SIC
Sistema de Implementação de Compiladores

Manual do Usuário Versão 1

por

Mariza Andrade da Silva Bigonha

Roberto da Silva Bigonha

RT 002/86 DCC/ICEx/UFMG

Sinopse

Este manual apresenta a especificação de uma ferramenta, denominada SIC, destinada à assistir a implementação de compiladores através de uma linguagem especial, denominada SIC, que é baseada no Pascal. SIC é dotada de facilidades para a especificação de sintaxe de linguagens, de meios de associar rotinas semânticas às produções de gramáticas e possibilita, sem perda de eficiência, a implementação de compiladores de vários passos onde cada passo opera diretamente no fonte, eliminando a necessidade de linguagem intermediaria. A partir da especificação da sintaxe, o SIC gera tabelas LALR(1) compactadas e um reconhecedor sintático acrescido de um mecanismo de recuperação de erro independente de linguagem que fornece mensagens de erros automaticamente. O Sistema ainda provê facilidades que permitem a resolução "ad- hoc" de conflitos da Tabela LALR(1) e o uso de gramáticas ambiguas na produção de analisadores sintáticos determinísticos.

1 SISTEMA DE IMPLEMENTAÇÃO DE COMPILADORES (SIC)

1.1 Organização do Sistema

O SIC está baseado na linguagem SIC que foi projetada com o objetivo de servir como ferramenta adequada à escrita de compiladores, em particular, de *front-end* de compiladores. Esta linguagem pode ser vista como uma linguagem Pascal estendida, ou seja, ela contém todos os recursos da linguagem Pascal além dos mecanismos considerados importantes na escrita de um compilador. As principais extensões introduzidas são:

1. Os mecanismos para definir a gramática da linguagem a ser implementada juntamente com suas respectivas rotinas semânticas;
2. Os mecanismos para gerar a pilha semântica automaticamente a partir da declaração explícita de atributos [AHO 72] linguagem;
3. As facilidades para agrupar declarações de constantes, rótulos, tipos e variáveis da linguagem Pascal com declarações de procedimentos;
4. As facilidades para declarar explicitamente os terminais da linguagem a ser implementada.
5. Facilidades para a implementação de compiladores de vários passos onde, cada passo opera diretamente no fonte, eliminando a necessidade de linguagem intermediária.
6. Os mecanismos que definem a produção de compiladores interativos ou *batch*.
7. As facilidades que permitem a resolução *ad-hoc* de conflitos da Tabela *LALR(1)* via relação de precedência e associatividade, possibilitando o uso de gramáticas ambíguas na produção de analisadores sintáticos determinísticos [AHO 77].
8. As facilidades de se definir características dependentes de linguagem com o objetivo de se ter recuperação de erro mais precisa.

A maior parte das novas construções introduzidas visam à facilitar a expressão dos mecanismos de compilação. Algumas, entretanto, foram incorporadas a SIC com o objetivo de suprir deficiências do Pascal com relação à programação modular. Por exemplo, é permitido em SIC ao declarar as variáveis globais distribuí-las na parte de declaração do programa de acordo com o seu uso. O mesmo recurso existe para os outros tipos de declarações da linguagem Pascal, permitindo ao projetista do compilador simular módulos, ou seja, pode-se agrupar várias declarações de constantes, tipos, variáveis, rótulos e procedimentos, deixando ao SIC a incumbência de reconhecer essas declarações e dar-lhes a interpretação correta. O funcionamento de todos estes mecanismos será visto em detalhe na Seção 2. A compilação de um programa em SIC é feita em três fases. A primeira fase recebe como entrada um programa em SIC, armazenado no arquivo lógico *ENTRADA*, e a partir daí, produz como saída arquivos em disco contendo respectivamente, declarações Pascal, a gramática em uma forma interna, as tabelas: de símbolos, de produções, de abridores e fechadores de escopo (Seção 2.1.6), de não-terminais (Seção 2.1.7) e

o arquivo de erros (Figura 1). Dentre as operações realizadas nesta fase incluem-se a geração de um arquivo para cada classe de objeto do Pascal, ou seja, o arquivo *CONSTSIC* contém todas as declarações de constantes em *ENTRADA; TYPESIC* contém todas as declarações de tipo; *VARSIC* recebe as declarações de variáveis e *LABELSIC* armazena os rótulos. Os procedimentos declarados em *ENTRADA* são coletados em um arquivo denominado *PROCSIC*. Os arquivos *SEMSIC* e *CORPOSIC* contêm respectivamente o procedimento gerado pelo SIC que define as rotinas semânticas e o corpo do programa, definido em *ENTRADA*.

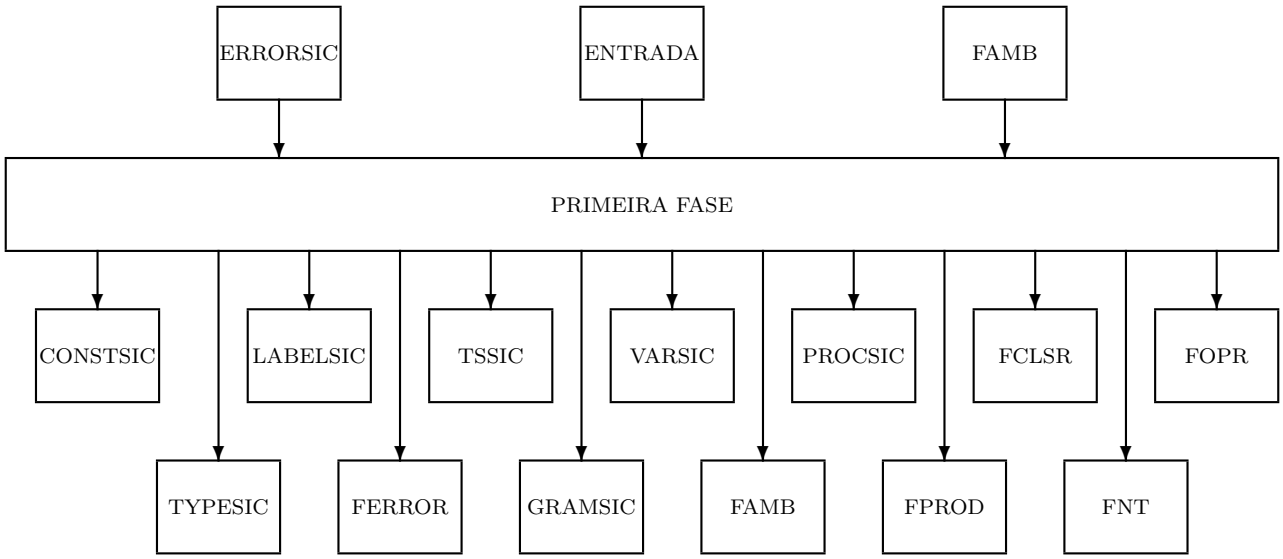


Figura 1: Primeira Fase do SIC

Outras operações realizadas nesta primeira fase dizem respeito à leitura e transformação da gramática para uma forma interna, a geração da regra inicial da gramática e subsequente gravação em *GRAMSIC*, a determinação de não-terminais que são inatingíveis a partir do símbolo inicial da gramática, a determinação de não-terminais que produzem terminais, à geração do arquivo *FPROD* contendo a tabela de produções, a geração de constantes que representam símbolos terminais da linguagem, a incorporação na tabela de símbolos *TSSIC*, das relações de precedência e associatividade associadas aos símbolos terminais da linguagem e às produções, a gravação de informações necessárias as outras fases do SIC no arquivo de comunicação *FAMB*. A segunda fase recebe como entrada o arquivo *FAMB*, as tabelas de produções e de símbolos e a gramática em sua forma interna produzida na fase anterior e armazenadas nos arquivos *FPROD, TSSIC* e *GRAMSIC* respectivamente, e produz como saída o arquivo *FLR* contendo a tabela *LALR(1)* compactada, o arquivo *FLR0* contendo a coleção canônica *LR(0)* e o arquivo *FAMB* atualizado (Figura 2). Esta fase só é executada se houver alterações na tabela símbolos e/ou gramática após a rodada anterior.

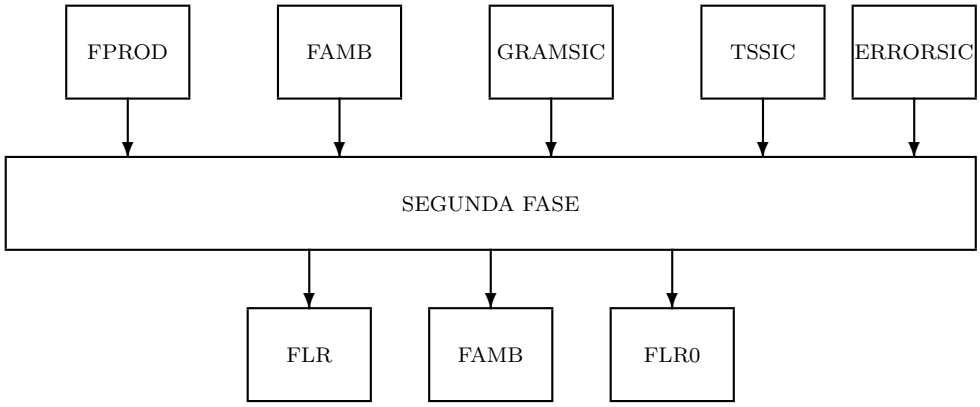


Figura 2: Segunda Fase do SIC

A terceira fase por sua vez, recebe como entrada os seguintes arquivos:

- Os arquivos *PROCSIC* e *SEMSIC* contendo os procedimentos e as rotinas semânticas da primeira fase.
- Os arquivos de declarações gerados nas fases anteriores.
- O arquivo *CORPOSIC* contendo o corpo do programa fonte.
- O procedimento do analisador sintático com ou sem um recuperador automático de erros sintáticos embutido lido do arquivo *PARSERSIC*

Esta fase entre outras coisas é responsável pela atualização do arquivo de constantes, ou seja, inclui neste arquivo declarações de constantes que denotam a dimensão da tabela gerada no passo anterior, bem como endereços dentro desta tabela. O resultado desta fase é um programa completo em Pascal. Este programa gerado é armazenado em arquivos, com extensões, *DCL, PRO, SEM* e *PRS* (Figura 3) da seguinte maneira: os arquivos de declarações são unidos em um único arquivo com a extensão DCL; os arquivos contendo as rotinas semânticas e os procedimentos já estão respectivamente nos arquivos com as extensões *SEM* e *PRO*; os arquivos contendo o analisador sintático e o corpo do programa fonte são unidos gerando assim o arquivo com a extensão

