      word FILHO.CORRENTE;
      word NO.FALSO;
      procedure FILHOS.PROXIMOS (word PARAMETRO);
          begin
            repeat
              begin
                FILHO.CORRENTE^PROXIMO := PARAMETRO;
                PREENCHE.PROXIMO(FILHO CORRENTE);
              end;
            until (FILHO.CORRENTE:=FILHO.CORRENTE^IRMAO)= NULO;
          end FILHOS.PROXIMO;
```

```
  FILHO.CORRENTE := ARVORE^FILHO;
  if ARVORE^TIPO.DO.NO=NO.OPCIONAL or
     ARVORE^TIPO.DO.NO=NO.OBRIGATORIO then
               FILHOS.PROXIMO (ARVORE^PROXIMO)
  else if ARVORE^TIPO.DO.NO = NO.OPCIONAL.REPETITIVO then
          FILHOS.PROXIMO(ARVORE)
  else if ARVORE^TIPO.DO.NO = NO.OBRIGATORIO.REPETITIVO or
          ARVORE^TIPO.DO.NO = NO.BARRA.BARRA          then
       begin
         (NO.FALSO:=CRIA(NO.OPCIONAL.REPETITIVO))^FILHO:=
                                  ARVORE.FILHO;
          NO.FALSO^PROXIMO := ARVORE^PROXIMO;
          FILHOS.PROXIMO (NO.FALSO);
       end
  else if ARVORE^TIPO.DO.NO = NO.CONCATENACAO or
          ARVORE^TIPO.DO.NO = NO.STEND        or
          ARVORE^TIPO.DO.NO = NO.OPEND        then
       begin
         while FILHO.CORRENTE^IRMAO    NULO do
          begin
           FILHO.CORRENTE^PROXIMO:=FILHO.CORRENTE^IRMAO;
           PREENCHE.PROXIMO (FILHO.CORRENTE);
           FILHO.CORRENTE := FILHO.CORRENTE^IRMAO;
          end;
       FILHO.CORRENTE^PROXIMO := if ARVORE^TIPO.DE.NO =
                                              NO.CONCATENACAO
                                    then ARVORE^PROXIMO
                                    else ARVORE;
       PREENCHE.PROXIMO (FILHO.CORRENTE);
     end;
end PREENCHE.PROXIMO;
```

IV.6 Substituição

Após ser reconhecido um Trecho-substituível, ele deve ser substituído pelo Padrão-final correspondente. No interior desse Padrão-final pode haver diversas Referências-a-parâmetro, Testes ou novos Trechos-substituíveis. Trechos-substituíveis também podem referir-se à mesma Declaração-de-substituição. Tudo isso torna o processo de substituição muito complexo para ser encarado simultaneamente com o reconhecimento. Ou seja, quando forem reconhecidos Trechos-substituíveis referentes a uma Declaração-de-substituição, é conveniente que esta já tenha sido analisada e sintetizada em componentes de mais fácil manipulação do que os textos que a compõem, para que as substituições desses Trechos, por si um trabalho difícil, possam ser feitas sem repetir o trabalho de reconhecimento do Padrão-final dessa Declaração.

Estas considerações levam à solução adotada: durante o reconhecimento, são geradas comandos codificados, que sintetizam o trabalho de substituição. Estes comandos não precisam ser executados logo que gerados. Podem ser arquivados para execução após o reconhecimento total do Programa fonte.



Nada impede, porém, que o TRADUTOR e o SUBSTITUIDOR possam processar em paralelo, num esquema de co-rotinas. Mesmo assim, é conveniente que sejam encarados como funcionando em série, o que nos permite conceituar melhor a geração e a execução do código, graças a idealização de arquivos abstratos, que poderiam não existir ou não funcionar exatamente como foram idealizados.

O arquivo INSTRUCOES, que contém os tais comandos e faz parte da interface entre o TRADUTOR e o SUBSTITUIDOR, é um exemplo do que foi dito. O módulo TRADUTOR do processador LPSE gera

os comandos como se gravasse nesse arquivo através da rotina GRAVA.INSTRUCAO. O módulo SUBSTITUIDOR lê esses comandos, para executá-los, através da rotina LE.INSTRUCAO. A idéia é: alterando-se as duas rotinas podemos ter os dois módulos rodando em paralelo, ou o arquivo possuir múltiplas áreas de entrada/saída ou, ainda, o arquivo simplesmente não existir, sendo guardado na memória tudo o que for necessário para a substituição do programa fonte. O importante é que, tanto o TRADUTOR, quanto o SUBSTITUIDOR sejam feitos como se esse arquivo existisse, abstraindo-se da implementação das rotinas GRAVA.INSTRUCAO e LE.INSTRUCAO.

O SUBSTITUIDOR grava (abstratamente) o texto objeto no arquivo SIMBOLOS, através de chamadas à rotina GRAVA.SIMBOLO. Este arquivo, portanto, após a substituição de todo o texto Programa, contém o Texto substituído. Uma palavra que apareça no Padrão-final de alguma Declaração pode ser repetida no Texto substituído toda vez que houver Trecho-substituível por essa Declaração.

O SUBSTITUIDOR, que lê o arquivo INSTRUCOES, executando-as, assemelha-se à UCP de uma máquina, cujas instruções serão descritas adiante.

IV.6.1 O SUBSTITUIDOR

O SUBSTITUIDOR trabalha sobre uma memória de dados linear sequencial cujas posições são endereçadas por 0, 1, ..., TOPO-1, cada uma delas podendo conter endereços para essa memória, endereços para a memória de instruções e valores lógicos (VERO ou FALSO). Esta memória de dados é chamada de PILHA. O maior endereço de uma posição de memória de dados em uso durante a execução é TOPO -1. Inicialmente a pilha está vazia (TOPO = 0).

O SUBSTITUIDOR também contém uma memória de instruções linear sequencial (o arquivo INSTRUCOES) com posições endereçadas por 1, 2, ..., n. O conteúdo desta memória de instruções e seu tamanho não são alterados durante a substituição. A instrução de endereço 1 é a primeira a ser executada.

A variável PROXIMA.INSTRUCAO contém o endereço da próxima instrução a ser executada. Esta instrução é obtida por uma chamada à sub-rotina LE.INSTRUCAO, que, além de atualizar a variável PROXIMA.INSTRUCAO, preenche as variáveis CODIGO e OPERANDO que descrevem a instrução a ser executada.

Duas variáveis, BASE.INTERNA e BASE.EXTERNA, servem para endereçar a memória de dados. Como um Trecho-substituível correspondente a uma Declaração pode estar dentro do Padrão-final dessa própria Declaração, as informações correspondentes a um Trecho-substituível (endereço da instrução a ser executada após a substituição desse Trecho, endereço das instruções dos parâmetros, indicações de se uma Cláusula foi empregada nesse Trecho e se ela se repete) não podem ter posição fixa na memória de dados. Se tivessem, essas informações se misturariam no caso recursivo acima citado.

Essas informações são empilhadas em blocos associados aos Trechos-substituíveis. Portanto, ao ser feita uma substituição definida numa Declaração, serão executadas instruções que referir-se-ão a campos do bloco corrente.

A BASE.INTERNA fornece o endereço do bloco correspondente ao Trecho que está sendo substituído naquele instante. A BASE.EXTERNA dá o endereço do bloco que está sendo montado para posterior substituição.

Exemplo:

```
macro UM   ($)  define ... endmacro
macro DOIS ($A) define UM ($A) endmacro
```

No Trecho-substituível "UM ($A)", o parâmetro $A pertence à Declaração de nome DOIS, cujo bloco de informações é endereçado pela BASE.INTERNA, durante a execução da substituição indicada nessa Declaração. Esse parâmetro, entretanto, está fazendo parte do parâmetro efetivo desse Trecho, que corresponde à Declaração de nome UM, cujo bloco de informações é endereçado pela BASE EXTERNA, durante a montagem desse bloco.

Para que as instruções tenham tamanho fixo, com um único operando, se fez necessária a existência de um ACUMULADOR, que retivesse resultados parciais entre duas, ou mais, instruções.

O algoritmo SUBSTITUIDOR é apresentado a seguir, através de seus traços mais gerais. Inicialmente, a memória de dados está vazia, PROXIMA.INSTRUCAO=1 e TOPO=BASE.INTERNA=BASE.EXTERNA=0. O programa expandirá e contrairá a memória de dados através de suas instruções e parará ao encontrar uma instrução FINALIZA.

```
word PROXIMA.INSTRUCAO   = 1;
word ACUMULADOR;
word TOPO                = 0;
word BASE.INTERNA        = 0;
word BASE.EXTERNA        = 0;
word (TAMANHO.DA.PILHA)PILHA;
repeat
   begin
     LE.INSTRUCAO;
     case CODIGO of
       begin
         ... & executa a instrução
             & essa execução pode modificar PROXIMA.INSTRUCAO
       end;
   end;
```

IV.6.2 Instruções

IV.6.2.1 Entrada e Saída

A instrução TRANSMITE comanda a saída de um símbolo, ou seja, grava-o no arquivo de Texto substituído. O operando dessa instrução é o próprio número-de-token a ser transmitido.

Simbolicamente, a execução dessa instrução pode ser descrita assim:

```
TRANSMITE(OPERANDO): Dá saída ao operando imediato
    GRAVA.SIMBOLO(OPERANDO);
```

O TRADUTOR, por seu lado, gera essa instrução quando encontra um Elemento-insubstituível. Ou seja, ao analisar numa Definição uma palavra precedida por ORIGINAL, ou uma palavra que não inicie um Trecho-substituível, é feita uma chamada-a-rotina:

```
GRAVA.INSTRUCAO (TRANSMITE, NUMERO.DO.TOKEN);
```

Também é necessária uma instrução FINALIZA, cujo operando é desprezado, e que simplesmente para a execução do SUBSTITUIDOR. Essa instrução é gerada quando, na análise de um Programa, é encontrado o FIM.DE.FONTE.

IV.6.2.2 Trecho-substituível

O reconhecimento de um Trecho-substituível inicia quando, ao analisar uma Definição, o TRADUTOR identifica que a palavra corrente é o nome de alguma Declaração. A partir daí, é feito o reconhecimento desse Trecho-substituível dirigido pela árvore que sintetiza a sintaxe declarada.

Como essa sintaxe pode permitir parâmetros, omissões de trechos ou repetições de trechos e o Padrão-final pode fazer referências a esses parâmetros ou testes, é importante que sejam guardadas informações sobre como esse Trecho-substituível constitui-se, para serem usadas na substituição desse Trecho.

Ao iniciar o reconhecimento de um Trecho-substituível é gerada uma instrução ALOCA, que reserva na memória de dados uma área para conter essas informações. Durante o reconhecimento desse Trecho, são geradas instruções que servem para preencher essa área. Ao terminar esse reconhecimento, é gerada a instrução CHAMA, que, quando executada pelo SUBSTITUIDOR, provoca a execução das instruções no Padrão-final da Declaração correspondente a esse Trecho. A última dessas instruções é uma RETORNA, que desaloca essa área.

A maneira como essa área é alocada e desalocada obedece à disciplina "último a entrar, primeiro a sair", característica das pilhas, pelo motivo enunciado no parágrafo IV.6.1.

Durante o preenchimento dessa área, ela é endereçada pela variável BASE.EXTERNA. Durante a substituição, pela BASE.INTERNA.

Os campos dessas áreas são chamados de "células". Cada Declaração tabelada tem associada, além do nome e da árvore sintática, o número de células necessárias para a substituição dos Trechos-substituíveis por essa Declaração e o endereço da primeira instrução gerada pelo seu Padrão-final.

Portanto, no final da rotina DECLARACAO há uma chamada do tipo:

```
DECORE.DECLARACAO (NOME.DECLARACAO,ENDER.ARVORE.DECLARACAO,
    ENDER.INSTR.INICIAL.DECLARACAO,NUMERO.DE.CELULAS);
```

A função NOME.DE.DECLARACAO, que informa se uma palavra é nome de alguma declaração tabelada, preenche as variáveis ENDER.ARVORE.INFORMADO, ENDER.INSTR.INFORMADO e NUMERO.DE.CELULAS.INFORMADO com as restantes informações tabeladas para essa palavra.

Essa função é chamada na DEFINICAO:

```
if NOME.DE.DECLARACAO (NUMERO.DO.TOKEN) then
  begin
    GRAVA.INSTRUCAO (ALOCA,NUMERO.DE.CELULAS.INFORMADO);
    ENDER.INSTR.INICIAL.CHAMADO:=ENDER.INSTR.INICIAL.INFORMADO;
    TRECHO.SUBSTITUIVEL (ENDER.ARVORE.INFORMADO);
    GRAVA.INSTRUCAO (CHAMA, ENDER.INSTR.INICIAL.CHAMADO);
  end;
else
  begin
    GRAVA.INSTRUCAO (TRANSMITE, NUMERO.DO.TOKEN);
    ...
  end;
```

A instrução ALOCA marca o início de um novo Trecho-substituível. Enquanto suas células estiverem sendo preenchidas, elas são endereçadas através da BASE.EXTERNA. Este Trecho (T1) pode estar contido até em um parâmetro de outro Trecho-substituível (T2). Neste caso, a BASE.EXTERNA, que aponta para a área de T2, deve passar a apontar para a área de T1, quando esta for alocada. No retorno da substituição a BASE.EXTERNA de T2 é restaurada.

A instrução CHAMA inicia a substituição do Trecho. A partir dela, as células passam a ser endereçadas à partir da BASE.INTERNA. Como essa substituição (S1) pode estar contida na Definição de outra substituição (S2), a BASE.INTERNA, que apontava para a área de S2, deve passar a apontar para a área de S1, quando esta substituição for chamada.

A instrução RETORNA deve desalocar a área de células e restaurar a BASE.INTERNA e a BASE.EXTERNA. A próxima instrução a ser executada é a que seguia a chamada desta substituição.

Portanto, a área associada a um Trecho-substituível de N células tem tamanho N + 3, pois nela devem ser guardados os valores da BASE.INTERNA, BASE.EXTERNA e da PROXIMA.INSTRUCAO após a chamada. Podemos descrever a memória de dados da seguinte maneira:

```
Pilha ::= Área*
Área ::= Célula* Base-externa Base-interna Próxima-instrução
```

A instrução ALOCA tem como operando o número de células da área. Sua execução pode ser descrita assim:

```
ALOCA (OPERANDO): reserva área com OPERANDO células
     TOPO := TOPO + OPERANDO + 3;
     PILHA (TOPO-3) := BASE.EXTERNA;
     BASE.EXTERNA := TOPO;
```

A instrução CHAMA tem como operando o endereço da instrução que inicia a substituição. Sua descrição é:

```
CHAMA (OPERANDO): chama  substituição do endereço OPERANDO
     PILHA (TOPO -2) := BASE.INTERNA;
     PILHA (TOPO -1) := PROXIMA.INSTRUCAO;
     BASE.INTERNA := TOPO;
     PROXIMA.INSTRUCAO:=LE.ENDERECO(OPERANDO);
```

Os endereços gerados nas instruções são todos indiretos, cada um deles tem um número de ordem associado que é o número do registro no arquivo ENDERECOS que contêm endereços de instruções efetivos. A função LE.ENDERECO, para um número de ordem passado como parâmetro, fornece o endereço efetivo, lendo o registro correspondente.

Na instrução RETORNA, o operando é o número de células da área a ser desalocada.

```
RETORNA(OPERANDO): retorna para quem chamou
      PROXIMA.INSTRUCAO := PILHA(TOPO-1);
      BASE.INTERNA := PILHA (TOPO-2);
      BASE.EXTERNA := PILHA (TOPO-3);
      TOPO := TOPO - OPERANDO - 3;
```

IV.6.2.3 Endereços indiretos e Desvios

Como foi dito, todos os endereços gerados nas instruções são indiretos. Isto porque durante a geração das instruções existem pontos onde há necessidade de gerar-se instruções que se refiram a um endereço futuro, ainda não determinado. Sendo os endereços indiretos, basta gerar nas instruções um número de ordem do endereço indefinido e, quando esse endereço for conhecido, fazer-se a associação do número-de-ordem com o endereço efetivo.

Para implementar isto, foi definido um arquivo ENDERECOS manipulado pelas rotinas GRAVA.ENDERECO e LE.ENDERECO. A primeira recebe um número de ordem como parâmetro e grava o endereço da próxima instrução a ser gerada, no registro que tem esse número de ordem. A segunda é uma função que fornece o endereço, dado o número de ordem. Ela, simplesmente, lê o registro com esse número de ordem.

Há também a função NUMERO.DE.ORDEM, que, a cada vez que é chamada, fornece um novo número de ordem. Foi implementada assim:

```
word N = 0;
word procedure NUMERO.DE.ORDEM;
   begin
     NUMERO.DE.ORDEM := N := N + 1;
   end;
```

Nas rotinas de reconhecimento dos não-terminais, também são necessárias variáveis-atributos para guardar esses números de ordem.

A instrução DESVIA tem como operando o número de ordem de uma instrução que deve ser a próxima a ser executada. Sua descrição:

```
DESVIA(OPERANDO): desvia para endereço OPERANDO
     PROXIMA.INSTRUCAO := LE.ENDERECO(OPERANDO);
```

Durante o reconhecimento das Declarações-de-substituição, são geradas instruções que só devem ser executadas na substituição de algum trecho. A primeira instrução que o SUBSTITUIDOR deve realizar será a primeira gerada pelo reconhecimento do Texto que segue as Declarações. Na verdade, ela será a segunda, pois a primeira será uma DESVIA para ela.

Exemplo IV.13:

```
procedure PROGRAMA;
  begin
   word ENDER.INSTR.TEXTO;
   ...
   GRAVA.INSTRUCAO (DESVIA, ENDER.INSTR.TEXTO:=NUMERO.DE.ORDEM);
   ...
   while TOKEN (TMACRO) do
     DECLARACAO;
   ...
   GRAVA.ENDERECO(ENDER.INSTR.TEXTO);
   ...
   DEFINICAO;
   ...
  end PROGRAMA;
```

Nesse exemplo, a chamada à GRAVA.INSTRUCAO gera uma DESVIA para o endereço ENDER.INSTR.TEXTO. Esse endereço é definido através de uma chamada à GRAVA.ENDERECO, logo após serem reconhecidas todas as Declarações.

IV.6.2.4 Parâmetro

Em uma Definição, podem aparecer Referências-a-parâmetro. Cada uma dessas Referências indica que o texto passado como parâmetro no Trecho-substituível deve substituir à Referência. É como se o parâmetro efetivo fosse chamado e suas instruções fossem executadas até uma RETORNA.

Para que a instrução RETORNA possa ser usada, uma chamada a parâmetro deve gerar na pilha uma área com nenhuma célula.

Exemplo IV.14:

```
macro M $Q;
    define
        ...
        $Q
        ...
    endmacro
macro P $Q;
    define
        ...
        M $Q + 1;
        ...
    endmacro
        ...
```

Nesse exemplo, quando a Declaração-de-substituição M é evocada no interior da Definição P, o controle do SUBSTITUIDOR passa para as instruções geradas na Declaração M. Ao encontrar a referência ao parâmetro $Q da Declaração M o controle volta para P, precisamente para o início do parâmetro efetivo, que é o trecho "$Q + 1". Nesse ponto, na pilha, a área de P está abaixo da de M, que, por sua vez, está abaixo de uma área sem células, contendo informações para retornar a M. As informações sobre o parâmetro $Q de P estão na terceira área, a contar do topo. A BASE.INTERNA, antes da chamada ao parâmetro da Declaração M, apontava para a área de M e não a de P. Para que, durante a exe-

cução do parâmetro formal "$Q + 1", o SUBSTITUIDOR acesse o $Q adequado, é necessário que a instrução de chamada a parâmetro restaure a BASE.INTERNA para quem havia chamado a Declaração-de-substituição em curso.

Cada parâmetro tem um nome. Durante o reconhecimento de uma Declaração, seus parâmetros são tabelados pelo nome. Cada vez que uma Declaração é chamada, é alocada uma área para ela. Nessa área há uma célula para cada Cláusula, com parâmetro ou sem. Essa célula é usada para indicar se a Cláusula está presente ou não no Trecho-substituível, e se está presente e possui parâmetro efetivo.

Quando feita uma chamada a um parâmetro, o endereço da próxima instrução a ser executada deve ser obtido na célula referente a esse parâmetro. A posição da célula dentro da área é dada no operando da instrução CHAMA.PARAMETRO:

```
CHAMA.PARAMETRO(OPERANDO): chama parâmetro de célula OPERANDO
     TOPO:=TOPO + 3 ; & aloca área sem células
     PILHA(TOPO -3 ) := BASE.EXTERNA;
     PILHA(TOPO -2 ) := BASE.INTERNA;
     PILHA(TOPO -1 ) := PROXIMA.INSTRUCAO;
     PROXIMA.INSTRUCAO:=PILHA(BASE.INTERNA - 4 - OPERANDO);
     BASE.INTERNA :=PILHA(BASE.INTERNA-2 ); & RESTAURA B.I.
```

Para preencher células há a instrução ARMAZENA:

```
ARMAZENA(OPERANDO): copia ACUMULADOR para célula OPERANDO
     PILHA(BASE.EXTERNA-4-OPERANDO):=ACUMULADOR;
```

Para preencher o ACUMULADOR como um endereço efetivo há:

```
CARREGA.ENDERECO(OPERANDO): carrega endereço OPERANDO
     ACUMULADOR:=LE.ENDERECO(OPERANDO);
```

O programa dado como exemplo geraria às seguintes instruções:

```
                        DESVIA (ENDER.INSTR.TEXTO);
M:                      ...
                        CHAMA.PARAMETRO ($Q):
                        ...
                        RETORNA (NUMERO.DE.CELULAS);
P:                      ...
                        DESVIA(ENDER_INSTR.FIM.PARAMETRO);
ENDER.INSTR.PARAMETRO:  CHAMA.PARAMETRO ($Q);
                        TRANSMITE(+);
                        TRANSMITE(1);
                        RETORNA(0);
ENDER.INSTR.FIM.PARAMETRO:
                        CARREGA.ENDERECO(ENDER.INSTR.PARAMETRO);
                        ARMAZENA($Q);
                        CHAMA(M);
                        ...
                        RETORNA(0);
ENDER.INSTR.TEXTO:      ...
```

Para gerá-las, a análise de Cláusula-com-parâmetro na rotina TRECHO.SUBSTITUIVEL fica assim:

```
& NO.CLASULA.PARAMETRO:
   begin
     CONCATENA;
     GRAVA.INSTRUCAO(DESVIA,ENDER.INSTR.FIM.PARAMETRO:=
                          NUMERO.DE.ORDEM);
     GRAVA.ENDERECO(ENDER.INSTR.PARAMETRO:=NUMERO.DE.ORDEM);
     DEFINICAO; & GERA INSTRUCOES PARA O PARAMETRO EFETIVO
     GRAVA.INSTRUCAO(RETORNA,0);
     GRAVA.ENDERECO(ENDER.INSTR.FIM.PARAMETRO);
     GRAVA.INSTRUCAO(CARREGA.ENDERECO,ENDER.INSTR.PARAMETRO);
     GRAVA.INSTRUCAO(ARMAZENA,ARVORE^ATRIBUTO);
   end;
```

Os nós (Cláusula-parâmetro) e (Cláusula-sem-parâmetro) têm no campo ATRIBUTO a posição da célula dessas Cláusulas.

IV.6.2.5 Testes e Grupos

Grupos e Testes são entidades complementares. Se alguma coisa foi declarada como opcional, repetitiva ou tendo alternativas, na Definição certamente haverá um Teste que verifique a presença de alternativas ou controle as repetições.

Quando um Teste é executado, ele necessita conhecer como foi seguida a sintaxe do Grupo ao qual ele se refere: que alternativa foi usada e se o Grupo foi repetido.

Essas informações são geradas na análise do Trecho-substituível e são requisitadas a cada iteração do Teste. Para que o Teste obtenha essas informações ele executa uma instrução CHAMA.PARAMETRO. As instruções que armazenam essas informações nas células devidas são executadas até uma RETORNA(0).

Cada Cláusula tem uma célula na área de sua Declaração. Cada Grupo tem duas células adjacentes. A primeira contém o endereço do tal parâmetro. A segunda indica se existe repetição. Como elas são adjacentes, um nô (opcional), (obrigatório), (opcional-repetitivo) ou (obrigatório-repetitivo) contém no seu campo ATRIBUTO a posição da primeira célula. A posição da outra é calculável imediatamente.

Se um Grupo-opcional ou opcional-repetitivo for omitido no Trecho-substituível, na célula do endereço do parâmetro de informações será colocado o valor NULO (zero), que não pode ser um endereço de instrução. Como nos Testes pode haver Default, que é um trecho que irá para o texto-objeto caso o Grupo tenha sido omitido, são necessárias as instruções:

```
DESVIA.SE.PRESENTE(OPERANDO): desvia para endereço OPERANDO se
                              acumulador não nulo
     if not ACUMULADOR = NULO then
          PROXIMA.INSTRUCAO:=LE.ENDERECO (OPERANDO);

DESVIA.SE.NULO(OPERANDO): desvia para endereço se acumulador nulo
     if ACUMULADOR=NULO then
        PROXIMA.INSTRUCAO:=LE.ENDERECO (OPERANDO);
```

```
CARREGA.CELULA(OPERANDO): carrega célula OPERANDO no acumulador
     ACUMULADOR:=PILHA(BASE.INTERNA-4 -OPERANDO);
```

Exemplo IV.15:

```
- macro M [OPC];
    define
         ...
         {OPC define ...
         | define -30    }    & default
         ...
    endmacro
```

- essa declaração gerará:

```
   M:                              ...
                                   DESVIA(ENDER.INSTR.TESTA.OMISSAO);
                                   ...
   ENDER.INSTR.PREPARACAO:         ...
   ENDER.INSTR.TESTA.OMISSAO:      CARREGA.CELULA(CELULA.CLAUSULA);
                                   DESVIA.SE.PRESENTE(ENDER.INSTR.
                                                      PREPARACAO);
                                   TRANSMITE(-);
                                   TRANSMITE(30);
                                   ...
```

- um Trecho "M ;" geraria :

```
                                   ALOCA(NUMERO.DE.CELULAS);
                                   CARREGA.IMEDIATO(NULO);
                                   ARMAZENA(ARVORE^ATRIBUTO);
                                   CHAMA (M);
```

- um Trecho "M OPC ;" geraria:

```
                    ALOCA(NUMERO.DE.CELULAS);
                    CARREGA.ENDERECO(ENDER.INSTR.
                                PRIMEIRA.REPETICAO);
                    ARMAZENA(ARVORE^ATRIBUTO + 1);
                    CARREGA.IMEDIATO (PRESENTE);
                    DESVIA(ENDER.FIM.REPETICOES);
ENDER.INSTR.PRIMEIRA.REPETICAO:
                    ...
                    RETORNA (0);
                    ...
ENDER.INSTR.FIM.REPETICOES:
                    ARMAZENA (ARVORE^ATRIBUTO);
                    CHAMA (M);
```

As instruções que armazenam as informações nas células devem, caso o Grupo tenha alternativas, marcar NULO nas células das Alternativas que não forem a tomada, preencher a célula da Alternativa tomada ou indicar que nenhuma foi tomada. Isto para que os Testes-de-alternativas funcionem.

CAPÍTULO V

RESUMO DA LINGUAGEM LPSE

A linguagem LPSE é o mecanismo de extensão da linguagem de programação LPS. Nada mais é do que a adaptação da linguagem extensora E à linguagem LPS.

V.1 sequência-de-símbolos-léxicos

sequência-de-símbolos-léxicos ::= símbolo-léxicos+;

Um programa "E" é uma sequência de símbolos léxicos, embora nem toda sequência de símbolos léxicos seja um programa "E".

V.1.1 símbolo-léxico

símbolo-léxico ::= palavra | símbolo-reservado |
identificador-de-parâmetro;

Os símbolos-léxicos são palavras, símbolos-reservados e identificadores-de-parâmetros.

V.1.2 palavra

palavra ::= cadeia-de-caracteres | identificador |
número-sem-sinal | símbolo-especial

Uma palavra é um "token" da linguagem LPS. Entretanto, alguns identificadores não são considerados como palavra e sim com símbolo-reservado.

V.1.3 identificador-de-parâmetro

identificador-de-parâmetro ::= "$"identificador

Um identificador-de-parâmetro é um identificador precedido pelo caráter "$".

V.1.4 símbolo-reservado

símbolo-reservado ::=
"macro" | "define" | "endmacro" | "opend"
| "stend" | "original" | "..." | "{" | "}"
| "[" | "]" | "|" | "||" | "$"

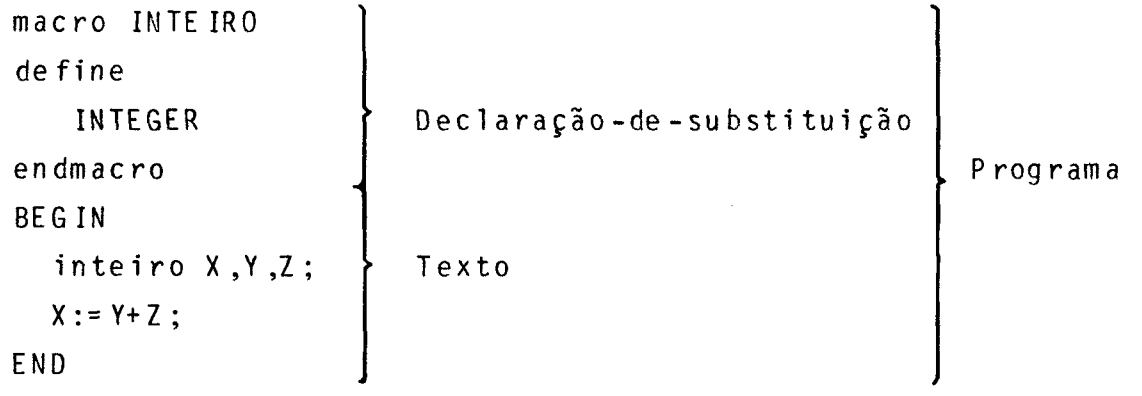
Os símbolos-reservados têm significado especial na linguagem, não podendo serem usados como palavras não reservadas. Servem basicamente, para montar declarações. Símbolos-reservados que difiram apenas no uso de letras maiúsculas ou minúsculas correspondentes são considerados os mesmos.

V.2 Programa LPSE

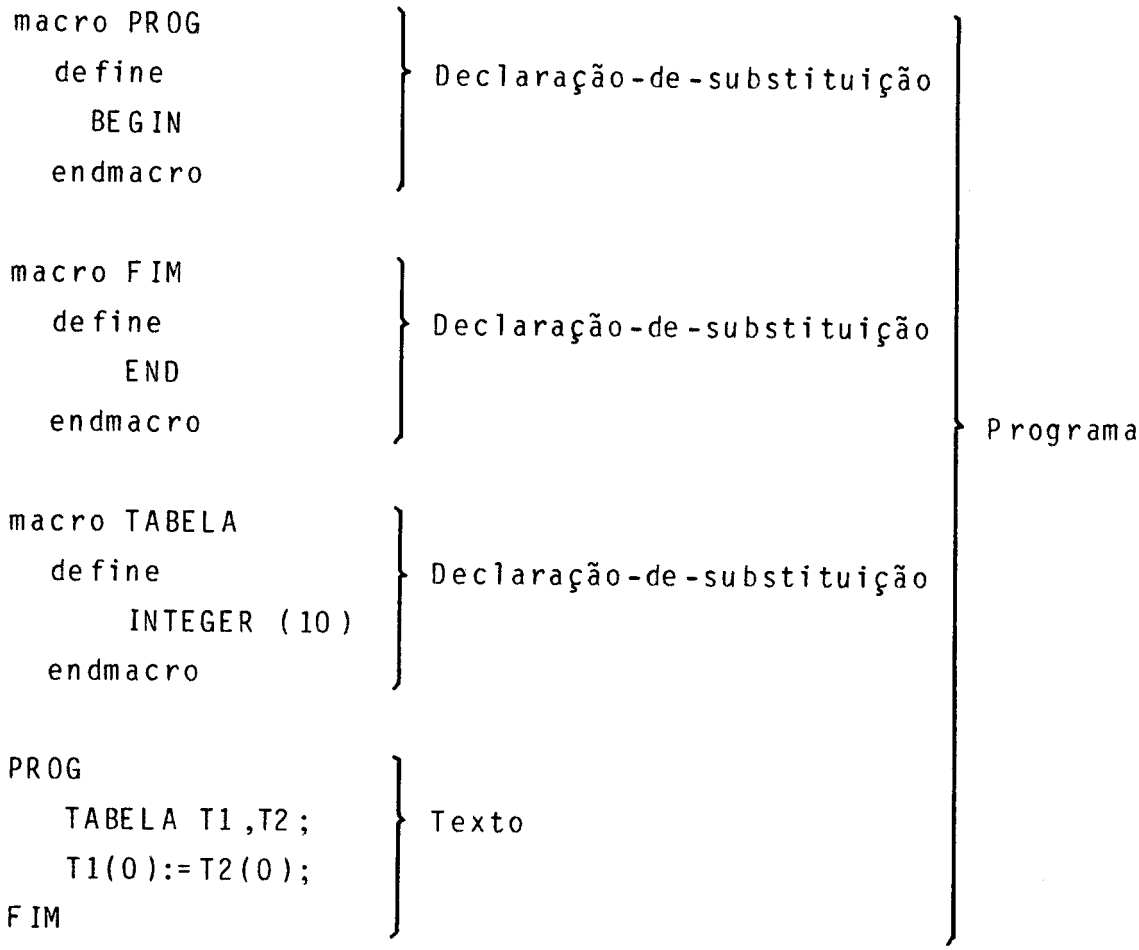
Programa ::= Declaração-de-substituição* Texto

Programa é um Texto precedido por Declaração-de-substituição, que indicam as substituições a serem feitas nesse Texto.

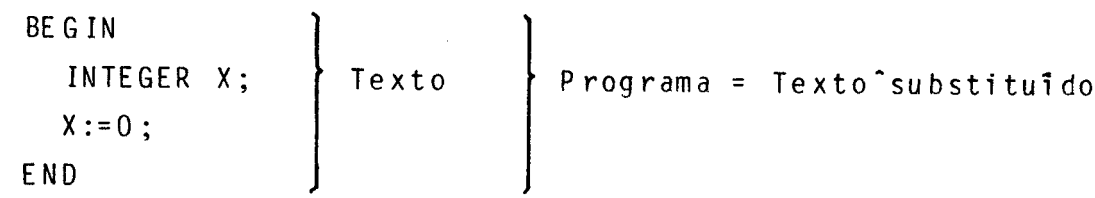
Exemplo V.1:



Exemplo V.2:



Exemplo V.3:



V.2.1 Declaração-de-substituição

Declaração-de-substituição ::=
"MACRO" Padrão-inicial "DEFINE" Padrão-final "ENDMACRO"

Uma Declaração-de-substituição inicia com o símbolo-reservado "MACRO" e termina com "ENDMACRO". É composta por um Padrão-inicial e um Padrão-final separados pelo símbolo-reservado "DEFINE".

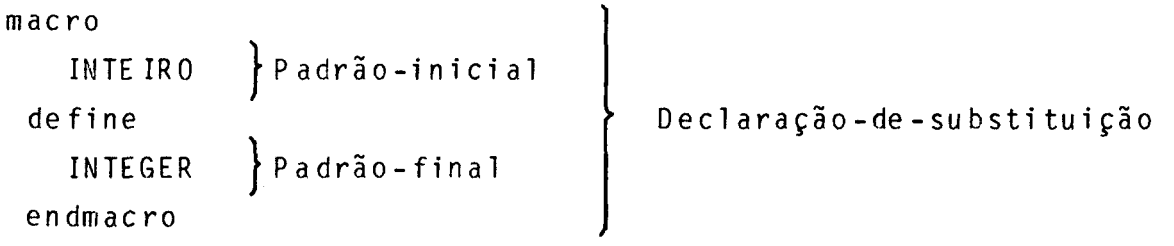
Ela acrescenta uma regra gramatical à linguagem "E". Essa regra vale à partir do "DEFINE" da própria Declaração-de-Substituição e permanece válida até o fim do Programa.

O Padrão-inicial é uma fórmula que fornece a sintaxe da regra acrescentada.

O Padrão-final indica como será feita a substituição dos trechos de texto que se encaixarem com a sintaxe do Padrão-inicial.

Sempre que essa sintaxe for reconhecida num trecho de texto será feita a substituição desse trecho conforme indicado no Padrão-final.

Exemplo V.4:



Exemplo V.5:

```
macro
   TROQUE X COM Y
 define
   BEGIN
     INTEGER Z;
     Z:=X;
     X:=Y;
     Y:=Z;
   END
 endmacro
```

Padrão-inicial

Padrão-final

Declaração-de-Substituição

V.2.2 Texto

Texto ::= (Trecho-substituível | Elemento-insubstituível)*

Um Texto é uma sequência, às vezes vazia, de Trechos-substituíveis e Elementos-insubstituíveis.

As regras gramaticais para Trecho-substituível são geradas pelas Declarações-de-substituição.

Exemplo V.6:

```
macro
   VETOR DE INTEIRO
 define
   INTEGER (*)
 end macro
BEGIN
   VETOR DE INTEIRO X = 10:0;
   VETOR DE INTEIRO Y = 5:0;
   X(3) := Y(1)
END
```

Texto

Exemplo V.7:

```
macro
    INTEIRO
 define
    INTEGER
 endmacro
macro
    .
 define
    ;
 endmacro
BEGIN              ┐
   INTEIRO X.      │
   INTEIRO Y.      } Texto
   X:=Y            │
END                ┘
```

V.3 Padrão-inicial

Padrão-inicial ::= Sintaxe Terminador?

O Padrão-inicial é uma fórmula que fornece a sintaxe da regra acrescentada.

O padrão-inicial de uma Declaração-de-substituição é composto de uma sintaxe seguida opcionalmente por um Terminador.

Entretanto esta Sintaxe tem algumas restrições:

1) Deve iniciar com uma Cláusula, cujo nome será o nome da substituição declarada. Não pode-se portanto, iniciar o Padrão-final com um Grupo.

2) Se o Padrão-inicial não possui um Terminador, a sua Sintaxe deve terminar com uma cláusula-sem-parâmetro, não podendo terminar nem com um Grupo, nem com uma Cláusula-com-parâmetro.

V.3.1 Cláusula-sem-parâmetro

Cláusula-sem-parâmetro ::= palavra+

Uma cláusula-sem-parâmetro é uma sequência de palavras num Padrão-inicial e indica que essas palavras aparecem em sequência na sintaxe atribuída a esse Padrão-inicial.

Toda Cláusula-sem-parâmetro tem um nome, que é a primeira palavra dessa cláusula.

Exemplo V.8:

```
macro
  SOME X COM Y      } Cláusula-sem-parâmetro } Padrão-inicial
 define
    X:=X+Y
 endmacro
BEGIN                                        }
  INTEGER X = 1, Y = 10;                     }
  SOME X COM Y;                              } Texto
  Y:=X                                       }
END                                          }
```

V.3.2 Indicador-de-parâmetro

Indicador-de-parâmetro::=identificador-de-parâmetro | "$"

Um Indicador-de-parâmetro ou é um identificador-de-parâmetro, ou é o símbolo-reservado "$".

V.3.3 Cláusula-com-parâmetro

Cláusula-com-parâmetro ::= palavra+ Indicador-de-parâmetro

Uma Cláusula-com-parâmetro difere de uma sem parâmetro por ter um Indicador-de-parâmetro no final. Na posição correspondente na sintaxe gerada pelo Padrão-inicial será criado um parâmetro-formal. Graças às Cláusulas-com-parâmetro temos substituições parametrizadas.

Do mesmo modo que a Cláusula-sem-parâmetro, a primeira palavra da Cláusula-com-parâmetro é o seu próprio nome. Entretanto, uma Cláusula-com-parâmetro tem um outro atributo: o nome-do-parâmetro.

Quando o Indicador-de-parâmetro for um identificador-de-parâmetro, o nome-do-parâmetro virá desse identificador-de-parâmetro; se for um "$", o nome-do-parâmetro será igual ao nome da cláusula.

Portanto, "$" é usado como Indicador-de-parâmetro, herdando o nome da cláusula para ser o seu nome-do-parâmetro.

Exemplo V.9:

```
macro DOBRE $P;
  define
      $P := $P * 2;
  endmacro
BEGIN
  INTEGER I,J;
  DOBRE I;
  DOBRE J;
END
```

Exemplo V.10:

```
macro SOME $A COM $B;
  define
       $A := $A + $B;
  endmacro
BEGIN
  INTEGER I,J;
  SOME I COM J;
  SOME J COM I*3;
END
```

Exemplo V.11:

```
macro QUAD ($)
  define
     ($QUAD) * ($QUAD)
  endmacro
BEGIN
  INTEGER I,J;
  I:= 2 + QUAD (J+1);
  J:= QUAD (0)
END
```

V.3.4 Sintaxe

```
Sintaxe ::= (Cláusula | Grupo)+

Cláusula ::= Cláusula-sem-parâmetro |
             Cláusula-com-parâmetro

Grupo ::= Grupo-obrigatório | Grupo-opcional

Grupo-obrigatório ::= Grupo-obrigatório-não-repetitivo
                    | Grupo-obrigatório-repetitivo
```

Exemplo V.12:

```
macro FAZ { [COMPRIME] ||[UMA | DUPLA] FACE || DISCO = $};
  define... endmacro
. . .
  FAZ DISCO = 3 UMA FACE;
  FAZ DUPLA FACE COMPRIME DISCO = 5;
. . .
```

Uma Sintaxe é composta por Grupos e/ou Cláusulas.

V.3.5 Conjunto-de-sintaxes

Conjunto-de-sintaxes ::= Sintaxe ("||" Sintaxe)*

Um Conjunto-de-sintaxes é composto por uma Sintaxe ou por várias separadas pelo símbolo " ". Este símbolo indica que a ordem de aparecimento das sintaxes especificadas não importa no Trecho-substituivel correspondente.

Exemplo V.13:

```
macro COBRE { CUSTOS = $ || LUCRO = $ || IMP = $ } ;
  define
     . . .
  endmacro
. . .
COBRE IMP = 10 CUSTO = 50000 LUCRO = 12;
COBRE LUCRO = 33 IMP = 5 CUSTO = 100000;
. . .
```

Exemplo V.14:

```
macro FAZ [TUDO | ESQ || DIR] ...;
  define ... endmacro
. . .
  FAZ TUDO ESQ DIR DIR ESQ DIR ESQ TUDO ESQ DIR;
  FAZ;
. . .
```

V.3.6 Alternativas

Alternativas::=
Conjunto-de-sintaxes ("|" Conjunto-de-sintaxes)*

Como o nome diz, Alternativas fornecem sintaxes alternativas para os Trechos-substituíveis. Elas só podem aparecer dentro de um grupo e podem ser composta de uma única alternativa (conjunto-de-sintaxes).

Exemplo V.15:

```
macro FOR $ {UPTO $|DOWNTO $}  DO $;
  define
    . . .
  endmacro
. . .
  FOR I UPTO 10 DO X(I):=0;
. . .
  FOR J DOWNTO I DO Z(J):=X(J)-J;
. . .
```

V.3.7 Grupo-obrigatório-não-repetitivo

Grupo-obrigatório-não-repetitivo ::= "{ " Alternativas "}"

Um Grupo-obrigatório-não-repetitivo indica que do Trecho-substituível correspondente deve constar uma, e apenas uma, das Alternativas enfeixadas nesse Grupo.

Exemplo V.16:

```
macro FOR $ {UPTO $ |DOWNTO $}  DO $;
  define
    . . .
  endmacro
. . .
 FOR I UPTO 10 DO X(I):=0;
. . .
 FOR J DOWNTO I DO Z(J):=X(J)-J;
. . .
```

V.3.8 Grupo-obrigatório-repetitivo

Grupo-obrigatório-repetitivo ::= "{ " Alternativas "}" ..."

Um Grupo-obrigatório-repetitivo indica que do Trecho-substituível correspondente deve constar pelo menos uma das Alternativas enfeixadas nesse Grupo, podendo, inclusive, haver repetição da mesma alternativa.

V.3.9 Grupo-opcional-não-repetitivo

Grupo-opcional-não-repetitivo ::= "[" Alternativas "]"

Um Grupo-opcional aparece num Padrão-inicial para indicar trechos opcionais na sintaxe desse Padrão-inicial. Um Trecho-substituível referente a esse Padrão-inicial pode ter ou não esses trechos opcionais.

Exemplo V.17:

```
macro SE $ ENTAO $ [SENAO $];
   define          ──────v──────
      . . .        Grupo-opcional-não-repetitivo
   endmacro
BEGIN
   INTEGER I,J;
   SE I J ENTAO I:=J;                  } Trecho-substituível
   SE I J ENTAO J:=0  SENAO J:=1;      } Trecho-substituível
END
```

V.3.10 Grupo-opcional-repetitivo

Grupo-opcional-repetitivo ::="[" Alternativas "]" "..."

Enquanto um Grupo-opcional-nao-repetitivo indica que um trecho opcional pode aparecer nenhuma ou uma vez num Trecho-substituível, um Grupo-opcional-repetitivo indica que o trecho pode aparecer nenhuma, uma ou mais vezes.

Exemplo V.18:

```
macro CASO $ [QUANDO $ => $C;] ... FIMCASO;
   define
      . . .
   endmacro
. . .
CASO I                          ┐
  QUANDO 0    =   X:=Y;         │
  QUANDO 3    =   X:=Z;         ├ Trecho-substituível
  QUANDO N+7  =   Z:=Y;         │
FIMCASO;                        ┘
. . .
CASO A+B*Z FIMCASO;             } Trecho-substituível
. . .
CASO I + J                      ┐
  QUANDO 1 =   X:=X+1;          ├ Trecho-substituível
FIMCASO;                        ┘
. . .
```

V.3.11 Terminador

```
Terminador ::= "STEND" | "OPEND"
```

Um Terminador é usado no Padrão-inicial para indicar que a sintaxe do Trecho-substituível correspondente termina com qualquer palavra do conjunto indicado e essa palavra não faz parte do Trecho-substituível. Esses dois conjuntos são:

```
STEND:   ; END  ELSE  UNTIL

OPEND:   ) , THEN DO TO DOWNTO STEP OF;  END
                                  ELSE UNTIL
```

Exemplo V.19:

```
macro IF $ THEN $ [ELSE $];
  define ... endmacro
. . .
  IF A   B THEN
      IF ERRADO THEN
         GOTO FIM;
  GOTO RPT;
. . .
```

V.4 Padrão-final

```
Padrão-final ::= ( Elemento-insubstituível |
                   Trecho-substituível |
                   Referência-a-parâmetro | Teste )*
```

Um Padrão-final é uma fórmula que especifica como uma determinada substituição deve ser realizada.

É composto por uma sequência, possivelmente vazia, de Elementos-insubstituíveis, Trechos-substituível, Referência-a-parâmetro e Testes.

V.4.1 Elemento-insubstituível

Elemento-insubstituível ::= "ORIGINAL"? palavra

Um Elemento-insubstituível é uma palavra que, ou não faz parte da sintax de um Trecho-substituível, ou vem precedida pelo símbolo-reservado "ORIGINAL".

V.4.2 Trecho-substituível

As regras gramaticais para Trecho-substituível são geradas pelas Declarações-de-substituição.

Quando a sintaxe gerada pelo Padrão-inicial de uma Declaração-de-substituição é encaixada em um Texto ou Padrão-final fica reconhecido um Trecho-substituível.

A substituição desse trecho é realizada atravéz do Padrão-final dessa Declaração-de-substituição.

Exemplo V.20:

```
macro
   VETOR DE INTEIRO
 define
   INTEGER (*)
 end macro
BEGIN
   VETOR DE INTEIRO X = 10:0;
   VETOR DE INTEIRO Y = 5:0;
   X(3) := Y(1)
END
```

Texto

TEXTO	
Elemento-insubstituível	BEGIN
Trecho-substituível	VETOR DE INTEIRO
Elemento-insubstituível	X
Elemento-insubstituível	=
Elemento-insubstituível	10
Elemento-insubstituível	:
Elemento-insubstituível	0
Elemento-insubstituível	;
Trecho-substituível	VETOR DE INTEIRO
Elemento-insubstituível	Y
Elemento-insubstituível	=
Elemento-insubstituível	5
Elemento-insubstituível	:
Elemento-insubstituível	0
Elemento-insubstituível	;
Elemento-insubstituível	X
Elemento-insubstituível	(
Elemento-insubstituível	3
Elemento-insubstituível	)
Elemento-insubstituível	:=
Elemento-insubstituível	Y
Elemento-insubstituível	(
Elemento-insubstituível	1
Elemento-insubstituível	)
Elemento-insubstituível	END

Exemplo V.21:

```
macro
   INTEIRO
 define
   INTEGER
 endmacro
macro
   .
 define
   ;
 endmacro
BEGIN          ┐
   INTEIRO X.  │
   INTEIRO Y.  } Texto
   X:=Y        │
END            ┘
```

```
Elemento-insubstituível:    BEGIN
Trecho-substituível:        INTEIRO
Elemento-insubstituível:    X
Trecho-substituível:        .
Trecho-substituível:        INTEIRO
Elemento-insubstituível:    Y
Trecho-substituível:        .
Elemento-insubstituível:    X
Elemento-insubstituível:    :=
Elemento-insubstituível:    Y
Elemento-insubstituível:    END
```

V.4.3 Referência-a-parâmetro

Cada Cláusula-com-parâmetro acrescenta uma regra gramatical à linguagem "E" criando uma nova sintaxe para Referência-a-parâmetro. Essa regra é válida apenas no Padrão-final da própria Declaração-de-substituição que contém essa Cláusula-com-parâmetro.

A sintaxe contida nessa regra sempre refere-se a um identificador-de-parâmetro de mesmo nome-do-parâmetro que o da Cláusula-com-parâmetro.

Em cada substituição feita, um Trecho-substituível é trocado por um Padrão-final. As Referências-a-parâmetro que apareçam nesse Padrão-final são também substituídas pelos Parâmetros-efetivos correspondentes que apareçam nesse Trecho-substituível.

Note-se que as Referências-a-parâmetro podem aparecer em qualquer número e ordem, sendo um Parâmetro-efetivo repetido tantas vezes, numa substituição, quantas forem as repetições da correspondente Referência-a-parâmetro no Padrão-final.

Exemplo V.22:

```
macro DOBRE $;
   define
      $DOBRE := $DOBRE * 2;      Padrão-final
   endmacro


macro SOME ($,$B)
   define
      $SOMA := $SOMA + $B        Padrão-final
   endmacro
```

V.4.4 Nome-de-cláusula

Cada Cláusula acrescenta uma regra gramatical à linguagem LPSE criando uma nova sintaxe para Nome-de-cláusula. Essa regra é válida apenas no Padrão-final da própria Declaração-de-substituição que contém essa Cláusula.

A sintaxe contida nessa regra refere-se sempre a uma palavra que é o nome da Cláusula.

Um Nome-de-cláusula é usado em um Teste-de-alternativa para testar a ocorrência de uma determinada Cláusula.

V.4.5 Indicador-de-cláusula

Indicador-de-cláusula ::= Nome-de-cláusula | Referência-a-parâmetro

Um Indicador-de-cláusula é usado em um Teste-de-alternativa para testar a ocorrência de uma determinada Cláusula. Se esta for uma Cláusula-com-parâmetro, o Indicador-de-cláusula pode ser uma Referência-ao-parâmetro dessa cláusula. Caso contrário, deve ser o Nome-de-cláusula.

V.4.6 Teste-de-alternativa

```
Teste-de-alternativa ::= Indicador-de-cláusula
                         "DEFINE" Padrão-final
                         | Teste
```

Um Teste-de-alternativa é usado em um Padrão-final para fazer substituições condicionadas a ocorrência no Trecho-substituível de uma Alternativa indicada nesse Teste-de-alternativa.

Exemplo V.23:

```
macro FOR $ { UPTO $ | DOWNTO $}  DO $
  define
     WHILE $FOR { UPTO define <=$UPTO
                |DOWNTO define >= $DOWNTO }
     DO BEGIN
          $DO;
          $FOR := $FOR {DOWNTO define - |UPTO define + }  1
        END;
  endmacro
. . .


macro INCR {UM | DOIS};
  define
     X:= X + 1  DOIS define + 1 ;
  endmacro
. . .
```

V.4.7 Default

Default ::= "|" "DEFINE" Definição

O Default é usado em um Teste num Padrão-final para fazer substituições condicionadas a não ocorrência no Trecho-substituível de nenhuma das Alternativas do Grupo a que refere-se esse Teste.

Exemplo V.24:

```
macro PARA $  [INCR $ | DECR $] ATE $ FAZ $;
  define
     . . .
     $PARA := PARA { INCR define + $INCR
                   |DECR define - $DECR
                   |define + 1          }
     . . .
  endmacro
. . .
```

V.4.8 Teste