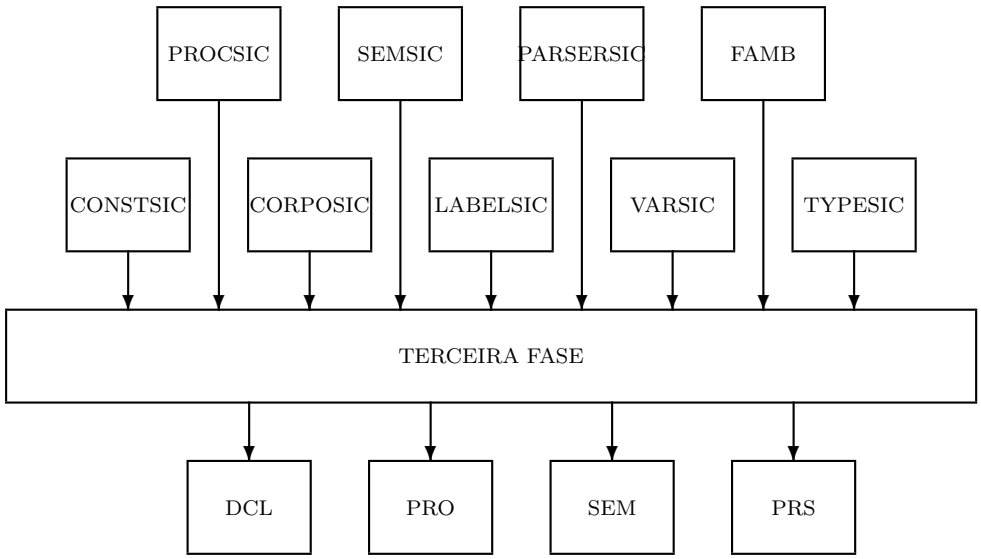


Figura 3: Terceira Fase do SIC

PRS.

Finalmente, para compilar o programa Pascal obtido na terceira fase, é necessário a criação de um arquivo com o nome do compilador fonte e a extensão .PAS. Este arquivo deve conter os arquivos *.DCL, .PRO, .SEM* e *.PRS*, usando o *INCLUDE* do Pascal e acrescentando um ponto ou ponto e virgula ao final, dependendo se o fonte em *.SIC* é um procedimento ou um programa (Veja Seção 2.1.3). Por exemplo, o arquivo *EXEMPLO.PAS* conteria as seguintes informações:

```
(*$I EXEMPLO.DCL *)
(*$I EXEMPLO.PRO *)
(*$I EXEMPLO.SEM *)
(*$I EXEMPLO.PRS *)
.
```

A partir daíusando o *TURBO PASCAL 3.0* produz-se *EXEMPLO.COM* que junto com a tabela *LALR(1) (FLR)* e a tabela de produções *(FPROD)* geradas nas fases dois e um respectivamente, formam o compilador para a linguagem definida em *ENTRADA* (Figura 4).

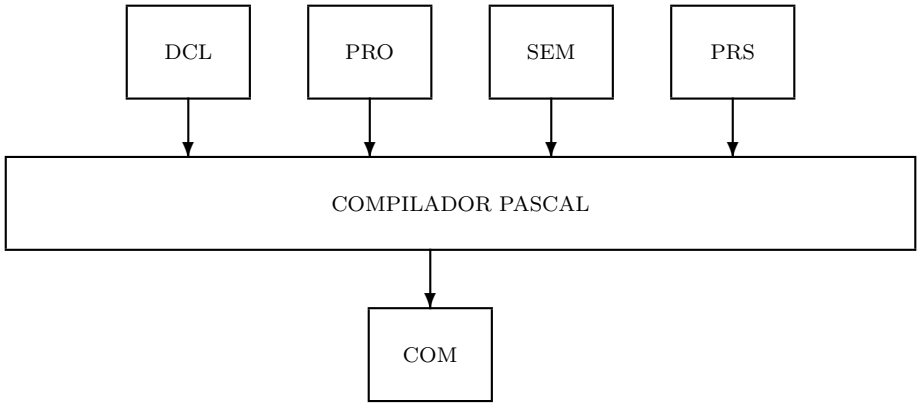


Figura 4: Programa SIC compilado

1.1.1 Interação com o Usuário

O SIC é um sistema para microcomputadores de 16 bits, rodando sob o *sistema operacional MS-DOS*. Para executá-lo, o disquete ou o diretório com o *sistema MS-DOS* deve conter o arquivo *CONFIG.SYS* com o comando

FILES = 26

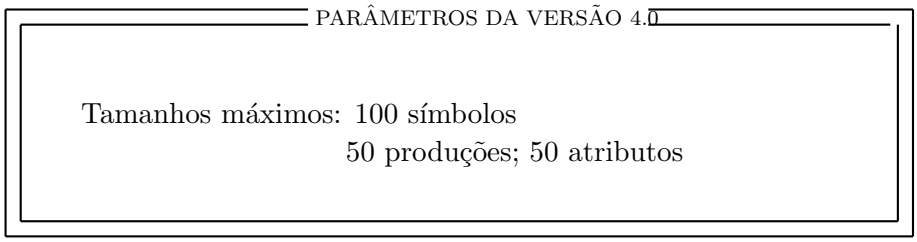
no momento em que se faz a carga do sistema operacional. A ausência deste arquivo pode causar a mensagem *Too many open files* durante a tentativa de execução do SIC. Caso este arquivo exista mas o número de arquivos definidos em *FILE* seja inferior a uma certa constante necessária para a execução do sistema, uma mensagem aparecerá na primeira tela do SIC e a execução do programa pára. Neste momento o usuário deve sair do sistema pressionando a tecla [ESC], atualizar o número de arquivos definido em *FILE* e dar *bootstrap* no sistema antes de rodar o SIC novamente.

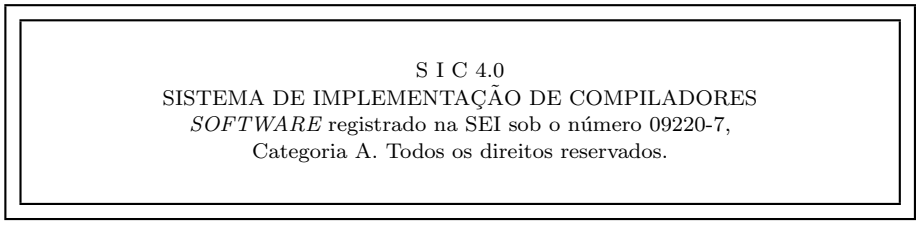
Instalação do SIC
Para se colocar o SIC em operação deve ser colocado no arquivo *AUTOEXEC.BAT* o comando

```
path \SIC
```

Através dele o usuário define o diretório em que conterá o SIC, possibilitando a execução do sistema em qualquer sub-diretório. O arquivo do analisador sintático, *.DAT* pode estar no mesmo diretório em que está o compilador fonte, .SIC ou no diretório SIC definido pelo usuário.

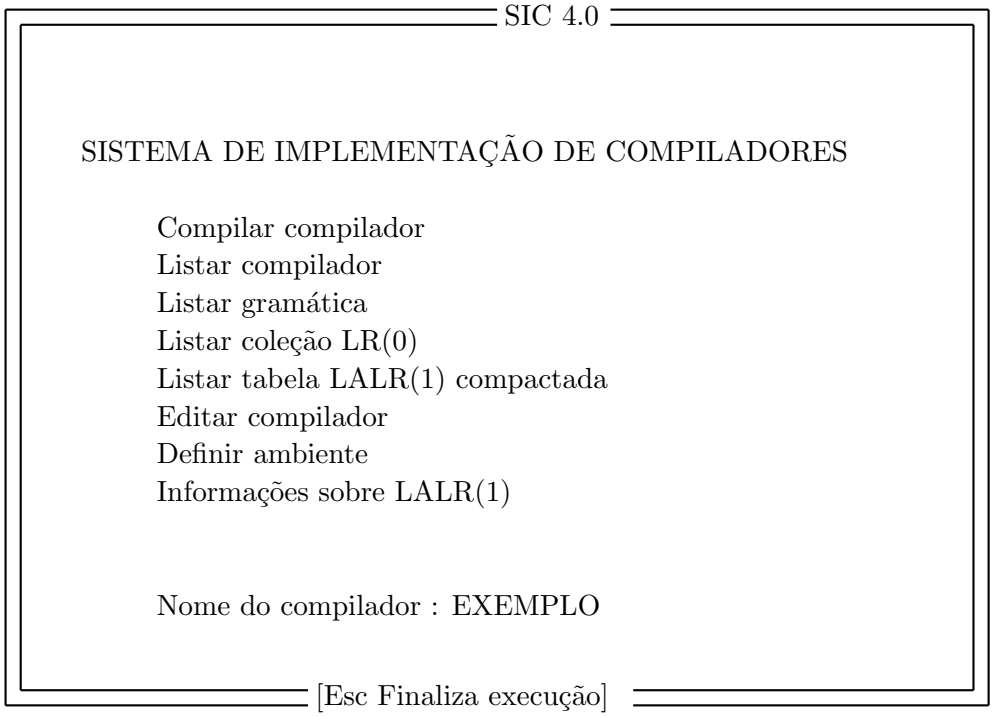
O Sistema primeiro procurará por este arquivo no diretório corrente, não o encontrando ele o procurará no diretório SIC. A ausência dele interromperá a execução da terceira fase do sistema e uma mensagem aparecerá na tela juntamente com o menu com informações relativas aquela execução. O usuário deve então pressionar a tecla [ESC] para retornar ao Menu 2 e sair do sistema para providenciar a cópia do analisador sintático. Note que após a ativação do SIC novamente, o sistema executará somente a primeira e a terceira fase. Para a ativação do sistema, digite a palavra SIC. Após a carga, o SIC interage com o usuário através do Menu 1, solicitando o nome do compilador. Neste ponto, ao presionar a tecla [ESC] a execução é interrumpida.





Menu 1

O usuário deve responder com o nome do arquivo de entrada (até oito caracteres) sem especificação da extensão, a qual, deve obrigatoriamente ser .SIC. Ao processar o nome do compilador, o SIC exibe o Menu 2, e assume que a unidade de entrada do compilador, a unidade que conterá o compilador gerado e a unidade de trabalho seja o *drive* c. A partir daí, o sistema grava estas informações, bem como outras informações pertinentes a execução das fases do SIC no arquivo *FAMB*. As informações referentes ao ambiente serão válidas até que o mesmo seja explicitamente mudado pelo usuário (Veja Menu 3).



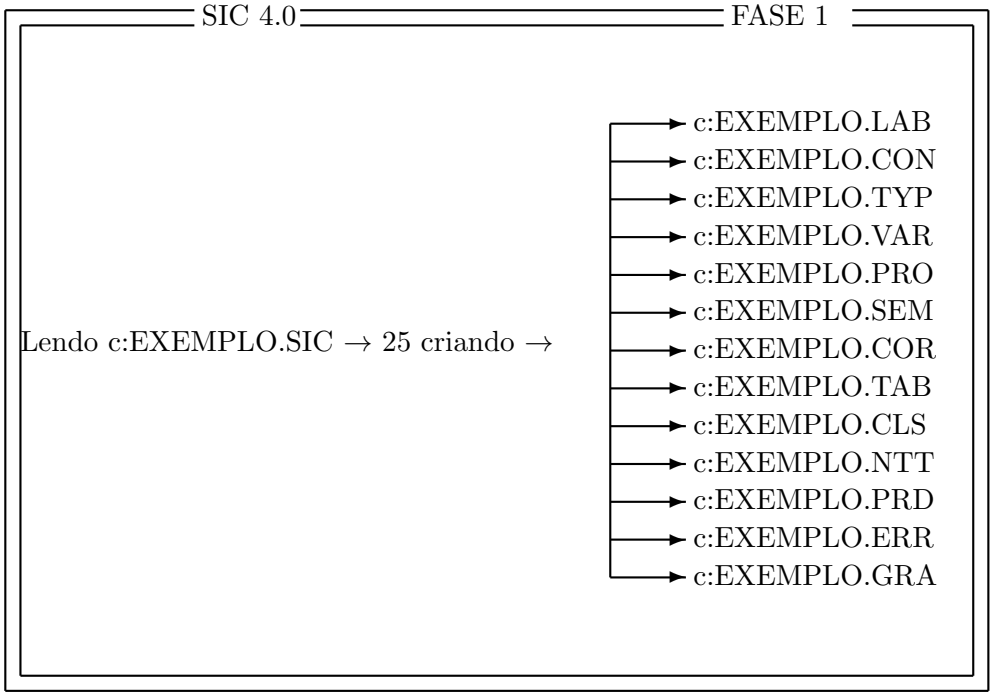
Menu 2

O cursor no Menu 2 fica posicionado na função *Compilar compilador* Para desativar o SIC, o usuário deve presionar a tecla [ESC], à esquerda do teclado. Qualquer função é escolhida presionando-se as setas para-cima ou para-baixo. A seguir descrevemos cada uma das funções exibidas pelo Menu 2.

Compilar compilador

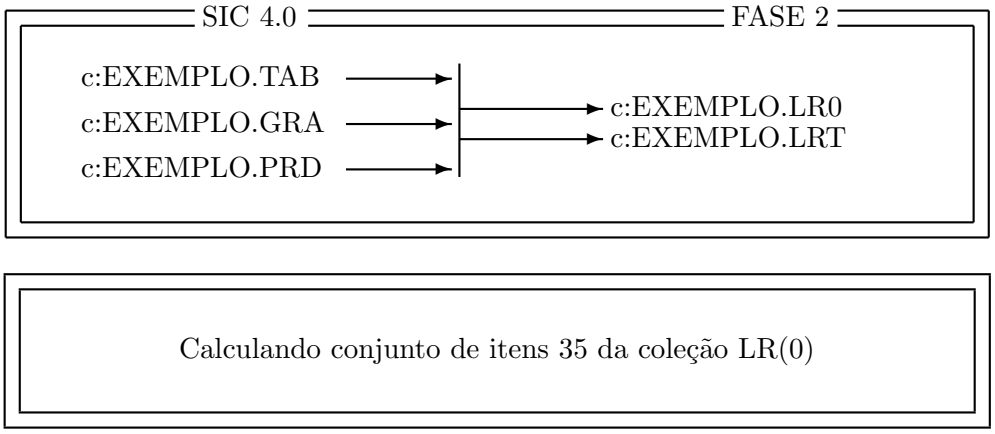
Esta função é responsável pela execução das três fases descritas na Seção 1.1. Há três Menus relacionados com esta função, cada um deles exibe informações referentes à fase em questão acompanhando os estágios de sua execução.

FASE 1:



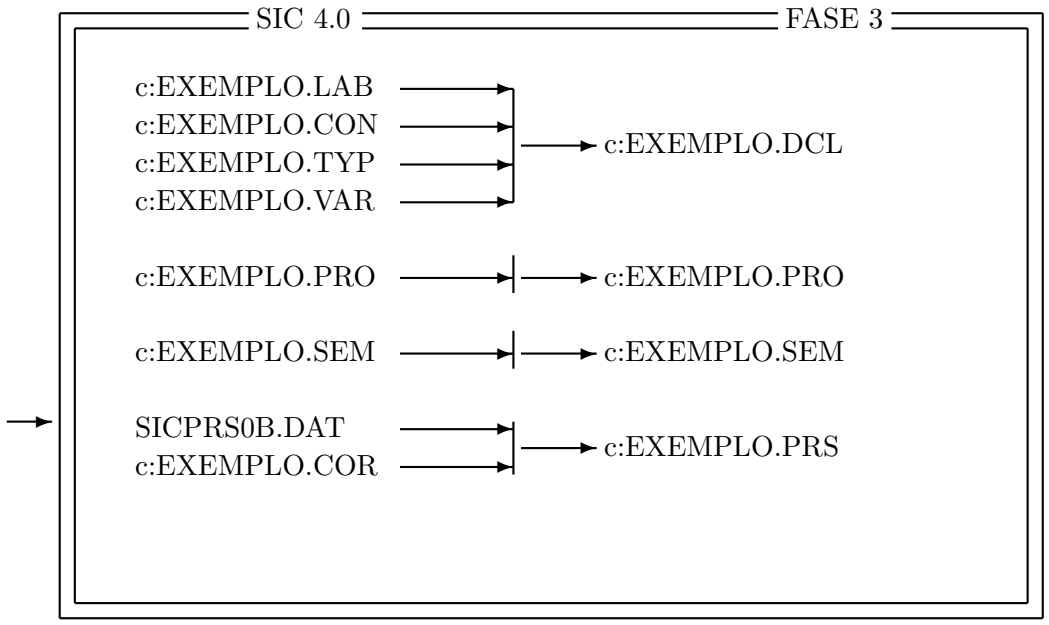
O número que aparece neste menu, 25 neste ponto, se refere a linha que está sendo processada.

FASE 2:

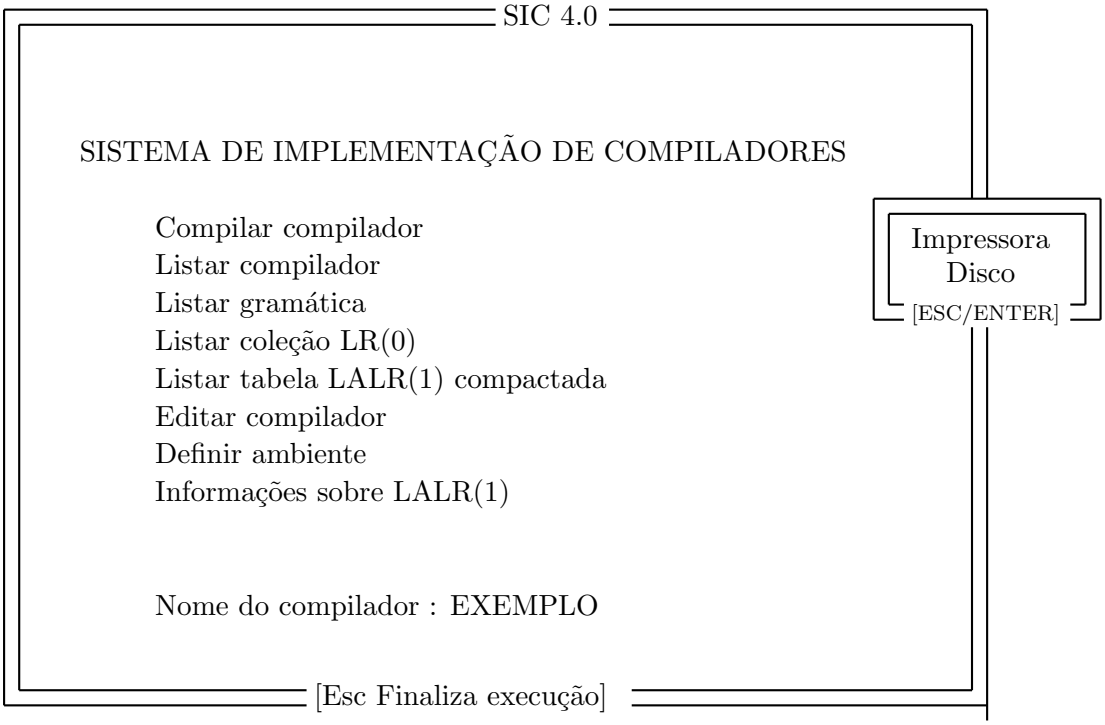


Esta segunda janela da fase 2 mostra em que estágio está a execução da mesma.