```
Teste::="{ "Teste-de-alternativa("|"Teste-de-alternativa)*
  Default? "}"
```

Nesse exemplo vemos que o Teste define um pedaço do Padrão-final que é repetido para cada ocorrência do trecho opcional no Trecho-substituível. Se o Grupo-opcional-repetitivo possui Indicadores-de-parâmetro, cada Referência-a-parâmetro que corresponda a algum deles, será substituída em cada repetição pelo respectivo parâmetro-efetivo da respectiva repetição do trecho opcional no Trecho-substituível.

Observe-se que a definição do funcionamento de um Teste independe se ele refere-se a um grupo repetitivo ou não, pois, para este, só haverá uma ocorrência do pedaço correspondente ao grupo. Portanto, a definição anterior do teste para Grupos não-repetitivos é um caso particular desta nova.

Os Testes correspondentes a Grupo-obrigatório nao devem ter "default" pois estes nunca serão escolhidos pelo processador da linguagem "E".

CAPÍTULO VI

CONCLUSÕES

A linguagem extensora "E" foi proposta como um mecanismo de alteração de linguagens. Um processador dessa linguagem extensora adaptado a um compilador é uma ferramenta para estender a linguagem processada por esse compilador. Um processador "E" foi construído para o compilador LPS, conforme descrito no capítulo IV, sendo chamado de compilador LPSE.

O compilador LPSE dá um poderoso auxílio para o programador na linguagem LPS, pois este passa a poder usar macros. Com efeito, as Declarações-de-substituição da linguagem "E" podem ser usadas como macros ultra-poderosas. Outra aplicação óbvia do LPSE é a construção de bibliotecas de extensões da linguagem especializadas para determinados tipos de aplicações.

A linguagem e o processador "E" assimilariam, sem problemas, novas facilidades que aumentassem ainda mais seu poder de estender linguagens.

As mais úteis seriam:

- variáveis em tempo de substituição. Permitem que valores possam ser calculados durante a substituição de texto, testados e gerados na saída. Estas variáveis poderiam ser globais às declarações ou internas. Esta facilidade de implementação simples, multiplicaria o poder da linguagem "E".

- tipificações dos parâmetros e declarações. Equivale a transformar os parâmetros em não-terminais, e as declarações em produções gramaticais com o lado direito regular.

- fim das restrições sintáticas dos Padrões-iniciais. Permitiria declaração com mais de um nome e mais de um símbolo terminador.

- especificações dos "tokens" de entrada e dos "tokens" de saída. Trasnformaria o processador "E" num mecanismo de traduções de uma linguagem qualquer em outra, com elementos léxicos distintos. Libertaria o processador da adaptação ao compilador.

Quanto à linguagem LPSE, sugerimos a criação de bibliotecas nessa linguagem de modo a estender a linguagem LPS. As bibliotecas em vista proveriam a LPS de comandos de entrada/saída, formatação de telas, geração de relatórios, comandos de "pattern-matching", comandos para aplicações em tempo-real, "multi-tasking" etc.

Bibliografia

1. Aho, A.V., Ullman, J.D. - Principles of Compiler Design, Addison-Wesley, 1976.

2. Aho, A.V., Ullman, J.D. - The Theory of Parsing, Translation and Compiling, Volume 1, Prentice-Hall, 1972.

3. Backus, J.W et. al. - Revised report on the algorithmic language Algol 60. Num. Math., 4:420-53, 1963.

4. Bauer, F.L., Eickel, J. - Compiler Construction: An Advanced Course, Springer-Verlag, 1976.

5. Cole, A.J. - Macro Processors, Cambridge University Press, 1976.

6. Irons, E.T. - Experience with an Extensible Language. Comm. A.C.M., 13(1):31-40, 1970.

7. Leavenworth, B.M. - Syntax macros and extended translation. Comm. A.C.M., 9(11):790-3, 1966.

8. Lewis, P.M., Stearns, R.E. - Syntax-directed Transduction. JACM, 15(3):465:24, 1968.

9. Manual de Referência da Linguagem LPS - COBRA-500, COBRA S/A, 1980.

10. McIlroy, M.D. - Macro instruction extensions of compiler languages. Comm. A.C.M., 3(4):214-20, 1960.

11. Prasad, V.R. - Variable Number of Parameters in Types Languages. Soft. Pract. and Exp., 10:507-517, 1980.

12. Reference Manual for the Ada Programming Language, U.S.D.o.Defense, 1980.

13. Sassa, M. - A Pattern Matching Macro Processor. Soft. Pract. and Exp., 9:439-456, 1979.

14. Standish, T.A. - Extensibility in Programming language design. SIGPLAN Notices, 10(7):18-21, 1975.

```
ARQUIVO FONTE: SMM      [DP01]   VOLUME=00001    VERSAO=        CODIGO=LPSPPF
EQUIPAMENTO:    C500
PARAMETROS:     F:SMM:NS:L1
OPCOES INICIAIS: LE  NLC LA  NS  D0  L1  D2  D3  D4  D5  D6  D7

0000-D 00 ?LPS,"SEGMM",S,V="X01",D
0000-D 00 &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
0000-D 00 &                                                                      &
0000-D 00 &                            S  E  G  M  M                             &
0000-D 00 &                                                                      &
0000-D 00 &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
0000-D 00 &
0000-D 00 & PROJETO:
0000-D 00 &     LPS
0000-D 00 & FUNCAO:
0000-D 00 &     RECONHECIMENTO DE MACROS E GERACAO DO INTERMEDIARIO PRA EXPANSAO.
0000-D 00 & MODELO:
0000-D 00 &     UNICO
0000-D 00 & LINGUAGEM:
0000-D 00 &     LPS
0000-D 00 & RELOCAVEL:
0000-D 00 &     SMM
```

```
0020 0000-D 00 BEGIN
0021 0000-P 01
0022 0000-P 01 & CONSTANTES:
0023 0000-P 01 & ===========
0024 0000-P 01
0025 0000-P 01 &     PSEUDO-CODIGOS:
0026 0000-P 01 &     ---------------
0027 0000-P 01
0028 0000-P 01 CONSTANT DESVIA                          = 000;
0029 0000-D 01 CONSTANT FINALIZA                        = 001;
0030 0000-D 01 CONSTANT RETORNA                         = 002;
0031 0000-D 01 CONSTANT TRANSMITE                       = 003;
0032 0000-D 01 CONSTANT ALOCA                           = 004;
0033 0000-D 01 CONSTANT CHAMA                           = 005;
0034 0000-D 01 CONSTANT CHAMA.PARAMETRO                 = 006;
0035 0000-D 01 CONSTANT CARREGA.DADO                    = 007;
0036 0000-D 01 CONSTANT DESVIA.SE.PRESENTE              = 008;
0037 0000-D 01 CONSTANT DESVIA.SE.NULO                  = 009;
0038 0000-D 01 CONSTANT CARREGA.ENDERECO                = 00A;
0039 0000-D 01 CONSTANT ARMAZENA                        = 00B;
0040 0000-D 01 CONSTANT CARREGA.IMEDIATO                = 00C;
0041 0000-D 01 CONSTANT MARCA.DADO                      = 00D;
0042 0000-D 01
0043 0000-D 01 &     TIPOS-DE-TOKENS:
0044 0000-D 01 &     ----------------
0045 0000-D 01
0046 0000-D 01 CONSTANT TMACRO                          = 800;      & MACRO
0047 0000-D 01 CONSTANT TDEFINE                         = 801;      & DEFINE
0048 0000-D 01 CONSTANT TENDMACRO                       = 802;      & ENDMACRO
0049 0000-D 01 CONSTANT TOPEND                          = 803;      & OPEND
0050 0000-D 01 CONSTANT TSTEND                          = 804;      & STEND
0051 0000-D 01 CONSTANT TORIG                           = 805;      & ORIGINAL
0052 0000-D 01 CONSTANT T...                            = 806;      & ...
0053 0000-D 01 CONSTANT ABRE.CHAVES                     = 807;      & <<
0054 0000-D 01 CONSTANT FECHA.CHAVES                    = 808;      & >>
0055 0000-D 01 CONSTANT ABRE.COLCHETES                  = 809;      & [
0056 0000-D 01 CONSTANT FECHA.COLCHETES                 = 80A;      & ]
0057 0000-D 01 CONSTANT BARRA                           = 80B;      & \
0058 0000-D 01 CONSTANT BARRA.BARRA                     = 80C;      & \\
0059 0000-D 01 CONSTANT IDENTIFICADOR.DE.PARAMETRO      = 80D;      & C<IDENTIFICADOR>
0060 0000-D 01 CONSTANT DOLAR                           = 80E;      & C
0061 0000-D 01 CONSTANT PALAVRA                         = 80F;      & <PALAVRA>
0062 0000-D 01 CONSTANT FIM.DE.FONTE                    = 810;      & <FIM.DE.ARQUIVO>
```

```
0064 0000-0 01 %     TIPOS DE ENTRADA DA TABELA DE VARIAVEIS:
0065 0000-0 01 %     ----------------------------------------
0066 0000-0 01
0067 0000-0 01 CONSTANT DE.PARAMETRO                    =                    0;
0068 0000-0 01 CONSTANT DE.CLAUSULA                     =                    1;
0069 0000-0 01 CONSTANT DE.GRUPO                        =                    2;
0070 0000-0 01
0071 0000-0 01 %     NOS DE ARVORES:
0072 0000-0 01 %     --------------
0073 0000-0 01
0074 0000-0 01 CONSTANT NO.PALAVRA                      =                  800;
0075 0000-0 01 CONSTANT NO.STEND                        =                  801;
0076 0000-0 01 CONSTANT NO.OPEND                        =                  802;
0077 0000-0 01 CONSTANT NO.OPCIONAL                     =                  803;
0078 0000-0 01 CONSTANT NO.OBRIGATORIO                  =                  804;
0079 0000-0 01 CONSTANT NO.OPCIONAL.REPETITIVO          =                  805;
0080 0000-0 01 CONSTANT NO.OBRIGATORIO.REPETITIVO       =                  806;
0081 0000-0 01 CONSTANT NO.BARRA.BARRA                  =                  807;
0082 0000-0 01 CONSTANT NO.CONCATENACAO                 =                  808;
0083 0000-0 01 CONSTANT NO.CLAUSULA.SEM.PARAMETRO       =                  809;
0084 0000-0 01 CONSTANT NO.CLAUSULA.PARAMETRO           =                  80A;
0085 0000-0 01
0086 0000-0 01 %     ERROS:
0087 0000-0 01 %     -----
0088 0000-0 01
0089 0000-0 01 CONSTANT FALTEIMFONT                     =                   64;
0090 0000-0 01 CONSTANT FALTSIMS                        =                  141;
0091 0000-0 01 CONSTANT FALTNOMMCR                      =                  169;
0092 0000-0 01 CONSTANT ORIGINV                         =                  170;
0093 0000-0 01 CONSTANT PARMMCRINV                      =                  171;
0094 0000-0.01 CONSTANT ITEINV                          =                  172;
0095 0000-0 01 CONSTANT CHAMMCRINV                      =                  173;
0096 0000-0 01
0097 0000-0 01 %     DIVERSAS:
0098 0000-0 01 %     --------
0099 0000-0 01
0100 0000-0 01 CONSTANT FALSO                           =                    0;
0101 0000-0 01 CONSTANT VERO                            =                    1;
0102 0000-0 01 CONSTANT NULO                            =                    0;
0103 0000-0 01 CONSTANT PRESENTE                        =                    1;
0104 0000-0 01 CONSTANT NUMERO.DE.STEND                 =                    4;
0105 0000-0 01 CONSTANT NUMERO.DE.OPEND                 =                   12;
0106 0000-0 01 CONSTANT ULRELAT2                        =                  211;
0107 0000-0 01 CONSTANT TAMANHO.DO.VETOR.STEND    = 2*NUMERO.DE.STEND+1;
0108 0000-0 01 CONSTANT TAMANHO.DO.VETOR.OPEND    = 2*NUMERO.DE.OPEND+1;
```

```
0110 0000-0 01 & DADOS E ROTINAS EXTERNOS:
0111 0000-0 01 & ========================
0112 0000-0 01
0113 0000-0 01 & MODULO "LPS":
0114 0000-0 01 & ------------
0115 0000-0 01
0116 0000-0 01 EXTERNAL BYTE(*) A.COMUM;
0117 0000-0 01      BYTE(*)     A.PRINC                    POS A.COMUM + 00000;
0118 0000-0 01           BYTE   RASTRO                     POS A.COMUM + 00004;
0119 0000-0 01      BYTE(*)     A.LIST                     POS A.COMUM + 0014F;
0120 0000-0 01           BYTE   ULIST                      POS A.COMUM + 0014F;
0121 0000-0 01 EXTERNAL BYTE(*) A.LPS;
0122 0000-0 01      WORD        LINHAS.COMPILADAS          POS A.LPS   + 00021;
0123 0000-0 01      BYTE        SEGUNDO.PASSO              POS A.LPS   + 00023;
0124 0000-0 01      WORD        LINHA.INICIAL.DO.TEXTO     POS A.LPS   + 00024;
0125 0000-0 01      BYTE        ULRELATI                   POS A.LPS   + 00029;
0126 0000-0 01 PROCEDURE        ERRO                      (BYTE NUMERO);          EXTERNAL
0127 0000-0 01 PROCEDURE        LIST                      (WORD ENDERECO;
0128 0000-0 01                                             BYTE TAMANHO);         EXTERNAL
0129 0000-0 01 & MODULO "ANF":
0130 0000-0 01 & ------------
0131 0000-0 01
0132 0000-0 01 PROCEDURE        CAPSEGM                   (BYTE SEGM);            EXTERNAL
0133 0000-0 01
0134 0000-0 01 & MODULO "GINT":
0135 0000-0 01 & -------------
0136 0000-0 01
0137 0000-0 01 EXTERNAL BYTE    TIPO.DO.TOKEN;
0138 0000-0 01 EXTERNAL WORD    NUMERO.DO.TOKEN;
0139 0000-0 01 EXTERNAL WORD    NUMERO.DO.TOKEN.ANTERIOR;
0140 0000-0 01 PROCEDURE        INI.MCR;                                           EXTERNAL
0141 0000-0 01 PROCEDURE        GRAV.INTER                (BYTE COD;
0142 0000-0 01                                             WORD ATRIBUTO);        EXTERNAL
0143 0000-0 01 WORD PROCEDURE   NORD;                                              EXTERNAL
0144 0000-0 01 PROCEDURE        GRAV.DEF                  (WORD NUM.ORD);         EXTERNAL
0145 0000-0 01 PROCEDURE        LE.TOKEN;                                          EXTERNAL
0146 0000-0 01 WORD PROCEDURE   TOKEN.NUMERO              (WORD SIMBOLO.LEXICO);  EXTERNAL
0147 0000-0 01
0148 0000-0 01 & MODULO "TBVAR":
0149 0000-0 01 & --------------
0150 0000-0 01
0151 0000-0 01 EXTERNAL BYTE    CELULA.PESQUISADA;
0152 0000-0 01 EXTERNAL BYTE    CELULA.DO.GRUPO.DA.PESQUISADA;
0153 0000-0 01 PROCEDURE        ESVAZIA.TABELA.DE.CELULAS;                         EXTERNAL
0154 0000-0 01 PROCEDURE        DECORE.CELULA             (BYTE TIPO;
0155 0000-0 01                                             WORD ARVORE,
0156 0000-0 01                                                  CELULAS, GRUPO);  EXTERNAL
0157 0000-0 01 BYTE PROCEDURE   PESQUISA.CELULA           (BYTE TIPO;
0158 0000-0 01                                             WORD CARACTERISTICA);  EXTERNAL
```

```
0160 0000-D 01 & MODULO "TB*CR":
0161 0000-D 01 & --------------
0162 0000-D 01
0163 0000-D 01 EXTERNAL BYTE(*) DADOS.INFORMADOS;
0164 0000-D 01      WORD        ARVORE.INFORMADA                  POS DADOS.INFORMADOS + 2;
0165 0000-D 01      WORD        CODIGO.INFORMADO                  POS DADOS.INFORMADOS + 4;
0166 0000-D 01      BYTE        CELULAS.INFORMADAS                POS DADOS.INFORMADOS + 6;
0167 0000-D 01 PROCEDURE        DECORE.DECLARACAO                (WORD NUMERO.DO.TOKEN,
0168 0000-D 01                                                         ARVORE,  CODIGO;
0169 0000-D 01                                                   BYTE CELULAS         ); EXTERNAL;
0170 0000-D 01 BYTE PROCEDURE   INICIO.DE.TRECHO.SUBSTITUIVEL (WORD NUMERO.DO.TOKEN); EXTERNAL;
0171 0000-D 01
0172 0000-D 01 & MODULO "G*EM":
0173 0000-D 01 & -------------
0174 0000-D 01
0175 0000-D 01 BYTE(*)          NO                                REF *;
0176 0000-D 01      BYTE        TIPO.DO.NO                        POS NO + 0;
0177 0000-D 01      WORD        FILHO                             POS NO + 1;
0178 0000-D 01      WORD        IRMAO                             POS NO + 3;
0179 0000-D 01      WORD        ATRIBUTO                          POS NO + 5;
0180 0000-D 01      WORD        PROXIMO                           POS NO + 7;
0181 0000-D 01 PROCEDURE        SALVA                            (WORD EDADOS);          EXTERNAL;
0182 0000-D 01 PROCEDURE        RESTAURA                         (WORD EDADOS);          EXTERNAL;
0183 0000-D 01 WORD PROCEDURE   CRIA                             (BYTE NO);              EXTERNAL;
```

```
0185 0000-D 01 % DADOS E ROTINAS LOCAIS:
0186 0000-D 01 % ======================
0187 0000-D 01
0188 0000-D 01 BYTE (TAMANHO.DO.VETOR.STEND) CONJUNTO.STEND   = (NUMERO.DE.STEND,
0189 0001-D 01                                                   NUMERO.DE.STEND:0,
0190 0005-D 01                                                   NUMERO.DE.STEND:0);
0191 0009-D 01        WORD(*)                 ELEMENTO.STEND  POS CONJUNTO.STEND + 1;
0192 0009-D 01
0193 0009-D 01 BYTE (TAMANHO.DO.VETOR.OPEND) CONJUNTO.OPEND   = (NUMERO.DE.OPEND,
0194 000A-D 01                                                   NUMERO.DE.OPEND:0,
0195 0016-D 01                                                   NUMERO.DE.OPEND:0);
0196 0022-D 01        WORD(*)                 ELEMENTO.OPEND  POS CONJUNTO.OPEND + 1;
0197 0022-D 01
0198 0022-D 01
0199 0022-D 01
0200 0022-D 01
0201 0022-D 01
0202 0022-D 01
0203 0022-D 01
0204 0022-D 01 PROCEDURE DEIXA.RASTRO (BYTE N);
0205 0022-D 01    %       ------------
0206 0022-D 01    BEGIN
0207 0008-P 02 ?C,1
0208 0008-P 02       BYTE(*) MNE = (74,"<  >");
0209 0028-D 02       BYTE(*) TAB.MNE = ("PGMDCLSTXCLACLPGRUALTDEFTSTTCL P  S  0  ?  !  * ",
0210 0058-D 02                                  " +  /  .  C  C ");
0211 0067-D 02       IF NOT RASTRO RTR 6 THEN
0212 0012-P 02            RETURN;
0213 0014-P 02       MNE(1):="<";
0214 001A-P 02       MNE(4):=">";
0215 0020-P 02       MRT(.TAB.MNE+(N/2)*3,.MNE+2-N MOD 2,3);
0216 0052-P 02       LIST(.MNE,4);
0217 005E-P 02 ?C
0218 005E-P 02    END DEIXA.RASTRO;
0219 0067-D 01
0220 0067-D 01
0221 0067-D 01
0222 0067-D 01
0223 0067-D 01
0224 0067-D 01
0225 0067-D 01 PROCEDURE MONITORA (WORD NOME, VALOR);
0226 0067-D 01    %       --------
0227 0067-D 01    BEGIN
0228 006C-P 02 ?C,1
0229 006C-P 02       BYTE(*) L = (16:0);
0230 007A-D 02       MRT(NOME,.L+1,10);
0231 007C-P 02       BINHEX(VALOR,.L+12,4,0);
0232 0098-P 02       LIST(.L,15);
0233 00A4-P 02 ?C
0234 00A4-P 02    END MONITORA;
```