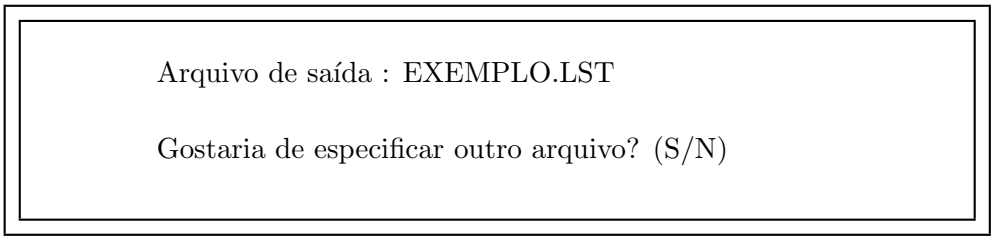
FASE 3:



A seta à esquerda deste menu indica o arquivo que está sendo lido no momento para *EXEMPLO.PRS*. As próximas quatro funções se referem a gravação em disco ou listagem na impressora, do programa fonte, da coleção canôncia *LR(0)*, da tabela *LALR(1)* compactada e da gramática. Todas elas usam o Menu 2. Ao pressionar a tecla [ENTER] para qualquer uma delas uma nova janela aparece no Menu 2, o usuário neste momento deve escolher entre uma das duas opções: *Impressora* ou *Disco*. A função *Disco*, nesta versão ainda não está implementada, para obter uma destas quatro listagens basta ligar a impressora e escolher a opção *impressora* para listá-las.



Ao pressionar a tecla [ENTER] após ter selecionado a função *Disco* o Menu 3 aparecerá na tela especificando o nome do arquivo que conterá uma das quatro funções desejada. Caso o usuário queira mudar este nome ele é livre para fazê-lo. É importante notar que este arquivo como definido tem o mesmo nome e a extensão *LST*. Portanto é necessário mudar pelo menos a extensão se o usuário deseja ter as quatro funções em disco (compilador, gramática, coleção canônica *LR(0)*, tabela *LALR(1))*, uma vez que a gravação de uma delas seguida de outra gravação implica na perda da anterior. Entretanto esta opção ainda não está disponível ao usuário nesta versão por questões de incompatibilidade de versões da linguagem Turbo Pascal.



Menu 3

Listar compilador

Esta função é responsável pela listagem na impressora ou a gravação em disco do programa fonte, com as linhas numeradas para facilitar a indicação de erros, e as mensagens de erros, se houver.

Listar gramática

Esta função é responsável pela gravação em disco ou listagem na impressora, da gramática, com as produções numeradas para facilitar a depuração das tabelas. Além disto ela é responsável pela impressão da tabela de referências cruzadas, assim como pela impressão dos símbolos inacessíveis a partir do símbolo inicial da gramática e dos símbolos não-terminais que não produzem terminais. Ao final uma mensagem informando se a gramática é reduzida ou não é impressa.

Listar coleção LR(0)

Esta função é responsável pela gravação em disco ou listagem na impressora, da coleção canônica LR(0) acompanhada de informações sobre a mesma, como por exemplo, se houve conflitos *SHIFT-REDUCE, REDUCE-REDUCE*, tamanho da coleção canônica *LR(0)*, etc.

Listar tabela LALR compactada

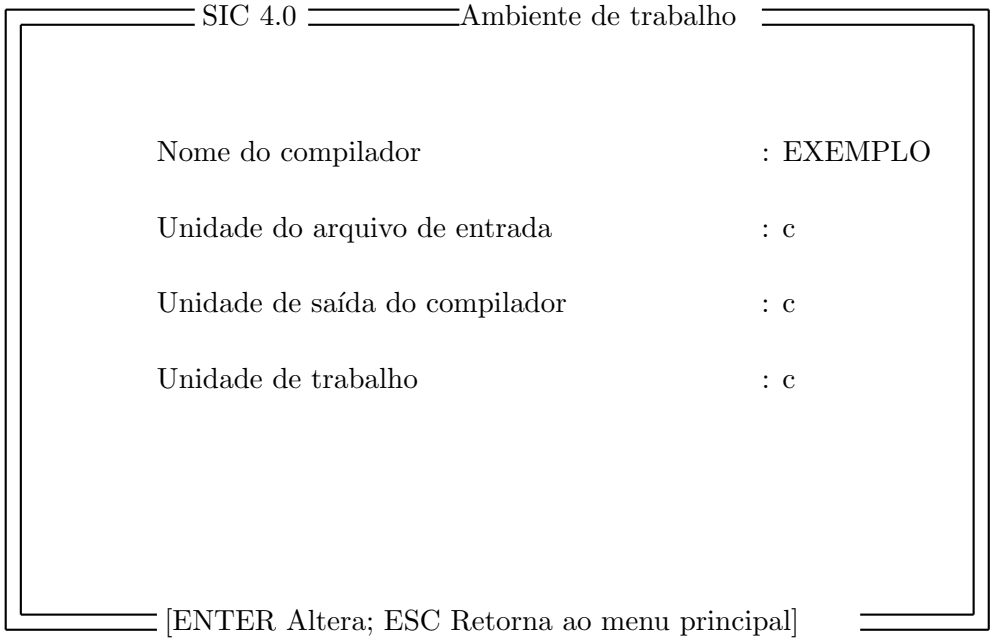
Esta função é responsável pela gravação em disco ou listagem na impressora, da tabela *LALR(1)* assim como informações referentes ao seu tamanho.

Editar compilador

Esta função está implementada, mas ainda não foi incorporada ao SIC, portanto não se pode usá-la.

Definir ambiente

Através da ativação desta função o Sistema permite ao usuário definir um novo ambiente de trabalho, ou seja, ele pode ativar outro compilador, mudar as unidades de entrada, saída e trabalho. Inicialmente o ambiente de trabalho é definido com o nome do compilador fornecido no início da execução do SIC e com as unidades de entrada, saída e trabalho localizadas no *drive* c. Se houver alguma mudança no Menu 4, um novo ambiente é gravado em disco e o sistema passa a considerar o novo ambiente deste momento em diante. (Veja Menus 5 e 6).



Menu 4

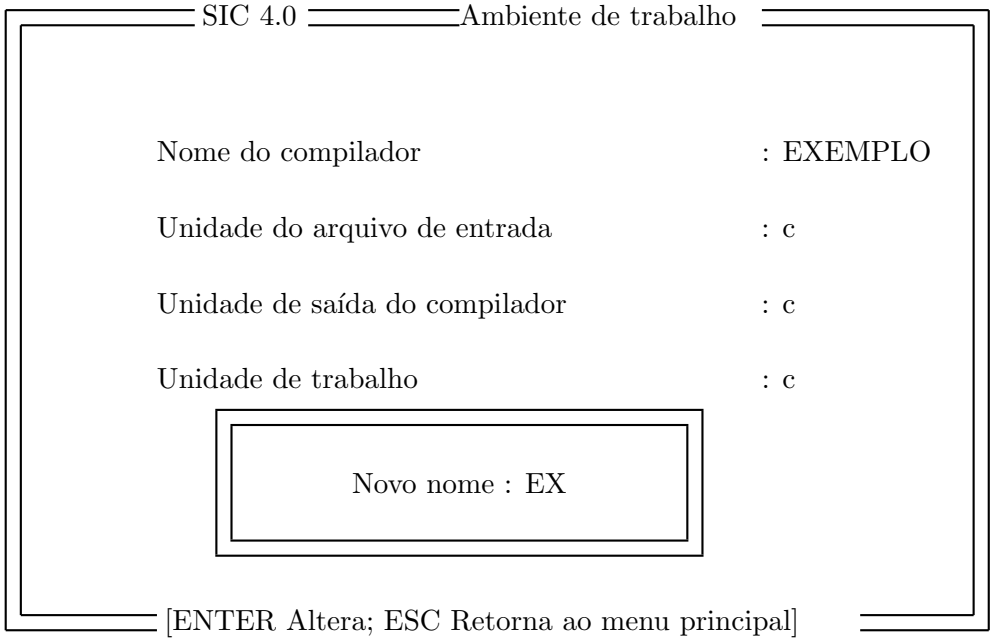
Inicialmente o cursor fica parado na linha

Nome do compilador : EXEMPLO

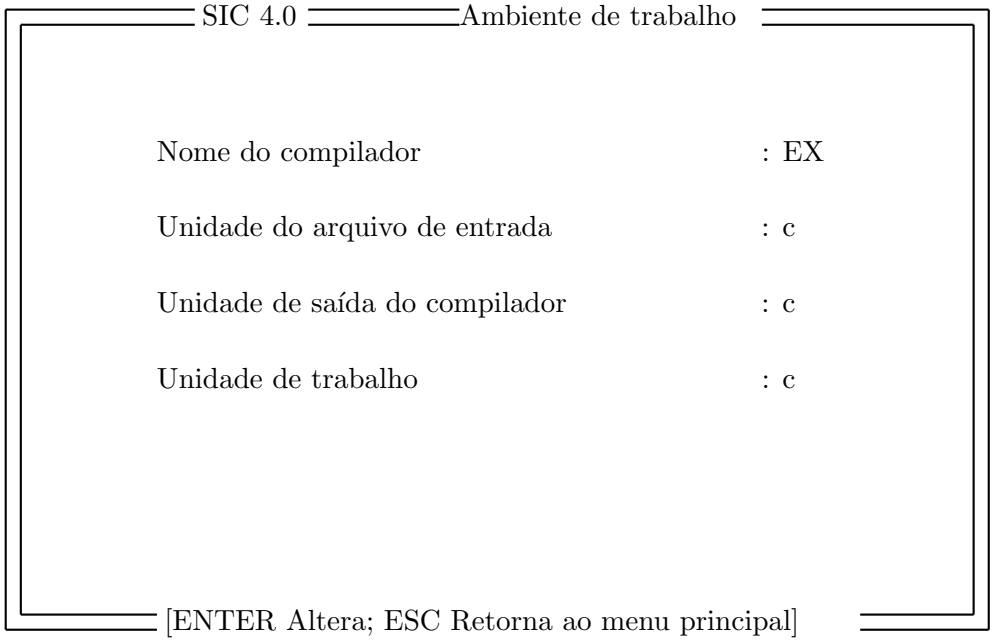
Para mudanças neste menu, procede-se da seguinte maneira:

1. Para mover de uma opção para outra, use as setas, para-cima e para-baixo,
2. Ao selecionar uma opção, precione a tecla [ENTER] e digite a mudança que deseja,
3. Precione a tecla [ENTER] novamente para poder mover para outra opção.
4. Para retornar ao principal, Menu 2, precione a tecla [ESC].

Exemplos:



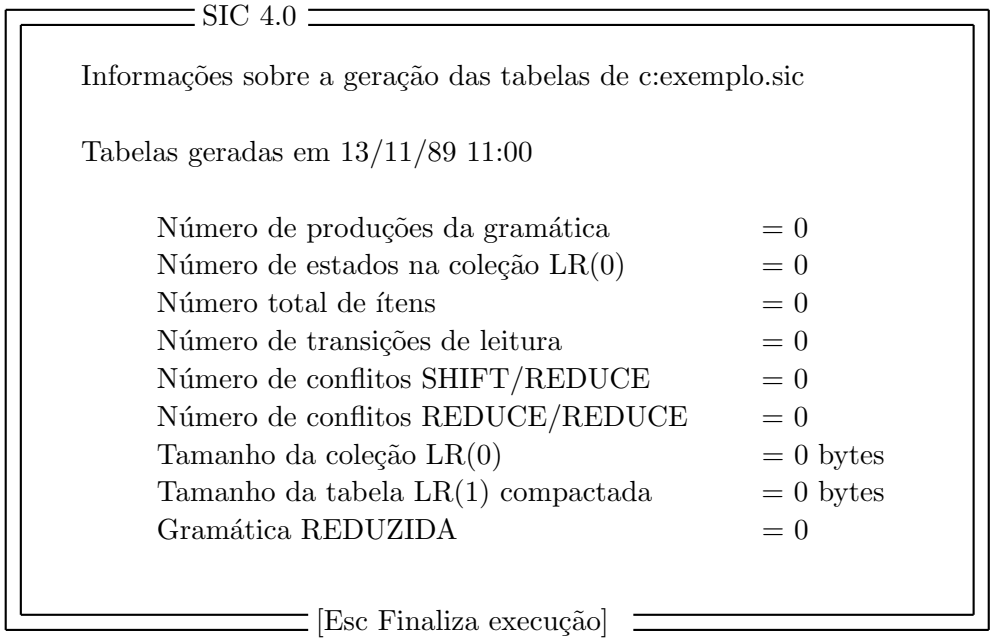
Menu 5



Menu 6

Informações sobre LALR(1)

Esta função fornece informações referentes as tabelas geradas na fase 2 do SIC. O Menu 7 aparece na tela automaticamente após a execução do sistema ou após algum problema. Se o Sistema não foi executado nenhuma vez para o compilador fornecido pelo usuário, este menu contem 0 em suas opções. Esta função pode ser ativada também independente de outra função. Observe que se a gramática do compilador fornecida pelo usuário não for reduzida, outras opções podem aparecer neste Menu, tais como: *Número de conflitos SHIFT/REDUCE ... Número de conflitos REDUCE/REDUCE ... Não-terminais inatingíveis a partir do símbolo de partida = Não-terminais que não produzem strings de símbolos terminais=*



Menu 7

1.1.2 Arquivos do SIC

O nome do arquivo fonte descrito na Seção 1.1.1 é usado na formação dos nomes dos arquivos físicos criados pelo SIC. Se o arquivo fonte for *EXEMPLO.SIC*, os demais arquivos terão os seguintes nomes:

```
NOME LOGICO                NOME FISICO
FAMB                        EXEMPLO.AMB
ENTRADA                     EXEMPLO.SIC
ERRORSIC                    EXEMPLO.ERR
CONSTSIC                    EXEMPLO.CON
LABELSIC                    EXEMPLO.LAB
TYPESIC                     EXEMPLO.TYP
VARSIC                      EXEMPLO.VAR
PROCSIC                     EXEMPLO.PRO
SEMSIC                      EXEMPLO.SEM
CORPOSIC                    EXEMPLO.COR
GRAMSIC                     EXEMPLO.GRA
USELSSNT                    EXEMPLO.SNT
GRAMOCUR                    EXEMPLO.OCR
TSSIC                       EXEMPLO.TAB
OBJ                         EXEMPLO.DCL
                            EXEMPLO.PRS
YYSAIDA                     EXEMPLO.LRO
                            EXEMPLO.LST
YYFLR                       EXEMPLO.LRT
YYFPROD                     EXEMPLO.PRD
YYFOPR                      EXEMPLO.OPR
YYFCLSR                     EXEMPLO.CLS
YYFNT                       EXEMPLO.NTT
```

O arquivo *EXEMPLO.AMB* contém informações necessárias para comunicação durante a execução do SIC, assim, cada função do Menu 2, (Seção 1.1.1) pode ser ativada a qualquer momento. O arquivo *EXEMPLO.ERR* contém as mensagens de erro do compilador de SIC. No final de cada execução, somente os arquivos de extensão . AMB, *.DCL, .PRO, .SEM, .PRS, .ERR, .SIC, .TAB, .GRA, LRT, PRD, OPR, CLS, NTT* e *DAT* devem ser preservados. Os demais são apagados.

2 LINGUAGEM DE PROGRAMAÇÃO DE COMPILADORES (SIC)

2.1 Definição do SIC

2.1.1 Notação

Para descrever a sintaxe de SIC é usado o seguinte formalismo: Os não-terminais da gramática são escritos em letras maiúsculas ou minúsculas e os símbolos terminais sempre colocados entre aspas. Cada produção tem a forma

S = E ;

onde S denota um símbolo não-terminal e E as alternativas que definem S. O termo E tem a forma

```
T1 | T2 | ... | Tn      (n > 0 )
```

onde cada termo Ti, $0 < i \leq n$, tem a forma

```
F1  F2  ...  Fm          ( m > 0 )
```

onde cada elemento Fi, $0 < i \leq m$, pode ser:

1. um símbolo terminal
2. um símbolo não-terminal
3. { T }
4. [T]
5. (E)

Uma sequência de F's denota a concatenação dos F's envolvidos. { T } indica sequência de zero ou mais T's. [T] representa um termo T que pode ser omitido. (E) representa um agrupamento de alternativas.

2.1.2 Símbolos Básicos

Os símbolos básicos de SIC são: símbolos terminais (token), símbolos não-terminais (nt), índices, texto, identificadores e palavras-chave.

```
simbolos_basicos  = token
                  |  nt
                  |  indice
                  |  texto
                  |  identificador
                  |  palavra_chave ;
token              =  "sequencia  de  caracteres  diferente  de
                        aspas";
nt                = identificador ;
indice            = " [ "   numero  " ] "  "." ;
```

Um índice é utilizado para se ter acesso a uma ocorrência específica de um símbolo terminal ou não-terminal em uma produção (veja Seção 2.1.9). Texto são sequências de construções do Pascal. Identificadores de SIC são sequências de letras e/ou dígitos iniciados sempre por uma letra. *Underscores* também podem ser usados na formação de identificadores. Identificadores podem ter qualquer tamanho; entretanto, só os quinze primeiros caracteres serão considerados, truncando-se o restante.

```
identificador  = letra   caracter_de_id  ;
caracter_de_id = letra
               | digito
               | "_" ;
digito         =   "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"
               |   "8" | "9" ;
letra          =    "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H"
               |    "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P"
               |    "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "Y"
               |    "X" | "Z" | "a" | "b" | "c" | "d" | "e" | "f"
               |    "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n"
               |    "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v"
               |    "w" | "x" | "y" | "z" ;
```

Os seguintes identificadores são reservados e têm significados pré-definidos em SIC:

YYEOF
: Constante do tipo *INTEGER* representando o código da marca de fim de arquivo (veja Seção 2.1.4).

YYSAIDA
: Arquivo do tipo *text*, declarado pelo SIC, no qual são gravados o programa fonte, as mensagens de erros sintáticos gerados pelo SIC, a coleção canônica *LR(0)*, a tabela compactada *LALR(1)* e a gramática. O usuário é responsável pela abertura deste arquivo.

YYSIMB
: Variável global do tipo *INTEGER* gerada e declarada pelo SIC, devendo ser usada para retornar o tipo do símbolo reconhecido pelo analisador léxico. Funciona como elo de comunicação entre o analisador léxico *YYSCAN* escrito pelo usuário e o analisador sintático *YYPARSER* gerado pelo SIC.

YYPOS
: Variável global do tipo *0..255* gerada e declarada pelo SIC servindo para retornar a posição do último símbolo reconhecido pela *YYSCAN*. Inicialmente *YYPOS = 0*. Pode ser usado como atributo (de leitura) de qualquer símbolo da gramática (veja Seção 2.1.5).

YYLINHA
: Variável global do tipo *INTEGER* gerada e declarada pelo SIC servindo para informar ao *YYPARSER* o número corrente da linha que contém o último *token* lido por YYSCAN. Inicialmente *YYLINHA = 0*. Pode ser usado como atributo (de leitura) de qualquer símbolo da gramática.

YYSCAN
Nome do analisador léxico do compilador a ser projetado, que deve ser escrito pelo usuário. Retorna valores ao analisador sintático através das variáveis *YYSIMB, YYLINHA, YYPOS* e de atributos de terminais (veja Seção 2.1.8).

YYPARSER
Nome do procedimento de análise sintática. O projetista deve invocar este procedimento explicitamente no corpo de seu programa fonte (veja Seção 2.1.11).

YYSEM
Procedimento gerado pelo SIC para conter as ações semânticas especificadas pelo projetista do compilador.

YYRESULTADO
Variável global, cujo tipo é a enumeração *(YYSINTAXEOK, YYERROFATAL, YYERROSINTAXE)*, usada para informar ao usuário o resultado da análise sintática. Seu valor pode ser testado após o retorno da chamada do procedimento *YYPARSER*.

YYSINTAXEOK
Constante que indica a ausência de erros de sintaxe.

YYERROFATAL
Constante que indica que houve estouro da pilha sintática interrompendo assim a análise.

YYERROSINTAXE
Constante que indica que houve erros de sintaxe durante a fase de compilação.