```
0236 0078-D 01 PROCEDURE DESENHA (WORD ARVORE);
0237 007B-D 01    &       -------
0238 0078-D 01    BEGIN
0239 00AE-P 02 ?C,1
0240 00AE-P 02       BYTE(*)   VARIAVEIS =   (2,2:0);
0241 0080-D 02            WORD ARV       POS VARIAVEIS + 1;
0242 0090-D 02       WORD I = 1;
0243 00B2-D 02
0244 00B2-D 02       BYTE(17) NO = "XXXX(X,XXXX,XXXX)";
0245 0093-D 02            BYTE(*) EARV  POS NO +  0;
0246 0093-D 02            BYTE    VARV  POS NO +  5;
0247 0093-D 02            BYTE(*) VATR  POS NO +  7;
0248 0093-D 02            BYTE(*) EPROX POS NO + 12;
0249 0093-D 02       BYTE(*) CAR = "PSO?!*+/.CC";
0250 009E-D 02       BYTE(*) LIN = (0,132:" ");
0251 0123-D 02       IF NOT PASTRO RTR 6 THEN
0252 00B3-P 02            RETURN;
0253 00BA-P 02       IF I >= 132-17 THEN
0254 00C2-P 02            RETURN;
0255 00C4-P 02       SALVA(AVARIAVEIS); ARV:=ARVORE;
0256 00D6-P 02       BINHEX(ARV,AEARV,4,0);
0257 00F6-P 02       VARV := CAR(ARV↑TIPO.00.NO);
0258 00Fe-P 02       BINHEX(ARV↑ATRIBUTO,AVATR,4,0);
0259 010C-P 02       BINHEX(ARV↑PROXIMO,AEPROX,4,0);
0260 0122-P 02       MBI(AND,ALIN+I,17);
0261 0134-P 02       IF ARV↑FILHO <> NULO THEN
0262 013E-P 02          BEGIN
0263 013E-P 03            I :=I+ 18;
0264 0145-P 03            DESENHA(ARV↑FILHO);
0265 0156-P 03            I := I - 18;
0266 0162-P 03          END
0267 0162-P 02       ELSE
0268 0168-P 02            LIST(ALIN,I+17);
0269 0174-P 02       IF ARV↑IRMAO <> NULO THEN
0270 017E-P 02          BEGIN
0271 017E-P 03            FILL(ALIN+1,132," ");
0272 0190-P 03            DESENHA(ARV↑IRMAO);
0273 019E-P 03          END;
0274 019E-P 02       RESTAURA(AVARIAVEIS);
0275 01A6-P 02 ?C
0276 01A6-P 02    END DESENHA;
```

```
0278 0123-D 01 PROCEDURE PREENCHE.CONJUNTOS;
0279 0123-D 01    %      ------------------
0280 0123-D 01    BEGIN
0281 014A-P 02
0282 014A-P 02      BYTE    I;
0283 0124-D 02
0284 0124-D 02      BYTE(*) SIMBOLO.LEXICO = (0 , 17 & ;
0285 0125-D 02                               ,0 ,  2 & END
0286 0127-D 02                               ,0 , 33 & ELSE
0287 0129-D 02                               ,0 , 28 & UNTIL
0288 012B-D 02                               ,0 , 20 & )
0289 012D-D 02                               ,0 , 16 & ,
0290 012F-D 02                               ,0 , 32 & THEN
0291 0131-D 02                               ,0 , 31 & DO
0292 0133-D 02                               ,0 , 29 & TO
0293 0135-D 02                               ,1 , 29 & DOWNTO
0294 0137-D 02                               ,0 , 27 & STEP
0295 0139-D 02                               ,0 , 30 & OF
0296 013B-D 02                               );
0297 013C-D 02
0298 013C-D 02      FOR I := 0 TO NUMERO.DE.STEND - 1 DO
0299 014A-P 02           ELEMENTO.STEND(I) := TOKEN.NUMERO(SIMBOLO.LEXICO(I));
0300 01E0-P 02      FOR I := 0 TO NUMERO.DE.OPEND - 1 DO
0301 01F0-P 02           ELEMENTO.OPEND(I) := TOKEN.NUMERO(SIMBOLO.LEXICO(I));
0302 0216-P 02
0303 0216-P 02    END PREENCHE.CONJUNTOS;
```

```
0305 013E-D 01 & ROTINAS PARA MANIPULACAO DE TOKENS:
0306 013E-D 01 & ==================================
0307 013E-D 01
0308 013F-D 01
0309 013E-D 01 BYTE PROCEDURE TOKEN (BYTE T);
0310 013E-D 01     &             -----
0311 013E-D 01     &
0312 013E-D 01     & INFORMA SE TOKEN CORRENTE E' DO TIPO T. SE FOR, LE OUTRO.
0313 013E-D 01     &
0314 013E-D 01     &
0315 013E-D 01     BEGIN
0316 0220-P 02       IF TOKEN := (TIPO.DO.TOKEN = T) THEN
0317 0232-P 02            LE.TOKEN;
0318 0236-P 02     END TOKEN;
0319 0140-D 01
0320 0140-D 01
0321 0140-D 01 PROCEDURE       PROCURA (BYTE T);
0322 0140-D 01     &           -------
0323 0140-D 01     &
0324 0140-D 01     & LE TOKENS ATE' ENCONTRAR UM DE TIPO T.
0325 0140-D 01     &
0326 0140-D 01     &
0327 0140-D 01     BEGIN
0328 0244-P 02       REPEAT
0329 0244-P 02            LE.TOKEN
0330 0244-P 02       UNTIL TOKEN(T) OR TIPO.DO.TOKEN = FIM.DE.FONTE;
0331 025E-P 02     END PROCURA;
0332 0141-D 01
0333 0141-D 01                  ____
0334 0141-D 01 PROCEDURE       EXIJA (BYTE T);
0335 0141-D 01     &           -----
0336 0141-D 01     &
0337 0141-D 01     & ACUSA ERRO SE TOKEN NAO FOR DO TIPO T.
0338 0141-D 01     &
0339 0141-D 01     &
0340 0141-D 01     BEGIN
0341 0268-P 02         IF NOT TOKEN(T) THEN
0342 0278-P 02            BEGIN
0343 0278-P 03              ERRO(FALTSIMB);
0344 0282-P 03              PROCURA(T);
0345 028C-P 03            END;
0346 028C-P 02     END EXIJA;
```

```
0348 0142-D 01 PROCEDURE PROGRAMA;
0349 0142-D 01     %       --------
0350 0142-D 01     BEGIN
0351 025E-P 02
0352 025E-P 02       WORD ENDER.INSTR.TEXTO; % ATRIBUTO DE TIPO CODIGO.
0353 0144-D 02       BYTE TEXTO;             % ATRIBUTO LOGICO, INDICA SE ESTA' FORA DE MACRO.
0354 0145-D 02       BYTE NUMERO.DE.CELULAS; % ATRIBUTO DE TIPO CELULA, CONTADOR.
```

```
0356 0146-D 02 PROCEDURE DEFINICAO;
0357 0146-D 02     %       ---------
0358 0146-D 02     %
0359 0146-D 02     % TRADUZ <DEFINICAO>
0360 0146-D 02     %
0361 0146-D 02     %
0362 0146-D 02     BEGIN
0363 0292-P 03
0364 0292-P 03     % VARIAVEIS LOGICAS:
0365 0292-P 03
0366 0292-P 03       BYTE       VEIO.ORIG;                % INDICA OCORRENCIA DE "ORIGINAL"
0367 0147-D 03
0368 0147-D 03     % VARIAVEIS RECURSIVAS:
0369 0147-D 03
0370 0147-D 03       BYTE(*)    VARIAVEIS.CL              = (2,0,0);
0371 014A-D 03            WORD ARV.CL                    POS VARIAVEIS.CL + 1;
0372 014A-D 03
0373 014A-D 03       BYTE(*)    VARIAVEIS                 = (2,0,0);
0374 014D-D 03            WORD ENDERCOD.MACRO.CHAMADA    POS VARIAVEIS + 1;
```

```
0376 014D-D 03 WORD PAL.TST;
0377 014F-D 03 &    -------
0378 014F-D 03
0379 014F-D 03 BYTE PROCEDURE PERTENCE (WORD ENDER.CONJ);
0380 014F-D 03     &            --------
0381 014F-D 03     &
0382 014F-D 03     & VERIFICA SE "PAL.TST" PERTENCE AO CONJUNTO ENDERECADO POR
0383 014F-D 03     &       "ENDER.CONJ"
0384 014F-D 03     &
0385 014F-D 03     BEGIN
0386 029E-P 04       BYTE CONT;
0387 0153-D 04       BYTE NUM.ELEM REF ENDER.CONJ;
0388 0153-D 04       WORD(*) ELEM POS NUM.ELEM + 1;
0389 0153-D 04
0390 0153-D 04       IF NOT (PERTENCE := TIPO.DO.TOKEN = PALAVRA) THEN
0391 02B0-P 04            RETURN;
0392 02B6-P 04       FOR CONT := 0 TO NUM.ELEM-1 DO
0393 02CA-P 04            IF PERTENCE := (ELEM(CONT) = PAL.TST) THEN
0394 02E8-P 04               RETURN;
0395 02E6-P 04
0396 02E6-P 04     END PERTENCE;
```

```
0398 0154-D 03 GLOBAL BYTE PROCEDURE INI (WORD ARVORE);
0399 0154-D 03    &                    ---
0400 0154-D 03    &
0401 0154-D 03    & VERIFICA SE "PAL.TST" E' INICIO VALIDO DA ARVORE
0402 0154-D 03    &       "ARV".
0403 0154-D 03    &
0404 0154-D 03    BEGIN
0405 0300-P 04      BYTE(*)   VARIAVEIS = (5,5:0);
0406 0150-D 04            WORD ARV        POS VARIAVEIS + 1;
0407 0150-D 04            WORD FIL.CORR   POS VARIAVEIS + 3;
0408 0150-D 04            BYTE RESULT     POS VARIAVEIS + 5;
0409 0150-D 04
0410 0150-D 04    BEGIN
0411 0300-P 05      BYTE PROCEDURE INI (WORD ARV); EXTERNAL;
0412 0150-D 05
0413 0150-D 05      PROCEDURE ALTERNATIVAS;
0414 0150-D 05         BEGIN
0415 0304-P 06           FIL.CORR := ARV↑FILHO;
0416 030E-P 06           REPEAT
0417 030E-P 06                 RESULT := INI(FIL.CORR)
0418 0318-P 06           UNTIL (FIL.CORR := FIL.CORR↑IRMAO) = NULO OR RESULT;
0419 0330-P 06         END ALTERNATIVAS;
0420 0150-D 05
0421 0150-D 05      SALVA(&VARIAVEIS); ARV := ARVORE;
0422 0304-P 05      CASE ARV↑TIPO.DO.NO OF
0423 0350-P 05         BEGIN
0424 0350-P 06           &NO.PALAVRA:
0425 0350-P 06                 RESULT := (ARV↑ATRIBUTO = PAL.TST);
0426 0360-P 06           &NO.STEND:
0427 0360-P 06                 ;
0428 0368-P 06           &NO.APEND:
0429 0368-P 06                 ;
0430 036A-P 06           &NO.OPCIONAL:
0431 0362-P 06                 ALTERNATIVAS;
0432 0370-P 06           &NO.OBRIGATORIO:
0433 0370-P 06                 ALTERNATIVAS;
0434 0376-P 06           &NO.OPCIONAL.REPETITIVO:
0435 0376-P 06                 ALTERNATIVAS;
0436 037C-P 06           &NO.OBRIGATORIO.REPETITIVO:
0437 037C-P 06                 ALTERNATIVAS;
0438 0382-P 06           &NO.BARRA.BARRA:
0439 0382-P 06                 ALTERNATIVAS;
0440 0386-P 06           &NO.CONCATENACAO:
0441 038A-P 06                 RESULT := INI(ARV↑FILHO);
0442 039C-P 06           &NO.CLAUSULA.SEM.PARAMETRO:
0443 039C-P 06                 RESULT := INI(ARV↑FILHO);
0444 0390-P 06           &NO.CLAUSULA.PARAMETRO:
0445 0390-P 06                 RESULT := INI(ARV↑FILHO);
0446 03C4-P 06         END;
0447 03DE-P 05    END;
0448 03DE-P 04      INI := RESULT;
0449 03E6-P 04      RESTAURA(&VARIAVEIS);
0450 03E0-P 04    END INI;
```

```
0452 015D-D 03 BYTE PROCEDURE DESCE (WORD ARV);
0453 015D-D 03     &            -----
0454 015D-D 03     &
0455 015D-D 03     & VERIFICA SE PALAVRA CORRENTE INICIA "ARV".
0456 015D-D 03     &
0457 015D-D 03     &
0458 015D-D 03     BEGIN
0459 03FE-P 04       PAL.TST := NUMERO.DO.TOKEN;
0460 040E-P 04       DESCE := INI(ARV);
0461 0414-P 04     END DESCE;
0462 0160-D 03
0463 0160-D 03 PROCEDURE TRADU (WORD ARVORE);
0464 0160-D 03     &       -----
0465 0160-D 03     &
0466 0160-D 03     & TRADUZ TRECHO DE UMA CHAMADA, ORIENTANDO-SE PELA ARVORE
0467 0160-D 03     &       "ARV".
0468 0160-D 03     &
0469 0160-D 03     BEGIN
0470 0422-P 04       BYTE(*)    VARIAVEIS = (6,6:0);
0471 0169-D 04            WORD ARV        POS VARIAVEIS + 1;
0472 0169-D 04            WORD LFIMPARM   POS VARIAVEIS + 3;
0473 0169-D 04            WORD LPARM      POS VARIAVEIS + 5;
0474 0169-D 04
0475 0169-D 04     & CONJUNTOS:
0476 0169-D 04
0477 0169-D 04       BYTE(3) CONJUNTO.UNITARIO = (1,0,0);
0478 016C-D 04            WORD ELEM POS CONJUNTO.UNITARIO + 1;
```

```
0480 016C-D 04 PROCEDURE QUERO.PAL (ARRD ENDER.CONJ);
0481 016C-D 04     &       ---------
0482 016C-D 04     &
0483 016C-D 04     & EXIGE QUE O SIMBOLO CORRENTE SEJA UMA PALAVRA
0484 016C-D 04     &       PERTENCENTE AO CONJUNTO ENDERECADO POR
0485 016C-D 04     &       "ENDER.CONJ".
0486 016C-D 04     &
0487 016C-D 04     BEGIN
0488 042E-P 05       BYTE PRIMEIRA.VEZ;
0489 016F-D 05
0490 016F-D 05       PRIMEIRA.VEZ := VERO;
0491 0432-P 05       REPEAT
0492 0432-P 05         BEGIN
0493 0432-P 06           IF PERTENCE(ENDER.CONJ) THEN
0494 0440-P 06                RETURN;
0495 0442-P 06           IF PRIMEIRA.VEZ THEN
0496 0448-P 06              BEGIN
0497 0448-P 07                ERRO(CHAMMCRINV);
0498 0452-P 07                PRIMEIRA.VEZ := FALSO;
0499 0456-P 07              END;
0500 0456-P 06         END
0501 0456-P 05         UNTIL NOT TOKEN(PALAVRA);
0502 0464-P 05
0503 0464-P 05     END QUERO.PAL;
```

```
0505 016F-D 04 PROCEDURE TRADU.ALT;
0506 016F-D 04     &       ---------
0507 016F-D 04     &
0508 016F-D 04     & TRADUZ <ALTERNATIVA> EM CHAMADA DE MACROS.
0509 016F-D 04     &
0510 016F-D 04     &
0511 016F-D 04     BEGIN
0512 0466-P 05
0513 0466-P 05     & VARIAVEIS RECURSIVAS:
0514 0466-P 05
0515 0466-P 05       BYTE(*)   VARIAVEIS      = (12,12:0);
0516 017C-D 05             WORD ARV.ALT.CORP   POS VARIAVEIS +  1;
0517 017C-D 05             WORD ARV.ALT.CERTA  POS VARIAVEIS +  3;
0518 017C-D 05             WORD LPRIMITE       POS VARIAVEIS +  5;
0519 017C-D 05             WORD LPROXITE       POS VARIAVEIS +  7;
0520 017C-D 05             WORD LFIMDASITE     POS VARIAVEIS +  9;
0521 017C-D 05             BYTE INICIO         POS VARIAVEIS + 11;
0522 017C-D 05             BYTE TEM.ITERACAO   POS VARIAVEIS + 12;
```

```
0524 017C-D 05 PROCEDURE ANULE (WORD ARVORE);
0525 017C-D 05     &       -----
0526 017C-D 05     &
0527 017C-D 05     & GERA CODIGO INTERMEDIARIO PARA INDICAR AUSENCIA
0528 017C-D 05     &       DE UMA <ALTERNATIVA>
0529 017C-D 05     &
0530 017C-D 05     BEGIN
0531 0472-P 06
0532 0472-P 06     & VARIAVEIS RECURSIVAS:
0533 0472-P 06
0534 0472-P 06       BYTE(*)    VARIAVEIS  = (4,4:0);.
0535 0183-D 06            WORD ARV          POS VARIAVEIS + 1;
0536 0183-D 06            WORD FILHO.CORR POS VARIAVEIS + 3;
0537 0183-D 06
0538 0183-D 06     & COMANDOS DA ANULE:
0539 0183-D 06
0540 0183-D 06       SALVA(&VARIAVEIS); ARV := ARVORE;
0541 0484-P 06       IF ARV↑TIPO.DO.NO = NO.CONCATENACAO OR
0542 048C-P 06          ARV↑TIPO.DO.NO = NO.BARRA.BARRA  THEN
0543 049C-P 06          BEGIN
0544 049C-P 07             FILHO.CORR := ARV↑FILHO;
0545 04A6-P 07             REPEAT
0546 04A6-P 07                  ANULE(FILHO.CORR)
0547 04B0-P 07             UNTIL(FILHO.CORR := FILHO.CORR↑IRMAO)=NULO;
0548 04BE-P 07          END
0549 04BE-P 06       ELSE IF ARV↑TIPO.DO.NO <> NO.PALAVRA THEN
0550 04C8-P 06          BEGIN
0551 04C8-P 07            GRAV.INTER(CARREGA.IMEDIATO,NULO);
0552 04D2-P 07            GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO);
0553 04E2-P 07          END;
0554 04E2-P 06       RESTAURA(&VARIAVEIS);
0555 04EC-P 06
0556 04EC-P 06     END ANULE;
```