Os seguintes identificadores têm funções específicas no compilador gerado pelo SIC devendo o usuário evitar modificar seus valores de forma a prevenir-se contra resultados imprevisíveis.

```
YYTSMAX           YYLPS        YYTATR       YYLRSTATE
YYMAXTERM         YYPOSPS      YY500Y       YYINDICEDAPILHA
YYLASTOKEN        YYPSEM       YYTAMATR     YYLR
YYTABTERM         YYTOPO       YYPROD       YYE
YYTABNT           YYMAXPILHA   YYNPROD      YYR
YYMONTASIMBOLOS   YYV          YYTEMP       YYMAXOPENER
YYMAXCLOSER       YYMAXNT      YYINDPROD    YYSCOPE
YYPRODUCTIONS     YYBUFFER     YYFCLSR      YYFOPR
YYFNT             YYFLR        YYFPROD
```

Além das palavras-chave da linguagem Pascal [JENSEN 74], os seguintes símbolos são reservados em SIC:

```
palavra_chave =  "%%BATCH"     |  "%%INTERACTIVE" |   "%%PROGRAM"
               |  "%%COMPILER" | "%%LABELS"       |  "%%RIGHT"
               | "%%CONFLICTS" | "%%LEFT"         |  "%%SCOPEMAP"
               | "%%CONSTANTS" | "%%NONE"         |  "%%STACK"
               | "%%END"       | "%%OTHER"        |  "%%TOKENS"
```

```
| "%%FIRST"     | "%%PREC"          |  "%%TYPES"
| "%%GRAMMAR"   | "%%PROCEDURES"    |  "%%VARIABLES"
| "%%NTMAP" ;
```

Palavras-chave podem ser escritas em letras maiúsculas e minúsculas. As palavras *PASS, REC, NOREC, OF, ATRIBUTTES, AND* e *SEMANTICS* não são reservadas, porém, têm significado especial em SIC quando usadas em determinados contextos como será visto a seguir. Os delimitadores simples em SIC são os seguintes caracteres especiais e pares de caracteres.

( ) { } ; , . : = |

Brancos e caracteres de controle de impressão, atuam como delimitadores. Comentários são iniciados com o símbolo (* e terminam com o símbolo *), como no Pascal. Dentro de texto Pascal, os comentários obedecem a regra do Pascal, ou seja, podem aparecer entre os pares { e } ou (* e *). Contudo, fora deste contexto, somente o par (* *) é permitido. Números que ocorrem fora de texto Pascal não possuem sinal, não podem conter brancos entremeados e são sempre do tipo inteiro. numero = digito { digito } ;

2.1.3 Programa em SIC

A especificação de um programa em SIC é a seguinte:

```
SIC   =  "%%COMPILER"   [texto]   opcoes  [passos]
                secao_um
                [secao_dois]
                corpo
         "%%END"
```

O texto após o a palavra chave %%COMPILER deve necessariamente conter a palavra *PROGRAM* ou *PROCEDURE* seguido do nome do programa. *PROGRAM* indica que o compilador desejado será um programa principal em *PASCAL*, enquanto *PROCEDURE* indica que o compilador desejado será um procedimento. As vezes é conveniente declará-lo como procedimento, pois quando o programa se torna muito grande, é mais fácil depurá-lo isoladamente. Note que a sintaxe de um programa em SIC não inclui o "." ou o ";", devendo o mesmo ser colocado pelo usuário no final do programa em SIC. O texto estabelece ainda as diretivas de compilação ou especificação de arquivos de entrada e saída dependendo do compilador usado na implementação.

```
Exemplo:
%%COMPILER PROGRAM teste %%BATCH
%%COMPILER PROCEDURE teste %%BATCH
```

O Apêndice A apresenta um exemplo completo de um programa em SIC.

Opções de Compilação
A cláusula opções serve para especificar o tipo de processamento *batch* ou *interativo* a ser usado no compilador.

```
opcoes  =  "%%INTERACTIVE" "NOREC"
        |  "%%INTERACTIVE" "REC"
        |  "%%BATCH"
        |  "%%INCREMENTAL" ;
```

A palavra *REC* após *%%INTERACTIVE* indica ao SIC que o analisador sintático a ser gerado deve tentar identificar todas as correções possíveis que permitiriam a análise sintática prosseguir. Com esta opção, as mensagens de erros de sintaxe serão certamente bastante elucidativas porque todas as ações necessárias a uma recuperação automática do erro serão efetuadas. Além disso, o usuário tem a opção de ignorar o erro e prosseguir a compilação. Por outro lado, o uso de *NOREC* após *%%INTERACTIVE*, indica que a posição de cada erro de sintaxe deverá ser apenas indicada sem que se tente determinar a real causa do erro. Neste caso, a compilação só poderá continuar após o erro ter sido eliminado. Em ambos os casos, após a indicação do erro, o controle da execução passa ao editor de texto (ainda não implementado) de forma a permitir ao usuário efetuar as correções necessárias. Com a opção *%%BATCH*, a recuperação automática de erro é sempre tentada, emitindo–se as respectivas mensagens sem que se interrompa a compilação. A opção *%%INCREMENTAL* indica que deve-se gerar compilador incremental interativo com editor de texto. Esta opção está pronta, faltando apenas incorporá-la ao SIC para torná-la operacional.

Passos do Compilador
A cláusula passos serve para informar ao SIC o número de passos que o compilador terá. A presença da palavra chave *%%FIRST PASS* no programa fonte indica que o compilador será de vários passos e que este é o primeiro deles. O uso da palavra chave *%%OTHER PASS* indica os demais passos do compilador. A omissão de *%%FIRST PASS* e *%%OTHER PASS* implica em um compilador de um único passo.

```
passos  =   "%%FIRST"  "PASS"
        |   "%%OTHER"  "PASS"  ;
Exemplos:
%%COMPILER PROGRAM exemplo; (*$R+*) %%BATCH
%%COMPILER PROCEDURE exemplo; %%INTERACTIVE  NOREC
%%COMPILER PROGRAM exemplo; (*$R+*) %%BATCH %%FIRST PASS
%%COMPILER PROGRAM exemplo; (*$R+*) %%INTERACTIVE REC %%FIRST PASS
%%COMPILER PROGRAM exemplo; (*$R+*) %%BATCH %%OTHER PASS
%%COMPILER exemplo (*$R+*) %%OTHER PASS
```

Cláusulas indicativas de opções usadas em conjunção com *PASS* são ignoradas porque após o primeiro passo somente a modalidade *BATCH* é permitida.

Seção_um
A seção seção_um serve para especificar os símbolos terminais da linguagem sendo implementada; a declaração dos atributos associados aos símbolos não-terminais e terminais da gramática dessa linguagem; a declaração dos mapas de escopo, isto é, dos símbolos terminais que atuarão como abridores de escopo e seus respectivos fechadores (veja Seção 2.1.6); a declaração dos símbolos não-terminais que poderão ser inseridos ou substituídos na recuperação de erro e declarações da linguagem Pascal.

```
secao_um   =  terminais  |  [atributos]  |  [mapadeescopo]
           |  [ntmapa]   |   [declaracoes]   ;
```

As declarações de terminais, de não-terminais, de mapas de escopo e dos atributos só podem aparecer uma vez em cada módulo de compilação. Declarações da linguagem Pascal pode aparecer várias vezes e em qualquer ordem. A ordem dos cinco elementos que formam a seção_um é irrelevante. Entretanto, se símbolos terminais forem usados em declarações da linguagem Pascal, a definição desses terminais deve aparecer antes, para diferenciar texto escrito entre aspas de símbolos terminais (veja seção 2.1.8.

Seção_dois
A seção seção_dois serve para especificar as regras da gramática e resolver conflitos.

```
secao_dois =   gramatica      resolucao_de_conflitos ;
```

2.1.4 Seção de Terminais

A finalidade desta seção é estabelecer um mecanismo de comunicação entre os analisadores sintáticos gerados pelo SIC *YYPARSER* e o analisador léxico *YYSCAN* projetado pelo usuário.

```
terminais  = "%%TOKENS"  token  "="  identificador    [";"]  ;
```

O usuário deve usar esta seção para associar a cada *token* um identificador que será tratado pelo SIC como uma constante que especifica o tipo léxico do símbolo. Dentro do analisador léxico o usuário deverá usar estes identificadores para definir os valores retornados em *YYSIMB*. O analisador sintático somente reconhece os *tokens* listados nesta seção, referindo-se a eles através dos identificadores das constantes que definem os seus tipos léxicos. É sempre obrigatória a presença de um símbolo terminal associado ao identificaor *YYEOF*, que dá o tipo léxico da marca de final do texto de entrada.

```
Exemplo:
%%TOKENS
      "id"    = xid;
      "cte"   = xcte;
```

```
"begin" = xbegin;
"eof"   = YYEOF;
"<"     = xmenor;
">"     = xmaior;
"*"     = xvezes;
"+"     = xmais;
";"     = xpv;
"end"   = xend;
"("     = xap;
")"     = xfp;
"if"    = xif;
```

2.1.5 Seção de Atributos

O SIC implementa o esquema clássico de geração pós-fixada de código dirigida por sintaxe [AHO 72]. Neste método, aos símbolos da gramática devem ser associados atributos sintetizados [KNUTH 68] os quais têm seus valores definidos na medida que a análise sintática é efetuada. Estes atributos correspondem a posições de uma pilha auxiliar, a pilha semântica, alocada ao lado da pilha sintática de estados do reconhecedor LR(1). Desta forma, os valores numa dada posição da pilha semântica denotam os atributos do símbolo representado pelo estado na posição correspondente da pilha sintática. A seção de atributos tem por finalidade prover informações necessárias à geração da pilha semântica. Identifica esta seção a palavra chave *%%STACK*. O número após *%%STACK* refere-se ao tamanho das pilhas sintática e semântica. A omissão desse número implica na geração pelo SIC de um tamanho *default*. Após as palavras *OF ATTRIBUTES* são especificados os atributos para cada símbolo terminal e não-terminal da gramática e as respectivas ações de inicialização de atributos.

```
atributos      =  "%%STACK"    [numero]    "OF"    "ATTRIBUTES"
                  atributo   ";"  ;
atributo       =  def_atrib      { def_atrib }   ;
def_atrib      =  simbolo  "="   "("  lista_de_atrib  ")"
                  [ decl_acao ]   [ ";" ];
lista_de_atrib =  decl_de_atrib    { ";" decl_de_atrib } ;
decl_de_atrib  =  nomes_de_atrib   ":"   nome_do_tipo ;
nomes_de_atrib =  identificador   { "," identificador }
nome_do_tipo   =  identificador ;
decl_acao      = " { "  [ texto ]  " } " ;
simbolos    =  nt
               |  token ;
Exemplos
%%STACK 200 OF ATTRIBUTES
exp    =  (R,tipo : integer)    { exp.r := 0; exp.tipo := 0 };
"id"   =  (valor  : integer)    { GERATEMPORARIO(t);
                                  "id".valor := INSTALA(T) };
"cte"  =  (valor  : integer)    { "cte".valor := 0 };
"*"    =  (P      : alfa);
```

```
cmd    =  (B,XX,K : integer);
"+"    =  (KK,ZZ  : alfa)
dcl    =  (DD     : integer; A,B,C : boolean)
cond   =  (quad   : integer)      { cond.quad := 0 };
```

Ações de inicialização de atributos somente são executadas quando os símbolos da gramática forem inseridos no programa fonte como resultado de ação de recuperação de erro. A ação de inicialização permite ao usuário assegurar que a pilha semântica sempre contém valores bem definidos. A ausência de inicialização de atributos pode causar erros de execução no compilador gerado quando hajam inserção ou substituição de símbolos, terminais ou não-terminais, durante a fase de recuperação. Com este mecanismo de inicialização, as rotinas semânticas do usuário podem ser ativadas nor malmente, mesmo após erros de sintaxe haverem sido detectados durante a compilação.

Todo símbolo, terminal ou não-terminal possui implícitamente os atributos *YYLINHA* e *YYPOS* que denotam a posição inicial do símbolo no arquivo de entrada. Estes atributos não podem ser modificados.

2.1.6 Seção de Abridores e Fechadores de Escopo

A seção mapadeescopo tem por finalidade prover informações úteis à recuperação de escopo. Escopo é definido como sendo construções sintaticamente aninhadas, tais como, procedimentos, blocos, estruturas de controle e expressões parentetizadas. Abridores e fechadores de escopo são pares de símbolos que iniciam e terminam, respectivamente, estas construções. Por exemplo, os pares, *(-)*, *begin - end*, *if - end if*, são delimitadores típicos de escopo. Nesta seção o usuário especifica quais são os símbolos terminais da linguagem a ser implementada que atuarão como abridores e fechadores de escopo. Ressalte-se que, somente para os abridores de escopo declarados nessa seção a recuperação de escopo é tentada.

```
mapadeescopo = "%%SCOPEMAP"  def_scopemap  { def_scopemap };
def_scopemap = "token"   ":"  listacloser  ";"  ;
listacloser  = dclcloser  { ","   dclcloser };
dclcloser    = "token"  { "token" };
Exemplo:
%%SCOPEMAP
     "begin"   :  "end" ;
     "("       :  ")"   ;
     "if"      :  "end" "if"  ,  "end" ;
     "record"  :  "end" "record" ;
     "while"   :  "do";
```

2.1.7 Seção Mapa de Não-terminais

A seção mapa de não-terminais permite a declaração dos símbolos não-terminais da gramática que poderão ser usados durante a recuperação de erros como candidatos à inserção e substituição de um símbolo.

```
ntmapa  = "%%NTMAP" listant ";" ;
listant = nt  { ","  nt };
Exemplo:
%%NTMAP
   exp, cmd, dcl ;
```

Recomenda-se listar nesta seção somente os não-terminais que seja do conhecimento do usuário.

2.1.8 Seção de Declarações

A seção de declaração permite a declaração de rótulos, tipos, constantes, variáveis e procedimentos Pascal em SIC. Estas declarações podem aparecer no texto várias vezes e em qualquer ordem.

```
declaracoes   =   membros   ;
membros  =  "%%LABELS"      declaracao de rotulos
         |  "%%TYPES"       declaracao de tipos
         |  "%%CONSTANTS"   declaracao de constantes
         |  "%%VARIABLES"   declaracao de variaveis
         |  "%%PROCEDURES"  declaracao de procedimentos  ;
declaracao de rotulos      = texto ;
declaracao de tipos        = texto ;
declaracao de constantes   = texto ;
declaracao de variaveis    = texto ;
declaracao de procedimentos= texto  ;
Exemplos
%%CONSTANTS
       TMID  = 8;
       A     = 2;
       B     = 3;
%%LABELS
       1,2;
%%VARIABLES
       x : alpha;   y : integer;
%%TYPES
       alpha = array [1..TMID] of char;
%%PROCEDURES
       procedure YYSCAN;
```

```
        begin
            ...
        end;
        function INSTALA( ... );
        begin
            ...
        end;
        function PROCURA( ... );
        begin
            ...
        end;
%%VARIABLES
        ch      : char;
```

Se o programa em SIC contiver a opção de único passo ou de primeiro passo, é obrigatória a declaração do procedimento *YYSCAN*, responsável pela análise léxica. *YYSCAN* é um procedimento sem parâmetros que é chamado por YYPARSER, o analisador sintático, sempre que o próximo *token* no fluxo de entrada tiver que ser obtido. A cada chamada, *YYSCAN* deve retornar em *YYSIMB* o tipo do *token* lido; em *YYLINHA* o número da linha fonte que contém o *token* e em *YYPOS* a posição do *token* dentro da linha. Os tipos de terminais devem necessariamente ser aqueles especificados na seção de terminais (veja Seção 2.1.4 de forma a permitir a interpretação correta de valores em *YYSIMB* pelo analisador sintático. Caso o *token* possua um valor, este deverá ser atribuído a um de seus atributos. Por exemplo, se o *token cte* tiver um atributo valor do tipo *INTEGER, YYSCAN* poderá retornar o valor *inum* de uma constante inteira lida mediante a atribuição

```
                    "cte".valor := inum ;
```

Exemplo:

```
PROCEDURE GETCHAR(var c:char);
begin
    if cc = ll then begin
                  ...
         ll := 0; cc:= 0;
         while not eoln(FONTE) do begin
             ll := ll + 1;   read(FONTE,linha[ll]);
         end;
         ll := ll + 1;       linha[ll] := CHR(NEWLINE);
         readln(FONTE);
         YYLINHA := YYLINHA + 1;  (* atualiza contador de linha *)
    end;
    cc := cc + 1;   c := linha[cc];
end (*GETCHAR*)
```

```
PROCEDURE YYSCAN:
        ...
BEGIN
     ...
   YYPOS := cc; (* salva inicio do token  *)
   if ch in letra then begin
   ...
       YYSIMB := xid;    (* retorna o tipo do token *)
       "id".valor := INSTALA(ident) (* retorna o valor do token *)
                ...
    end
    else if ch in ['0'..'9'] then begin
                      ...
             YYSIMB := xcte;    (* retorna o tipo do token *)
                      ...
           "cte".valor := inum;  (* retorna o valor da constante*)
         end
         else if ch = ';' then begin
                  GETCHAR(ch);
                  YYSIMB := xpv
              end
                ...
END;  (* YYSCAN *)
```

2.1.9 Seção Gramática e Rotinas Semânticas

No esquema de geração de código dirigida por sintaxe implementado no SIC, rotinas semânticas que definem as ações necessárias à geração de código devem ser associadas às produções da gramática. Essas rotinas serão ativadas quando as produções associadas forem usadas para reduzir elementos gramaticais na região do topo da pilha sintática. A seção gramática serve para especificar as regras da gramática da linguagem a ser implementada e as rotinas semânticas associadas. A gramática codificada nesta seção serve para produzir as tabelas *LALR(1)* necessárias à análise sintática.

```
gramatica        =  "%%GRAMMAR"   nt   [ "AND"  "SEMANTICS" ]
                    producoes ;
producoes        =  nt  "="   def_prod  { "|" def_prod } ";"  ;
def_prod         =  ldireito  [ precedencia ]
                    [ rotina_semantica ];
ldireito         =  { simbolos };
precedencia      =  "%%PREC"   token  |  "%%PREC" identificador ;
rotina_semantica =  " { "   [ texto ]   " } "  ;
```

Pode-se usar a barra vertical nas regras da gramática que tem o mesmo lado esquerdo para evitar a repetição do lado esquerdo. Assim, as regras da gramática

```
          cmd =  "id"   ":="   exp;
          cmd =  "if" cond "then" cmd "else"  cmd;
          cmd =  cmdc;
podem ser escritas
           cmd =  "id"   ":="   exp
               |  "if" cond "then" cmd "else"  cmd;
               |  cmdc;
```

Não é necessário agrupar as regras da gramática contendo o mesmo lado esquerdo, entretanto, este agrupamento torna a entrada mais legível e facilita mudanças.

```
Exemplo:
          cmd  =  "id"   ":="   exp;
          cond =  exp;
          cmd  =  "if" cond "then" cmd "else"  cmd
               |  cmdc;
```

As produções vazias são representadas da seguinte maneira:

$$\text{EMPTY} = ;$$

A rotina ou ação semântica, associada a cada produção é definida por um ou mais comandos em Pascal colocados entre chaves, e como tal, pode fazer entrada e saída, invocar procedimentos, alterar valores de variáveis e vetores externos; salvar tabela de s´ imbolos, etc. É importante ressaltar aquí, que os comentários dentro deste contexto só podem ser delimitados pelo par (* *) dado que o par { } é usado como delimitador de rotina semântica.

```
Por exemplo
cmd = "id"  ":="  exp
    { classe := ts["id".valor].classe;
       if (FOIDECLARADO("id".valor)) and
          ((classe=variavel) or ( classe = PAR ) then
          GEN(CATRIB,"id".valor,exp.r,0)
       else ERRO(7,"id".YYLINHA,"id".YYPOS)  };
e
exp  =  "cte"
     {  exp.r := "cte".valor ;
          exp.tipo := INTEIRO };
```

são regras com ações semânticas associadas. Para que referências a símbolos da gramática, i.e., terminais e não-terminais, ocorrendo do lado esquerdo ou lado direito de uma produção da gramática sejam feitas sem ambiguidades, na definição de ações semânticas esses símbolos podem conter índices. Adota-se a convenção de que um símbolo sem índice designa a sua primeira ocorrência na produção associada, contando com o lado esquerdo, e símbolo com um índice k $>$ 1 designa a sua k-ésima ocorrência na produção.

```
Exemplos:
      {  exp   =   exp  +  exp
          if exp[2].tipo <> exp[3].tipo then
             ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
          if exp[2].tipo = INTEIRO then op := CMAIS
          else op := COU;
          temporario := temp;
          GEN(OP,temporario,exp[2].r,exp[3].r);
          exp.r    := temporario;
          exp.tipo := exp[2].tipo;  };

       exp   =   exp  *  exp
        { if exp[2].tipo <> exp[3].tipo then
             ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
          if exp[2].tipo = INTEIRO then op := CVEZES
          else op := CE;
          temporario := temp;
          GEN(OP,temporario,exp[2].r,exp[3].r);
          exp[1].r    := temporario;
          exp[1].tipo := exp[2].tipo;   } ;
```

Nos exemplos acima, *exp[1].r* e *exp[1].tipo* ou *exp.r* e *exp.tipo* designam os atributos *r* e *tipo* do exp do lado esquerdo; *exp[2].r* e *exp[2].tipo* denotam os atributos *r* e *tipo* do primeiro exp do lado direito e *exp[3].r, exp[3].tipo* do último. Note que, pode-se atribuir valores a atributos do não-terminal do lado esquerdo sem que os atributos do primeiro símbolo do lado direito sejam afetados, embora ambos, ocupem a mesma posição na pilha semântica. Uma rotina semântica que nada faz, deve ser especificada com a forma

{ } ;

A associação de rotina semântica a toda produção não é compulsória. A ausência de rotina semântica indica que a construção definida pela produção ainda não foi implementada. Caso a produção associada seja envolvida em alguma redução, o compilador gerado pelo SIC emitirá uma mensagem lembrando o usuário que a construção ainda não foi implementada.