```
0558 0183-D 05    & COMANDOS DA TRADU.ALT:
0559 0183-D 05    &                ---------
0560 0183-D 05      SALVA(&VARIAVEIS);
0561 04F8-P 05      INICIO := TEM.ITERACAO := VERO;
0562 0500-P 05      REPEAT
0563 0500-P 05         BEGIN
0564 0500-P 06           ARV.ALT.CORR := ARV↑FILHO;
0565 050A-P 06           ARV.ALT.CERTA := NULO;
0566 050E-P 06           REPEAT
0567 050E-P 06                IF DESCE(ARV.ALT.CORR) THEN
0568 051C-P 06                     ARV.ALT.CERTA := ARV.ALT.CORR
0569 051C-P 06                ELSE
0570 0526-P 06                     ANULE(ARV.ALT.CORR)
0571 0530-P 06           UNTIL (ARV.ALT.CORR := ARV.ALT.CORR↑IRMAO)=NULO;
0572 053E-P 06           IF ARV.ALT.CERTA = NULO THEN
0573 0544-P 06                TEM.ITERACAO := FALSO
0574 0544-P 06           ELSE
0575 054A-P 06              BEGIN
0576 054A-P 07                IF INICIO THEN
0577 0550-P 07                   BEGIN
0578 0550-P 08                     GRAV.INTER(CARREGA.ENDERECO,LPRIMITE:=LPROXITE:=NORD);
0579 0566-P 08                     GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO+1);
0580 0578-P 08                     GRAV.INTER(CARREGA.IMEDIATO,PRESENTE);
0581 0582-P 08                     GRAV.INTER(DESVIA,LFIMDASITE:=NORD);
0582 0594-P 08                     GRAV.DEF(LPROXITE);
0583 059E-P 08                   END;
0584 059E-P 07                GRAV.INTER(CARREGA.ENDERECO,LPROXITE:=NORD);
0585 05B0-P 07                GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO+1);
0586 05C2-P 07                GRAV.INTER(CARREGA.IMEDIATO,PRESENTE);
0587 05CC-P 07                GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO);
0588 05DC-P 07                TRADU(ARV.ALT.CERTA);
0589 05E6-P 07                GRAV.INTER(RETORNA,0);
0590 05E0-P 07                GRAV.DEF(LPROXITE);
0591 05FA-P 07                INICIO := FALSO;
0592 05FE-P 07              END;
0593 05FE-P 06           IF ARV↑TIPO.DO.NO = NO.OPCIONAL     OR
0594 0606-P 06              ARV↑TIPO.DO.NO = NO.OBRIGATORIO THEN
0595 061C-P 06                 TEM.ITERACAO := FALSO;
0596 061A-P 06         END
0597 061A-P 05      UNTIL NOT TEM.ITERACAO;
0598 0624-P 05      IF INICIO THEN
0599 062A-P 05         BEGIN
0600 062A-P 06           GRAV.INTER(CARREGA.IMEDIATO,NULO);
0601 0634-P 06           IF ARV↑TIPO.DO.NO = NO.OBRIGATORIO                OR
0602 063C-P 06              ARV↑TIPO.DO.NO = NO.OBRIGATORIO.REPETITIVO THEN
0603 064C-P 06                ERRO(CHAMMCRINV);
0604 0656-P 06         END
0605 0656-P 05      ELSE
0606 0656-P 05       BEGIN
0607 0658-P 06       GRAV.INTER(CARREGA.ENDERECO,LPRIMITE);GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO+1)
0608 0676-P 06       GRAV.INTER(CARREGA.IMEDIATO,NULO);    GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO);
0609 0690-P 06       GRAV.INTER(RETORNA,0);
0610 069A-P 06       GRAV.DEF(LFIMDASITE);
0611 06A4-P 06       END;
```

```
0612 06A4-P 05      GRAV.INTER(ARMAZENA,ARVTATRIBUTO);
0613 06B4-P 05      RESTAURA(XVARTAVFIS);
0614 06BE-P 05    END TRADU.ALT;
```

```
0616 0183-D 04 PROCEDURE TRADU.BARRA.BARRA;
0617 0183-D 04    &       -----------------
0618 0183-D 04    &
0619 0183-D 04    & TRADUZ <INVERSAO> EM <CHAMADA DE MACRO>
0620 0183-D 04    &
0621 0183-D 04    &
0622 0183-D 04    BEGIN
0623 06C0-P 05      IF DESCE(ARV↑FILHO) THEN
0624 06D2-P 05         BEGIN
0625 06D2-P 06           TRADU(ARV↑FILHO);
0626 06E0-P 06           TRADU(ARV↑FILHO↑IRMAO);
0627 06F2-P 06         END
0628 06F2-P 05      ELSE
0629 06F4-P 05         BEGIN
0630 06F4-P 06           TRADU(ARV↑FILHO↑IRMAO);
0631 0706-P 06           TRADU(ARV↑FILHO);
0632 0714-P 06         END;
0633 0714-P 05    END TRADU.BARRA.BARRA;
```

```
0635 0183-D 04 PROCEDURE RECON.FILS;
0636 0183-D 04    &       ----------
0637 0183-D 04    &
0638 0183-D 04    & TRADUZ FILHOS DA ARVORE "ARV".
0639 0183-D 04    &
0640 0183-D 04    &
0641 0183-D 04    BEGIN
0642 0716-P 05
0643 0716-P 05    & VARIAVEIS RECURSIVAS:
0644 0716-P 05
0645 0716-P 05      BYTE(*)    VARIAVEIS  = (2,2:0);
0646 0186-D 05            WORD FILHO.CORR POS VARIAVEIS + 1;
0647 0186-D 05
0648 0186-D 05    & COMANDOS DE RECON.FILS:
0649 0186-D 05
0650 0186-D 05      SALVA(&VARIAVEIS);
0651 0720-P 05      FILHO.CORR := ARV↑FILHO;
0652 0724-P 05      REPEAT
0653 0724-P 05           TRADU(FILHO.CORR)
0654 0734-P 05      UNTIL (FILHO.CORR := FILHO.CORR↑IRMAO) = NULO;
0655 0742-P 05      RESTAURA(&VARIAVEIS);
0656 074C-P 05
0657 074C-P 05    END RECON.FILS;
```

```
0659 0186-D 04    & COMANDOS DA TRADU:
0660 0186-D 04    &              -----
0661 0186-D 04      SALVA(ÃVARIAVEIS); ARV := ARVORE;
0662 0760-P 04      SALVA(ÃVARIAVEIS.CL);
0663 076A-P 04      DEIXA.RASTRO(20+ARV↑TIPO.DO.NO*2);
0664 077A-P 04      PAL.TST := NUMERO.DO.TOKEN;
0665 0782-P 04      CASE ARV↑TIPO.DO.NO OF
0666 078E-P 04         BEGIN
0667 078E-P 05              BEGIN                              & NO.PALAVRA
0668 078E-P 06                ELEM := ARV↑ATRIBUTO;
0669 0798-P 06                QUERO.PAL(ÃCONJUNTO.UNITARIO);
0670 07A2-P 06                EXIJA(PALAVRA);
0671 07AA-P 06              END;
0672 07AE-P 05              BEGIN                              & NO.STEND
0673 07AE-P 06                RECON.FILS;
0674 07B2-P 06                QUERO.PAL(ÃCONJUNTO.STEND);
0675 07BC-P 06              END;
0676 07BE-P 05              BEGIN                              & NO.OPEND
0677 07BE-P 06                RECON.FILS;
0678 07C2-P 06                QUERO.PAL(ÃCONJUNTO.OPEND);
0679 07CC-P 06              END;
0680 07CE-P 05              TRADU.ALT;                         & NO.OPCIONAL
0681 07D4-P 05              TRADU.ALT;                         & NO.OBRIGATORIO
0682 07DA-P 05              TRADU.ALT;                         & NO.OPCIONAL.REPETITIVO
0683 07E0-P 05              TRADU.ALT;                         & NO.OBRIGATORIO.REPETITIVO
0684 07E6-P 05              TRADU.BARRA.BARRA;                 & NO.BARRA.BARRA
0685 07EC-P 05              RECON.FILS;                        & N.
0686 07F2-P 05              BEGIN                              & NCL
0687 07F2-P 06                RECON.FILS;
0688 07F6-P 06                GRAV.INTER(CARREGA.IMEDIATO,PRESENTE);
0689 0800-P 06                GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO);
0690 0810-P 06              END;
0691 0812-P 05              BEGIN                              & NCLP
0692 0812-P 06                RECON.FILS;
0693 0816-P 06                GRAV.INTER(DESVIA,LFIMPARM:=NORD);
0694 0828-P 06                GRAV.DEF(LPARM:=NORD);
0695 0836-P 06                ARV.CL := ARV;
0696 083E-P 06                DEFINICAO;
0697 0842-P 06                GRAV.INTER(RETORNA,0);
0698 084C-P 06                GRAV.DEF(LFIMPARM);
0699 0856-P 06                GRAV.INTER(CARREGA.ENDERECO,LPARM);
0700 0862-P 06                GRAV.INTER(ARMAZENA,ARV↑ATRIBUTO);
0701 0872-P 06              END
0702 0872-P 05           END;
0703 087A-P 04      DEIXA.RASTRO(21+ARV↑TIPO.DO.NO*2);
0704 0894-P 04      RESTAURA(ÃVARIAVEIS.CL);
0705 08A4-P 04      RESTAURA(ÃVARIAVEIS);
0706 08AE-P 04    END TRADU;
```

```
0708 0186-D 03 GLOBAL
0709 0186-D 03 BYTE PROCEDURE TERMINADOR (WORD ARVORE);
0710 0186-D 03    &            ==========
0711 0186-D 03    &
0712 0186-D 03    & INFORMA SE A PALAVRA CORRENTE TERMINA O PARAMETRO
0713 0186-D 03    &      EFETIVO CORRENTE.
0714 0186-D 03    &
0715 0186-D 03    BEGIN
0716 0888-P 04      BYTE(*)   VARIAVEIS = (5,5:0);
0717 018F-D 04            WORD ARV       POS VARIAVEIS + 1;
0718 018F-D 04            WORD ARV.PROX  POS VARIAVEIS + 3;
0719 018F-D 04            BYTE RESULT    POS VARIAVEIS + 5;
0720 018F-D 04
0721 018F-D 04    & COMANDOS DA TERMINADOR:
0722 018F-D 04
0723 018F-D 04      SALVA(@VARIAVEIS);  ARV := ARVORE;
0724 08CA-P 04      IF ARV=NULO THEN
0725 08D0-P 04            RESULT := FALSO
0726 08D0-P 04      ELSE
0727 08D6-P 04         BEGIN
0728 08D6-P 05           BYTE PROCEDURE TERMINADOR (WORD ARV); EXTERNAL;
0729 018F-D 05           ARV.PROX := ARV↑PROXIMO;
0730 08E0-P 05           PAL.TST := NUMERO.DO.TOKEN;
0731 08E8-P 05           RESULT := IF ARV.PROX↑TIPO.DO.NO=NO.OPCIONAL              OR
0732 08F0-P 05                       ARV.PROX↑TIPO.DO.NO=NO.OPCIONAL.REPETITIVO THEN
0733 0900-P 05                         IF INI(ARV.PROX) THEN
0734 090E-P 05                              VERO
0735 090E-P 05                         ELSE
0736 0912-P 05                              TERMINADOR(ARV.PROX)
0737 091C-P 05                    ELSE IF ARV.PROX↑TIPO.DO.NO=NO.STEND THEN
0738 092A-P 05                         PERTENCE(@CONJUNTO.STEND)
0739 0934-P 05                    ELSE IF ARV.PROX↑TIPO.DO.NO=NO.OPEND THEN
0740 0940-P 05                         PERTENCE(@CONJUNTO.OPEND)
0741 094A-P 05                    ELSE
0742 094C-P 05                         INI(ARV.PROX);
0743 095C-P 05           END;
0744 095C-P 04      TERMINADOR := RESULT;
0745 0964-P 04      RESTAURA(@VARIAVEIS);
0746 096E-P 04
0747 096E-P 04    END TERMINADOR;
```

```
0748 018F-D 03 PROCEDURE TESTE;
0750 018F-D 03     %       -----
0751 018F-D 03     %
0752 018F-D 03     % TRADUZ <TESTE>.
0753 018F-D 03     %
0754 018F-D 03     %
0755 018F-D 03     BEGIN
0756 0974-P 04
0757 0974-P 04     % VARIAVEIS RECURSIVAS:
0758 0974-P 04
0759 0974-P 04       BYTE(*)   VARIAVEIS       = (11,11:0);
0760 0198-D 04            WORD LELSE           POS VARIAVEIS +  1;
0761 0198-D 04            WORD LEORA           POS VARIAVEIS +  3;
0762 0198-D 04            WORD LPPT            POS VARIAVEIS +  5;
0763 0198-D 04            WORD LPROX           POS VARIAVEIS +  7;
0764 0198-D 04            WORD LPREP           POS VARIAVEIS +  9;
0765 0198-D 04            BYTE ENDERPIL.TESTE POS VARIAVEIS + 11;
0766 0198-D 04
0767 0198-D 04     % VARIAVEIS NAO RECURSIVAS:
0768 0198-D 04
0769 0198-D 04       BYTE TEM.ELSE;  % VARIAVEL LOGICA
```

```
0771 019C-D 04 PROCEDURE TESTE.DE.ALTERNATIVAS;
0772 019C-D 04    &      ----------------------
0773 019C-D 04    BEGIN
0774 0976-P 05
0775 0976-P 05      BYTE TESTE.DE.CLAUSULA;  & VARIAVEL LOGICA (TESTE E' DE CLAU
0776 019D-D 05
0777 019D-D 05      DEIXA.RASTRO(18);
0778 0980-P 05      IF TOKEN(ABRE.CHAVES) THEN
0779 098C-P 05         BEGIN
0780 098C-P 06           TESTE;
0781 0990-P 06           PESQUISA.CELULA(DE.GRUPO,CELULA.DO.GRUPO.DA.PESQUISADA);
0782 099C-P 06           TESTE.DE.CLAUSULA := FALSO;
0783 09A0-P 06         END
0784 09A0-P 05      ELSE
0785 09A2-P 05         BEGIN
0786 09A2-P 06           IF TOKEN(IDENTIFICADOR.DE.PARAMETRO) THEN
0787 09AE-P 06                PESQUISA.CELULA(DE.PARAMETRO,NUMERO.DO.TOKEN.ANTERIOR)
0788 09B4-P 06           ELSE
0789 09BC-P 06              BEGIN
0790 09BC-P 07                EXIJA(PALAVRA);
0791 09C4-P 07                PESQUISA.CELULA(DE.CLAUSULA,NUMERO.DO.TOKEN.ANTERIOR);
0792 09D0-P 07              END;
0793 09D0-P 06           TESTE.DE.CLAUSULA := VERO;
0794 09D4-P 06         END;
0795 09D4-P 05      GRAV.INTER(CARREGA.DADO,CELULA.PESQUISADA);
0796 09E0-P 05      GRAV.INTER(DESVIA.SE.NULO,LPROX:=NORO);
0797 09E2-P 05      IF TESTE.DE.CLAUSULA THEN
0798 09E8-P 05         BEGIN
0799 09E8-P 06           EXIJA(TDEFINE);
0800 0A00-P 06           DEFINICAO;
0801 0A04-P 06         END;
0802 0A04-P 05      GRAV.INTER(DESVIA,LPREP);
0803 0A10-P 05      GRAV.DEF(LPROX);
0804 0A14-P 05      IF CELULA.DO.GRUPO.DA.PESQUISADA<>ENDERPIL.TESTE AND
0805 0A22-P 05         ENDERPIL.TESTE <> 0FFFF       THEN
0806 0A32-P 05           ERRO(ITEINV);
0807 0A3C-P 05      ENDERPIL.TESTE := CELULA.DO.GRUPO.DA.PESQUISADA;
0808 0A44-P 05      DEIXA.RASTRO(19);
0809 0A4C-P 05    END TESTE.DE.ALTERNATIVAS;
```

```
0811 0190-D 04     & ROTULOS:
0812 0190-D 04
0813 0190-D 04       LABEL RPT;
0814 0190-D 04
0815 0190-D 04     & COMANDOS DA TESTE:
0816 0190-D 04     &                -----
0817 0190-D 04       DEIXA.RASTRO(16);
0818 0A50-P 04       SALVA(XVARIAVEIS);
0819 0A60-P 04       GRAV.INTER(DESVIA,LELSE:=NORD);
0820 0A72-P 04       GRAV.DEF(LRPT:=NORD);
0821 0A80-P 04       LPREP := NORD;
0822 0A86-P 04       ENDERPIL.TESTE := OFF;
0823 0A8E-P 04     RPT:
0824 0A8E-P 04       TESTE.DE.ALTERNATIVAS;
0825 0A92-P 04       TEM.ELSE := FALSO;
0826 0A96-P 04       IF TOKEN(BARRA) THEN
0827 0AA2-P 04            IF TOKEN(TDEFINE) THEN
0828 0AAE-P 04                 TEM.ELSE := VERO
0829 0AAE-P 04            ELSE
0830 0AB4-P 04                 GOTO RPT;
0831 0AB6-P 04       GRAV.DEF(LPREP);
0832 0AC0-P 04       GRAV.INTER(CHAMA.PARAMETRO,ENDERPIL.TESTE+1);
0833 0AD0-P 04       GRAV.INTER(CARREGA.DADO,ENDERPIL.TESTE);
0834 0ADC-P 04       GRAV.INTER(DESVIA.SE.PRESENTE,LRPT);
0835 0AE8-P 04       GRAV.INTER(MARCA.DADO,ENDERPIL.TESTE);
0836 0AF4-P 04       GRAV.INTER(DESVIA,LFORA:=NORD);
0837 0B06-P 04       GRAV.DEF(LELSE);
0838 0B10-P 04       GRAV.INTER(CARREGA.DADO,ENDERPIL.TESTE);
0839 0B1C-P 04       GRAV.INTER(DESVIA.SE.PRESENTE,LPREP);
0840 0B28-P 04       IF TEM.ELSE THEN
0841 0B2E-P 04            DEFINICAO;
0842 0B32-P 04       GRAV.DEF(LFORA);
0843 0B3C-P 04       EXIJA(FECHA.CHAVES);
0844 0B44-P 04       RESTAURA(XVARIAVEIS);
0845 0B4E-P 04       DEIXA.RASTRO(17);
0846 0B56-P 04
0847 0B56-P 04     END TESTE;
```

```
0849 0190-D 03    & COMANDOS DA DEFINICAO:
0850 0190-D 03    &           ---------
0851 0190-D 03      DEIXA.RASTRO(14);
0852 0860-P 03      SALVA(AVARIAVEIS);
0853 0866-P 03      REPEAT
0854 0866-P 03         BEGIN
0855 0866-P 04           IF TOKEN(TORIG) THEN
0856 0876-P 04                IF TOKEN(PALAVRA) THEN
0857 0882-P 04                     GRAV.INTER(TRANSMITE,NUMERO.DO.TOKEN.ANTERIOR)
0858 088E-P 04                ELSE
0859 0890-P 04                     ERRO(ORIGINV)
0860 089A-P 04           ELSE IF TIPO.DO.TOKEN = PALAVRA THEN
0861 08A4-P 04              BEGIN
0862 08A4-P 05                IF TERMINADOR(ARV.CL) THEN
0863 08B2-P 05                   BEGIN
0864 08B2-P 06                     DEIXA.RASTRO(15);
0865 08B4-P 06                     RESTAURA(AVARIAVEIS);
0866 08C4-P 06                     RETURN
0867 08C6-P 06                   END
0868 08C6-P 05                ELSE IF INICIO.DE.TRECHO.SUBSTITUIVEL(NUMERO.DO.TOKEN) THEN
0869 08D6-P 05                   BEGIN
0870 08D6-P 06                     GRAV.INTER(ALOCA,CELULAS.INFORMADAS);
0871 08E2-P 06                     ENDERCOD.MACRO.CHAMADA := CODIGO.INFORMADO;
0872 08EA-P 06                     TRADU(ARVORE.INFORMADA);
0873 08F4-P 06                     GRAV.INTER(CHAMA,ENDERCOD.MACRO.CHAMADA);
0874 0C00-P 06                   END
0875 0C00-P 05                ELSE
0876 0C02-P 05                   BEGIN
0877 0C02-P 06                     GRAV.INTER(TRANSMITE,NUMERO.DO.TOKEN);
0878 0C0E-P 06                     LE.TOKEN;
0879 0C12-P 06                   END
0880 0C12-P 05              END
0881 0C12-P 04           ELSE IF TOKEN(DOLAR) THEN
0882 0C20-P 04              GRAV.INTER(TRANSMITE,NUMERO.DO.TOKEN.ANTERIOR)
0883 0C2C-P 04           ELSE IF SCAN(A(TENDMACRO,FECHA.CHAVES,BARRA,FIM.DE.FONTE),
0884 0C32-P 04                     4,TIPO.DO.TOKEN) >= 0 THEN
0885 0C42-P 04              BEGIN
0886 0C42-P 05                DEIXA.RASTRO(15);
0887 0C4A-P 05                RESTAURA(AVARIAVEIS);
0888 0C54-P 05                RETURN
0889 0C56-P 05              END
0890 0C5E-P 04           ELSE IF TEXTO THEN
0891 0C5E-P 04                ERRO(PARMACRINV)
0892 0C68-P 04           ELSE IF TOKEN(IDENTIFICADOR.DE.PARAMETRO) THEN
0893 0C76-P 04              BEGIN
0894 0C76-P 05                IF PESQUISA.CELULA(DE.PARAMETRO,NUMERO.DO.TOKEN.ANTERIOR) THEN
0895 0C86-P 05                   GRAV.INTER(CHAMA.PARAMETRO,CELULA.PESQUISADA);
0896 0C98-P 05              END
0897 0C98-P 04           ELSE IF TOKEN(ABRE.CHAVES) THEN
0898 0CA6-P 04                TESTE
0899 0CA6-P 04           ELSE
0900 0CAC-P 04                EXIJA(TENDMACRO);
0901 0CB4-P 04         END;
0902 0CB6-P 03    END DEFINICAO;
```

```
0904 01A1-D 02 PROCEDURE DECLARACAO;
0905 01A1-D 02     %       ----------
0906 01A1-D 02     %
0907 01A1-D 02     % TRADUZ DECLARACAO DE MACRO
0908 01A1-D 02     %
0909 01A1-D 02     %
0910 01A1-D 02     BEGIN
0911 0CBA-P 03
0912 0CBA-P 03     % VARIAVEIS PARA NUMEROS DOS TOKENS:
0913 0CBA-P 03
0914 0CBA-P 03       WORD NUMERO.DO.TOKEN.DECLARACAO;
0915 01A3-D 03
0916 01A3-D 03     % VARIAVEIS PARA ENDERECOS DAS VARIAVEIS DE EXPANSAO:
0917 01A3-D 03
0918 01A3-D 03       BYTE(*)   Q          = (1,1:0);
0919 01A5-D 03            BYTE ITE.CORR POS Q + 1;
0920 01A5-D 03
0921 01A5-D 03     % VARIAVEIS PARA NUMEROS DE ORDEM (ENDERECOS INDIRETOS)
0922 01A5-D 03
0923 01A5-D 03       WORD LMCR;        % ENDERECO DA MACRO
0924 01A7-D 03
0925 01A7-D 03     % VARIAVEIS PARA APONTAR ARVORES:
0926 01A7-D 03
0927 01A7-D 03       WORD ENDER.ARVORE.DECLARACAO;   % APONTA ENDER.ARVORE DA ARVORE SINTATICA
0928 01A9-D 03       WORD ENDER.ARVORE;          APONTA ENDER.ARVORE DA ULTIMA SUB-ARVORE FORMA
0929 01AB-D 03
0930 01AB-D 03     % VARIAVEIS LOGICAS:
0931 01AB-D 03
0932 01AB-D 03       BYTE TERMINA.COM.PALAVRA;   % FIM DE SINTAXE DE MACRO PERMITIDO
```

```
0934 01AC-D 03 PROCEDURE SINTAXE;
0935 01AC-D 03     &       -------
0936 01AC-D 03     &
0937 01AC-D 03     & TRADUZ <SINTAXE>
0938 01AC-D 03     &
0939 01AC-D 03     &
0940 01AC-D 03     BEGIN
0941 0CBE-P 04
0942 0CBE-P 04     & VARIAVEIS LOCAIS DA SINTAXE:
0943 0CBE-P 04
0944 0CBE-P 04     &     NAO.RECURSIVAS:
0945 0CBE-P 04
0946 0CBE-P 04       BYTE TEM.CONCATENACAO;      & VARIAVEL LOGICA. TEM CONCATENACAO
0947 01AD-D 04                              &      DE CLAUSULAS
0948 01AD-D 04
0949 01AD-D 04     &     RECURSIVAS:
0950 01AD-D 04
0951 01AD-D 04       BYTE(*)    VARIAVEIS                          = (4,4:0);
0952 01B2-D 04            #ORD ENDER.ARVORE.ITEM.CORRENTE POS VARIAVEIS + 1;
0953 01B2-D 04            #ORD ENDER.ARVORE.PRIMEIRO.ITEM POS VARIAVEIS + 3;
```

```
0955 01B2-D 04 BYTE PROCEDURE ITEM;
0956 01B2-D 04     %            ----
0957 01B2-D 04     %
0958 01B2-D 04     % TRADUZ <ITEM>.
0959 01B2-D 04     %
0960 01B2-D 04     BEGIN
0961 0CC4-P 05
0962 0CC4-P 05       BYTE OBRIGATORIO;   % ATRIBUTO LOGICO, INDICA GRUPO.OBRIGATORIO.
```

```
0964 0184-D 05 PROCEDURE GRUPO;
0965 0184-D 05     &       -----
0966 0184-D 05     &
0967 0184-D 05     & TRADUZ <GRUPO>
0968 0184-D 05     &
0969 0184-D 05     &
0970 0184-D 05     BEGIN
0971 0CC8-P 06       BYTE(*)    VARIAVEIS                          = (6,6:0);
0972 0188-D 06            BYTE GRUPO.OBRIGATORIO                   POS VARIAVEIS + 1;
0973 0188-D 06            WORD ARV.ALT.CORR                        POS VARIAVEIS + 2;
0974 0188-D 06            WORD PRIM.ARV.CONJUNTO.DE.SINTAXES       POS VARIAVEIS + 4;
0975 0188-D 06            BYTE ENDERPIL.GRUPO                      POS VARIAVEIS + 6;
0976 0188-D 06
0977 0188-D 06     & VARIAVEIS NAO-RECURSIVAS
0978 0188-D 06
0979 0188-D 06       BYTE ALTERNATIVAS;   & VAR LOGICA, INDICA BARRA.
```

```
0981 01BC-D 06 PROCEDURE CONJUNTO.DE.SINTAXES;
0982 01BC-D 06    &       --------------------
0983 01BC-D 06    &
0984 01BC-D 06    & TRADUZ CONJUNTO-DE-SINTAXES
0985 01BC-D 06    &
0986 01BC-D 06    &
0987 01BC-D 06    BEGIN
0988 0CCC-P 07
0989 0CCC-P 07    & VARIAVEIS RECURSIVAS:
0990 0CCC-P 07
0991 0CCC-P 07      BYTE(*)   VARIAVEIS = (2,2:0);
0992 01BF-D 07           WORD FILHO.ESQUERDO POS VARIAVEIS + 1;
0993 01BF-D 07
0994 01BF-D 07    & COMANDOS DA CONJUNTO.DE.SINTAXES:
0995 01BF-D 07
0996 01BF-D 07      DEIXA.RASTRO(12);
0997 0CD4-P 07      SALVA(&VARIAVEIS);
0998 0CDE-P 07      SINTAXE;
0999 0CE2-P 07      FILHO.ESQUERDO := ENDER.ARVORE;
1000 0CEA-P 07      IF TOKEN(BARRA.BARRA) THEN
1001 0CF6-P 07         BEGIN
1002 0CF6-P 08           CONJUNTO.DE.SINTAXES;
1003 0CFA-P 08           FILHO.ESQUERDO↑IRMAO := ENDER.ARVORE;
1004 0D04-P 08           (ENDER.ARVORE:=CRIA(NO.BARRA.BARRA))↑FILHO :=
1005 0D12-P 08                FILHO.ESQUERDO;
1006 0D16-P 08         END;
1007 0D16-P 07      RESTAURA(&VARIAVEIS);
1008 0D20-P 07      DEIXA.RASTRO(13);
1009 0D28-P 07
1010 0D28-P 07    END CONJUNTO.DE.SINTAXES;
```

```
1012 01BF-D 06     & ROTULOS:
1013 01BF-D 06
1014 01BF-D 06       LABEL RETORNO;
1015 01BF-D 06
1016 01BF-D 06     & COMANDOS DA GRUPO:
1017 01BF-D 06     &              -----
1018 01BF-D 06       DEIXA.RASTRO(10);
1019 0D32-P 06       SALVA(&VARIAVEIS);  GRUPO.OBRIGATORIO := OBRIGATORIO;
1020 0D44-P 06       SALVA(&Q);
1021 0D4E-P 06       ENDERPIL.GRUPO := ITE.CORR := NUMERO.DE.CELULAS;
1022 0D5A-P 06       NUMERO.DE.CELULAS := ! + 2;
1023 0D5E-P 06       CONJUNTO.DE.SINTAXES;
1024 0D62-P 06       ARV.ALT.CORR := PRIM.ARV.CONJUNTO.DE.SINTAXES :=
1025 0D62-P 06            ENDER.ARVORE;
1026 0D6E-P 06       ALTERNATIVAS := FALSO;
1027 0D72-P 06       WHILE TOKEN(BARRA) DO
1028 0D7E-P 06          BEGIN
1029 0D7E-P 07            CONJUNTO.DE.SINTAXES;
1030 0D82-P 07            ARV.ALT.CORR := ARV.ALT.CORR↑IRMAO := ENDER.ARVORE;
1031 0D90-P 07            ALTERNATIVAS := VERO;
1032 0D94-P 07          END;
1033 0D96-P 06       EXIJA(IF GRUPO.OBRIGATORIO THEN FECHA.CHAVES
1034 0D9C-P 06                                  ELSE FECHA.COLCHETES);
1035 0DA8-P 06       IF TOKEN(T...) THEN
1036 0DB4-P 06            ENDER.ARVORE := CRIA(IF GRUPO.OBRIGATORIO
1037 0DB4-P 06                                 THEN NO.OBRIGATORIO.REPETITIVO.
1038 0DBA-P 06                                 ELSE NO.OPCIONAL.REPETITIVO)
1039 0DC6-P 06       ELSE
1040 0DCC-P 06          BEGIN
1041 0DCC-P 07            IF NOT ALTERNATIVAS AND GRUPO.OBRIGATORIO THEN
1042 0DD8-P 07               BEGIN
1043 0DD8-P 08                 RESTAURA(&Q);
1044 0DE2-P 08                 GOTO RETORNO
1045 0DE2-P 08               END;
1046 0DE6-P 07            ENDER.ARVORE := CRIA(IF GRUPO.OBRIGATORIO
1047 0DE6-P 07                                 THEN NO.OBRIGATORIO
1048 0DEC-P 07                                 ELSE NO.OPCIONAL);
1049 0DFC-P 07          END;
1050 0DFC-P 06       RESTAURA(&Q);
1051 0E06-P 06       ENDER.ARVORE↑FILHO := PRIM.ARV.CONJUNTO.DE.SINTAXES;
1052 0E10-P 06       DECOPE.CELULA(DE.GRUPO,NULO,ENDER.ARVORE↑ATRIBUTO:=ENDERPIL.GRUPO,
1053 0E20-P 06                                 ITE.CORR);
1054 0E2E-P 06       ENDER.ARVORE↑FILHO := PRIM.ARV.CONJUNTO.DE.SINTAXES;
1055 0E38-P 06     RETORNO:
1056 0E38-P 06       RESTAURA(&VARIAVEIS);
1057 0E42-P 06       DEIXA.RASTRO(11);
1058 0E4A-P 06     END GRUPO;
```

```
1060 01BE-D 05 PROCEDURE CLAUSULA;
1061 01BE-D 05     &       --------
1062 01BE-D 05     &
1063 01BE-D 05     & TRADUZ <CLAUSULA>
1064 01BE-D 05     &
1065 01BE-D 05     &
1066 01BE-D 05     BEGIN
1067 0E4C-P 06
1068 0E4C-P 06     & VARIAVEIS LOCAIS DE CLAUSULA
1069 0E4C-P 06
1070 0E4C-P 06       WORD PRIM.ARV.PAL; & APONTA NO' DA PRIMEIRA PALAVRA
1071 01C1-D 06       WORD ARV.PAL.CORR; & APONTA NO' DA PALAVRA CORRENTE
1072 01C3-D 06
1073 01C3-D 06     & COMANDOS DA CLAUSULA
1074 01C3-D 06
1075 01C3-D 06       DEIXA.RASTRO(8);
1076 0E54-P 06       (PRIM.ARV.PAL:=ARV.PAL.CORR:=CRIA(NO.PALAVRA))↑ATRIBUTO :=
1077 0E66-P 06                                          NUMERO.DO.TOKEN.ANTERIOR;
1078 0E6A-P 06       WHILE TOKEN(PALAVRA) DO
1079 0E76-P 06            (ARV.PAL.CORR := ARV.PAL.CORR↑IRMAO:=CRIA(NO.PALAVRA))
1080 0E8C-P 06                 ↑ATRIBUTO := NUMERO.DO.TOKEN.ANTERIOR;
1081 0E94-P 06       IF TOKEN(DOLAR) THEN
1082 0EA0-P 06          BEGIN
1083 0EA0-P 07            TERMINA.COM.PALAVRA := FALSO;
1084 0EA4-P 07            (ENDER.ARVORE := CRIA(NO.CLAUSULA.PARAMETRO))↑FILHO :=
1085 0EB2-P 07                 PRIM.ARV.PAL;
1086 0EB6-P 07            DECORE.CELULA(DE.PARAMETRO,PRIM.ARV.PAL↑ATRIBUTO,
1087 0EC0-P 07                                  NUMERO.DE.CELULAS,ITE.CORR);
1088 0ECE-P 07            DECORE.CELULA(DE.CLAUSULA, PRIM.ARV.PAL↑ATRIBUTO,
1089 0EDC-P 07                                  NUMERO.DE.CELULAS,ITE.CORR);
1090 0EE6-P 07          END
1091 0EE6-P 06       ELSE IF TOKEN(IDENTIFICADOR.DE.PARAMETRO) THEN
1092 0EF4-P 06          BEGIN
1093 0EF4-P 07            TERMINA.COM.PALAVRA := FALSO;
1094 0EF8-P 07            (ENDER.ARVORE := CRIA(NO.CLAUSULA.PARAMETRO))↑FILHO :=
1095 0F06-P 07                 PRIM.ARV.PAL;
1096 0F0A-P 07            DECORE.CELULA(DE.PARAMETRO,NUMERO.DO.TOKEN.ANTERIOR,NUMERO.DE.CELULA
1097 0F14-P 07                                                    ITE.CORR);
1098 0F1E-P 07          END
1099 0F1E-P 06       ELSE
1100 0F20-P 06          BEGIN
1101 0F20-P 07            TERMINA.COM.PALAVRA := VERO;
1102 0F24-P 07            (ENDER.ARVORE := CRIA(NO.CLAUSULA.SEM.PARAMETRO))↑FILHO
1103 0F32-P 07                 := PRIM.ARV.PAL;
1104 0F36-P 07            DECORE.CELULA(DE.CLAUSULA,PRIM.ARV.PAL↑ATRIBUTO,
1105 0F40-P 07                                  NUMERO.DE.CELULAS,ITE.CORR);
1106 0F4E-P 07          END;
1107 0F4E-P 06       ENDER.ARVORE↑ATRIBUTO := NUMERO.DE.CELULAS;
1108 0F5A-P 06       NUMERO.DE.CELULAS := ! + 1;
1109 0F5E-P 06       DEIXA.RASTRO(9);
1110 0F66-P 06
1111 0F66-P 06     END CLAUSULA;
```

```
1113 01C3-D 05    & COMANDOS DA ITEM:
1114 01C3-D 05    &              ----
1115 01C3-D 05
1116 01C3-D 05      DEIXA.RASTRO(6);
1117 0E70-P 05
1118 0E70-P 05      IF TOKEN(ABRE.CHAVES) THEN
1119 0E7C-P 05         BEGIN
1120 0E7C-P 06           OBRIGATORIO := VERO;
1121 0E80-P 06           GRUPO;
1122 0E84-P 06           TERMINA.COM.PALAVRA := FALSO;
1123 0E88-P 06           ITEM := VERO;
1124 0E8C-P 06         END
1125 0E8C-P 05      ELSE IF TOKEN(ABRE.COLCHETES) THEN
1126 0E9A-P 05         BEGIN
1127 0E94-P 06           OBRIGATORIO := FALSO;
1128 0E9E-P 06           GRUPO;
1129 0EA2-P 06           TERMINA.COM.PALAVRA := FALSO;
1130 0EA6-P 06           ITEM := VERO;
1131 0EAA-P 06         END
1132 0EAA-P 05      ELSE IF TOKEN(PALAVRA) THEN
1133 0EB8-P 05         BEGIN
1134 0EB8-P 06           CLAUSULA;
1135 0EBC-P 06           ITEM := VERO;
1136 0EC0-P 06         END
1137 0EC0-P 05      ELSE
1138 0EC2-P 05           ITEM := FALSO;
1139 0EC6-P 05
1140 0EC6-P 05      DEIXA.RASTRO(7);
1141 0ECE-P 05
1142 0ECE-P 05    END ITEM;
```

```
1144 01C3-D 04    & COMANDOS DA SINTAXE:
1145 01C3-D 04    &                -------
1146 01C3-D 04
1147 01C3-D 04      DEIXA.RASTRO(4);
1148 0FDE-P 04      SALVA(&VARIAVEIS);
1149 0FE8-P 04      IF NOT ITEM THEN
1150 0FF2-P 04           EXIJA(PALAVRA);
1151 0FF4-P 04      ENDER.ARVORE.ITEM.CORRENTE :=
1152 0FFA-P 04           ENDER.ARVORE.PRIMEIRO.ITEM := ENDER.ARVORE;
1153 1006-P 04      ITEM.CONCATENACAO := FALSO;
1154 100A-P 04      WHILE ITEM DO
1155 1012-P 04         BEGIN
1156 1012-P 05           ENDER.ARVORE.ITEM.CORRENTE :=
1157 1012-P 05                ENDER.ARVORE.ITEM.CORRENTE↑IRMAO := ENDER.ARVORE;
1158 1020-P 05           ITEM.CONCATENACAO := VERO;
1159 1024-P 05         END;
1160 1026-P 04      IF ITEM.CONCATENACAO THEN
1161 102C-P 04           (ENDER.ARVORE := CRIA(NO.CONCATENACAO))↑FILHO :=
1162 103A-P 04                ENDER.ARVORE.PRIMEIRO.ITEM;
1163 103E-P 04      RESTAURA(&VARIAVEIS);
1164 1048-P 04      DEIXA.RASTRO(5);
1165 1050-P 04
1166 1050-P 04    END SINTAXE;
```

```
1168 01C3-D 03 PROCEDURE MARC.PROX (WORD ARVORE);
1169 01C3-D 03    &       ---------
1170 01C3-D 03    &
1171 01C3-D 03    & PREENCHE PONTEIROS PROX DA ARVORE
1172 01C3-D 03    &
1173 01C3-D 03    BEGIN
1174 105A-P 04      BYTE(*)   VARIAVEIS = (6,6:0);
1175 01CC-D 04            WORD ARV        POS VARIAVEIS + 1;
1176 01CC-D 04            WORD FIL.CORR   POS VARIAVEIS + 3;
1177 01CC-D 04            WORD NO.FALSO   POS VARIAVEIS + 5;
1178 01CC-D 04
1179 01CC-D 04    & COMANDOS DA MARC.PROX:
1180 01CC-D 04
1181 01CC-D 04      SALVA(&VARIAVEIS);  ARV := ARVORE;
1182 106C-P 04      FIL.CORR := ARV↑FILHO;
1183 1076-P 04      IF ARV↑TIPO.DO.NO = NO.OPCIONAL    OR
1184 107E-P 04         ARV↑TIPO.DO.NO = NO.OBRIGATORIO THEN
1185 108E-P 04           REPEAT
1186 108E-P 04              BEGIN
1187 108E-P 05                FIL.CORR↑PROXIMO := ARV↑PROXIMO;
1188 109E-P 05                MARC.PROX(FIL.CORR);
1189 10A6-P 05              END
1190 10A8-P 04           UNTIL (FIL.CORR := FIL.CORR↑IRMAO) = NULO
1191 10B2-P 04      ELSE IF ARV↑TIPO.DO.NO = NO.OPCIONAL.REPETITIVO THEN
1192 10C2-P 04           REPEAT
1193 10C2-P 04              BEGIN
1194 10C2-P 05                FIL.CORR↑PROXIMO := ARV;
1195 10CC-P 05                MARC.PROX(FIL.CORR);
1196 10D6-P 05              END
1197 10D6-P 04           UNTIL (FIL.CORR := FIL.CORR↑IRMAO) = NULO
1198 10E0-P 04      ELSE IF ARV↑TIPO.DO.NO = NO.OBRIGATORIO.REPETITIVO OR
1199 10EE-P 04              ARV↑TIPO.DO.NO = NO.BARRA.BARRA                THEN
1200 10FE-P 04         BEGIN
1201 10FE-P 05           (NO.FALSO:=CRIA(NO.OPCIONAL.REPETITIVO))↑FILHO :=
1202 110C-P 05                ARV↑FILHO;
1203 1116-P 05           NO.FALSO↑PROXIMO := ARV↑PROXIMO;
1204 1126-P 05           REPEAT
1205 1126-P 05              BEGIN
1206 1126-P 06                FIL.CORR↑PROXIMO := NO.FALSO;
1207 1130-P 06                MARC.PROX(FIL.CORR);
1208 113A-P 06              END
1209 1134-P 05           UNTIL (FIL.CORR := FIL.CORR↑IRMAO) = NULO;
1210 1148-P 05         END
1211 1148-P 04      ELSE IF ARV↑TIPO.DO.NO = NO.CONCATENACAO OR
1212 1152-P 04              ARV↑TIPO.DO.NO = NO.STEND        OR
1213 1160-P 04              ARV↑TIPO.DO.NO = NO.OPEND        THEN
1214 116E-P 04         BEGIN
1215 116E-P 05           WHILE FIL.CORR↑IRMAO <> NULO DO
1216 1178-P 05              BEGIN
1217 1178-P 06                FIL.CORR↑PROXIMO := FIL.CORR↑IRMAO;
1218 1184-P 06                MARC.PROX(FIL.CORR);
1219 1192-P 06                FIL.CORR := FIL.CORR↑IRMAO;
1220 119C-P 06              END;
1221 119E-P 05           FIL.CORR↑PROXIMO := IF ARV↑TIPO.DO.NO=NO.CONCATENACAO
```

```
1222 11A8-P 05                                  THEN ARV↑PROXIMO      ELSE ARV;
1223 11C4-P 05          MARC.PROX(FIL.CORR);
1224 11CE-P 05        END;
1225 11CE-P 04      RESTAURA(AVARIAVEIS);
1226 11D8-P 04    END MARC.PROX;
```

```
1228 01CC-D 03   & COMANDOS DA DECLARACAO:
1229 01CC-D 03   &               ----------
1230 01CC-D 03
1231 01CC-D 03       DEIXA.RASTRO(2);
1232 11E2-P 03       IF TIPO.DO.TOKEN = PALAVRA THEN
1233 11EA-P 03         BEGIN
1234 11EA-P 04            ESVAZIA.TABELA.DE.CELULAS;
1235 11EE-P 04            NUMERO.DO.TOKEN.DECLARACAO := NUMERO.DO.TOKEN;
1236 11F6-P 04            ITE.CORR := NUMERO.DE.CELULAS := 0;
1237 11FE-P 04            SINTAXE;
1238 1202-P 04            ENDER.ARVORE.DECLARACAO := ENDER.ARVORE;
1239 120A-P 04            IF TOKEN(TSTEND) THEN
1240 1216-P 04                (ENDER.ARVORE.DECLARACAO:=CRIA(NO.STEND))↑FILHO:=ENDER.ARVORE
1241 1224-P 04            ELSE IF TOKEN(TOPEND) THEN
1242 1236-P 04                (ENDER.ARVORE.DECLARACAO:=CRIA(NO.OPEND))↑FILHO:=ENDER.ARVORE
1243 1244-P 04            ELSE IF NOT TERMINA.COM.PALAVRA THEN
1244 1252-P 04              BEGIN
1245 1252-P 05                 ERRO(FALTSIMB);
1246 125C-P 05                 PROCURA(TDEFINE);
1247 1264-P 05              END;
1248 1264-P 04            EXIJA(TDEFINE);
1249 126C-P 04            GRAV.DEF(LMCR:=NORD);
1250 127A-P 04            DECORE.DECLARACAO(NUMERO.DO.TOKEN.DECLARACAO,
1251 127E-P 04                              ENDER.ARVORE.DECLARACAO,LMCR,NUMERO.DE.CELULAS);
1252 1290-P 04            MARC.PROX(ENDER.ARVORE.DECLARACAO);
1253 129A-P 04            DESENHA(ENDER.ARVORE.DECLARACAO);
1254 12A4-P 04            DEFINICAO;
1255 12A8-P 04            EXIJA(TENDMACRO);
1256 12B0-P 04            GRAV.INTER(RETORNA,NUMERO.DE.CELULAS);
1257 12BC-P 04         END
1258 12BC-P 03       ELSE
1259 12BE-P 03         BEGIN
1260 12BE-P 04            ERRO(FALTNOMMCR);
1261 12C8-P 04            PROCURA(TENDMACRO);
1262 12D0-P 04         END;
1263 12D0-P 03       DEIXA.RASTRO(3);
1264 12D8-P 03
1265 12D8-P 03 END DECLARACAO;
```

```
1267 01CC-D 02 & COMANDOS DE PROGRAMA:
1268 01CC-D 02 &              --------
1269 01CC-D 02       DEIXA.RASTRO(0);
1270 12E2-P 02       GRAV.INTER(DESVIA,ENDER.INSTR.TEXTO:=NORD);
1271 12F4-P 02       TEXTO := FALSO;
1272 12F8-P 02       WHILE TOKEN(IMACRO) DO
1273 1304-P 02            DECLARACAO;
1274 130A-P 02       LINHA.INICIAL.DO.TEXTO := LINHAS.COMPILADAS;
1275 1312-P 02       GRAV.DEF(ENDER.INSTR.TEXTO);
1276 131C-P 02       TEXTO := VERO;
1277 1320-P 02       DEFINICAO;
1278 1324-P 02       IF TIPO.DO.TOKEN <> FIM.DE.FONTE THEN
1279 132C-P 02           ERRO(FALTEIMFONT);
1280 1334-P 02       GRAV.INTER(TRANSMITE,NUMERO.DO.TOKEN);
1281 1340-P 02       GRAV.INTER(FINALIZA,0);
1282 134A-P 02       DEIXA.RASTRO(1);
1283 1352-P 02 END PROGRAMA;
1284 01CC-D 01
1285 01CC-D 01 GLOBAL LABEL SEGMM;
1286 01CC-D 01    &          =====
1287 01CC-D 01 SEGMM:
1288 1354-P 01       INI.MCR;
1289 1358-P 01       PREENCHE.CONJUNTOS;
1290 135C-P 01       LE.TOKEN;
1291 1360-P 01       PROGRAMA;
1292 1364-P 01       SEGUNDO.PASSO := VERO;
1293 1368-P 01       ULRELAT1 := ULIST;
1294 1370-P 01       ULIST := ULRELAT2;
1295 1376-P 01       CARSEGM("P");
1296 137E-P 01
1297 137E-P 01 END SEGMM;

** FIM LPS **  ADVERTENCIAS:00     ERROS:00  MEMORIA: 0137E-P + 001CC-D
```

```
ARQUIVO FONTE: FIB      [DP01]   VOLUME=00001    VERSAO=       CODIGO=LPSPPF
EQUIPAMENTO:    C500
PARAMETROS:     E:FIB
OPCOES INICIAIS: LE  NLC LA  S   D0  D1  D2  D3  D4  D5  D6  D7

0001 0001-M 00 ?LPS,"FIB"
0002 0002-M 00 &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
0003 0003-M 00 &
0004 0004-M 00 &              TESTE         DA          LINGUAGEM           LPSE
0005 0005-M 00 &
0006 0006-M 00 &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
0007 0007-M 00
0008 0008-M 00 MACRO : ÇCOM ONDE ÇDCL FIM
0009 0009-M 00   DEFINE
0010 0010-M 00         BEGIN
0011 0011-M 01              ÇDCL
0012 0012-M 01              ÇCOM
0013 0013-M 01         END;
0014 0014-M 00   ENDMACRO
0015 0015-M 00
0016 0016-M 00 MACRO NOME Ç REPRESENTA << UM  NUMERO
0017 0017-M 00                          \ UMA SEQUENCIA DE ÇN <<NUMEROS\CARACTERES>> >> ;
0018 0018-M 00   DEFINE
0019 0019-M 00       << ÇN DEFINE << NUMEROS    DEFINE INTEGER
0020 0020-M 00                     \ CARACTERES DEFINE BYTE   >> (ÇN)
0021 0021-M 00        \                         DEFINE INTEGER        >> ÇNOME;
0022 0022-M 00   ENDMACRO
0023 0023-M 00
0024 0024-M 00 MACRO SEJA Ç, ÇQUEM
0025 0025-M 00 STEND
0026 0026-M 00   DEFINE
0027 0027-M 00       ÇQUEM:=ÇSEJA
0028 0028-M 00   ENDMACRO
0029 0029-M 00
0030 0030-M 00 MACRO PARA ÇV [DESCENDO] DE Ç A Ç,
0031 0031-M 00   DEFINE
0032 0032-M 00       FOR ÇV:=ÇDE <<DESCENDO DEFINE DOWNTO \ DEFINE TO>> ÇA DO
0033 0033-M 00   ENDMACRO
0034 0034-M 00
0035 0035-M 00 MACRO SOMA DO Ç COM  Ç
0036 0036-M 00 OPEND
0037 0037-M 00   DEFINE
0038 0038-M 00       ÇSOMA+ÇCOM
0039 0039-M 00   ENDMACRO
0040 0040-M 00
0041 0041-M 00 MACRO DE ORDEM ÇI
0042 0042-M 00 OPEND
0043 0043-M 00   DEFINE
0044 0044-M 00       (ÇI)
0045 0045-M 00   ENDMACRO
0046 0046-M 00
```

```
0046 0047-M 00 MACRO ESCREVA Ç
0049 0048-M 00 STEND
0050 0049-M 00   DEFINE
0051 0050-M 00      DISPLAY(CESCREVA,10)
0052 0051-M 00   ENDMACRO
0053 0052-M 00
0054 0053-M 00 MACRO EM C, CONVERTA ÇN PARA CARACTERES;
0055 0054-M 00   DEFINE
0056 0055-M 00      BEGIN
0057 0056-M 01         PROCEDURE INTASCII(INTEGER N; WORD E,F,T); EXTERNAL;
0058 0057-M 01         INTASCII(ÇN,ÇEM,A"I10",3)
0059 0058-M 01      END
0060 0059-M 00 ENDMACRO
0061 0060-M 00
0062 0061-M 00 MACRO A DEFINE ENDMACRO
0063 0062-M 00
0064 0063-M 00 MACRO O DEFINE ENDMACRO
0065 0064-M 00
0066 0065-M 00 MACRO IMPRESSAO DO ÇNUM
0067 0066-M 00 STEND
0068 0067-M 00   DEFINE:
0069 0068-M 00         EM RESULT, CONVERTA ÇNUM PARA CARACTERES;
0070 0069-M 00         ESCREVA "FIBONACCI:";        ESCREVA RESULT;
0071 0070-M 00   ONDE  O NOME RESULT REPRESENTA UMA SEQUENCIA DE 10 CARACTERES;
0072 0071-M 00   FIM
0073 0072-M 00   ENDMACRO
0074 0073-M 00
0075 0074-M 00 MACRO CALCULO E IMPRESSAO, DE TRAS PRA FRENTE,
0076 0075-M 00          DOS ÇN PRIMEIROS NUMEROS DE FIBONACCI.
0077 0076-M 00   DEFINE:
0078 0077-M 00         SEJA 0, O NUMERO.DE.FIBONACCI DE ORDEM 0;
0079 0078-M 00         SEJA 1, O NUMERO.DE.FIBONACCI DE ORDEM 1;
0080 0079-M 00         PARA I DE 2 A ÇN-1,
0081 0080-M 00              SEJA
0082 0081-M 00                 A SOMA DO    NUMERO.DE.FIBONACCI DE ORDEM I-1,
0083 0082-M 00                        COM O NUMERO.DE.FIBONACCI DE ORDEM I-2,
0084 0083-M 00                 O NUMERO.DE.FIBONACCI DE ORDEM I;
0085 0084-M 00         PARA I DESCENDO DE ÇN-1 A 0,
0086 0085-M 00              IMPRESSAO DO NUMERO.DE.FIBONACCI DE ORDEM I;
0087 0086-M 00   ONDE  O NOME NUMERO.DE.FIBONACCI REPRESENTA UMA SEQUENCIA DE ÇN NUMEROS;
0088 0087-M 00         O NOME I                     REPRESENTA UM  NUMERO;
0089 0088-M 00   FIM
0090 0089-M 00   ENDMACRO
```

```
0092 0090-M 00       CALCULO E IMPRESSAO, DE TRAS PRA FRENTE,
0093 0091-M 00          DOS 30 PRIMEIROS NUMEROS DE FIBONACCI.

** FIM LPS **  ADVERTENCIAS:00     ERROS:00  MEMORIA: 00036-P + 0008D-D
```

FIBONACCI:

514229
317811
196418
121393
75025
46368
28657
17711
10946
6765
4181
2548
1597
987
610
377
233
144
89
55
34
21
13
8
5
3
2
1
1
0