A claúsula %%PREC serve para associar um nível de precedência a uma regra da gramática. %%PREC deve aparecer imediatamente depois da definição do lado direito da regra e antes da ação semântica ou do delimitador ";".

O uso de %%PREC faz com que a regra tenha a mesma precedência do símbolo terminal, literal ou identificador especificado na cláusula. A ausência de %%PREC implica que a regra da gramática terá a mesma precedência do símbolo terminal ocorrido mais a direita. Se não houver este símbolo terminal, a precedência da regra é definida por uma constante indicando que a precedência não foi especificada. Por exemplo, para que a redução da produção

```
EXP = "-" EXP
```

tenha a mesma precedência que a multiplicação, deve-se escrever

```
exp = "-" exp %%PREC "*";
```

O valor da precedência de uma produção é usado para resolver conflitos da tabela LALR(1).

2.1.10 Seção Resolução de Conflitos

Gramáticas livres-do-contexto não-ambíguas constituem uma excelente ferramenta para definição precisa de sintaxe de linguagem de programação e geração automática de reconhecedores determinísticos [HOPCROFT 69]. Entretanto, várias construções sintáticas comuns em linguagens de programa são especificadas de forma mais natural e suscinta através de gramáticas ambíguas do que usando uma gramática equivalente não-ambígua. O SIC concilia estes dois interesses implementando o método de geração de tabelas *LALR(1)* usando gramáticas ambíguas proposto em [AHO 77]. O uso de gramáticas ambíguas levam, inevitavelmente, ao aparecimento de conflitos na tabela *LALR(1)* que devem ser resolvidos diretamente pelo usuário. A seção resolução de conflitos serve para especificar a precedência de todos os operadores e a associatividade dos operadores binários. Esta informação possibilita ao SIC resolver os conflitos existentes na análise sintática e construir um analisador que obedeça as relações de precedência e associatividade estabelecidas.

```
resolucao_de_conflito  =  "%%CONFLICTS"  resolucaoconflitos
                          { [ ";" ] resolucaoconflitos }
                          [ ";" ] ;
resolucaoconflitos     =   associa_se_a simbolos
                          { "," simbolos };
associa_se_a            = "%%RIGHT"
                        = "%%LEFT"
```

A declaração de precedência e associatividade é feita através de uma série de claúsulas começando com as palavras chaves *%%RIGHT, %%LEFT* e *%%NONE* seguida de uma lista de símbolos terminais. Cada claúsula define um novo nível de precedência que é maior do que o da cláusula anterior. Os símbolos terminais declarados na mesma cláusula têm o mesmo nível de precedência e associatividade.

```
Exemplos
              %%CONFLITS
                    %%NONE "<",">","="
                    %%LEFT "+", "-"
                    %%RIGHT "*","/"
```

O exemplo descreve a relação de precedência e associatividade para quatro operadores aritméticos e de relação. Em primeiro lugar, o "<", o ">" e o "=" têm precedência menor que o

“+” e “-” ” e não se associam. O “+” e o “-” associam-se à esquerda e têm menor precedência que “*” e “/”, os quais associam-se à direita. Uma regra da gramática também pode ter uma precedência diferente daquela dos símbolos terminais e literais da gramática, bastando que se defina um identificador na seção de resolução_de_conflitos, e use tal identificador na cláusula *%%PREC* correspondente. Por exemplo

```
%%CONFLITS
      %%LEFT "+" "-"
      %%RIGHT ZZ
      %%LEFT "*" "/"
      exp = exp "+" exp %%PREC ZZ
```

indica que a regra da gramática,

```
exp = exp "+"  exp
```

terá uma precedência *a do identificador ZZ* maior que “+” e menor que “*”.

2.1.11 Corpo

A seção corpo serve para especificação do programa principal do compilador.

corpo = ”%%PROGRAM” [texto] ;

Após a palavra chave *%%PROGRAM* pode ser compilado qualquer comando em Pascal, exigindo-se apenas que o procedimento *YYPARSER* seja ativado em algum ponto. MBIGONHA 83

AHO, A. V., Ullman, J. D., Principles of Compiler Design, Addison -Wesley Publishing Company, 1973.

AHO , A. V.,and ULLMAN, J. D., The Theory of parsing, translation and compiling, Vols. 1 e 2, Prentice-Hall, Englewood Cliffs, N.J. 1972.

AHO, Alfred V. & Ulmman, J. D., *Deterministic Parsing of Ambiguous Grammar*, Comm. ACM, Vol. 18, No. 8, Aug. 1977, pp.441–452.

HOPCROFT, J. E. & Ulmman, J. D. Formal Language and their Relation to Automata. Reading Mass., Addison Wesley, 1969.

Jensen, K. & Wirth, N., PASCAL User Manual and Report, Springer- Verlag 1974.

Knuth, D. E., *Semantics of Context-Free Languages, Mathematical System Theory 2, (1968) pp. 127-145. Correction : Mathematical System Theory 5 (1971),95-96.*

Knuth, D., The Art of Computer Programming, 2nd Edition,Addison- Wesley Publishing Co.,(1973).

BIGONHA, Mariza A. S., SIC : Sistema de Implementação de Compi ladores, Tese de Mestrado, Departamento de Ciência da Compu tação (DCC) Icex-UFMG, junho/1985.

BIGONHA, Mariza A.S., e BIGONHA, Roberto S., *Uma Experiência na Implementação de um Recuperador de Erro LR(1), Anais do V Simpósio sobre Desenvolvimento de Software Básico, Belo Hori zonte, 1985.*

BIGONHA, Mariza A.S. e BIGONHA, Roberto S., SIC: Sistema de Implementação de Compiladores, Série de Monografias do DCC - ICEX - UFMG, No. T02/86.

BIGONHA, Mariza A.S. e BIGONHA, Roberto S., *SIC: Um Sistema de Suporte à Implementação de Compiladores, Anais do XIII Seminário Integrado de Software e Hardware, Recife, 1986.*

BIGONHA, Mariza A.S e BIGONHA, Roberto S., SIC : Uma Ferramenta para Implementação de Linguagens, TRABALHO vencedor do III Prêmio Nacional de Informática 1988, categoria Software, Anais do XXI Congresso Nacional de Informática, SUCESU, 1988, 426-431.

BIGONHA, Roberto S. e BIGONHA, Mariza A.S., *Um Método de Compactação de Tabelas LR (1), Anais do III Seminário sobre Software Básico para Micros, Rio de Janeiro, 1983.*

A Apêndice 1: SAÍDA DO PROGRAMA FONTE

```
SIC : SISTEMA DE IMPLEMENTACAO DE COMPILADORES 4.0          PAGINA   1

    1   %%COMPILER PROGRAM exemplo; %%BATCH
    2
    3
    4
    5   %%TOKENS
    6
    7        "id"       = xid ;
    8        "cte"      = xcte ;
    9        "program " = xprogram
   10        "end"      = xend
   11        "begin"    = xbegin
   12        "integer"  = xinteger
   13        "procedure"= xprocedure
   14        "if"       = xif
   15        "then"     = xthen
   16        "else"     = xelse
   17        "while"    = xwhile
   18        "do"       = xdo
   19        ";"        = xpv
   20        ":"        = xdp
   21        ":="       = xatr
   22        "("        = xap
   23        ")"        = xfp
   24        "+"        = xmais
   25        "-"        = xmenos
   26        "eof"      = YYEOF
   27        "*"        = xvezes
   28        "/"        = xdiv
   29        "**"       = xpot
   30        "="        = xigual
   31        "error"    = xerror
   32        "boolean"  = xboolean
   33
   34
   35
   36   %%STACK 100  OF ATTRIBUTS
   37
   38
   39         exp      = (r,tipo:INTEGER)  { exp.r := 0; exp.tipo := INTEIRO };
   40         "id"     = (valor :INTEGER)  { "id".valor := 1 }
   41         "cte"    = (valor :INTEGER)
   42         cond     = (quad  :INTEGER)
   43         n        = (quad  :INTEGER)
   44         marca    = (quad  :INTEGER)
```

```
45         prochead = (valor :INTEGER; inicio : INTEGER)
46
47
48    %%SCOPEMAP
49
50         "begin" : "end" ;
51         "("     : ")"   ;
52
53    %%NTMAP
54
55         exp , dcl , cmd ;
56
57
58
59    (*=============================================================*)
60    (*                  MODULO DE TRATAMENTO DE ERRO               *)
61    (*=============================================================*)
62
63
64    %%CONSTANTS
65
66
67         MAXMSG   = 20;
68         MAXERROR = 11;
69
70
71    %%TYPES
72
73         erros  = packed array[1..MAXMSG] of char;
74
75         mensg  = record
76                     msg : erros;
77                  end;
78
79
80    %%VARIABLES
81
82         out : file of mensg;
83
84
85    %%PROCEDURES
86
87
88         procedure ERRO(n,l,p:integer);
```

```
 89        var i       : integer;
 90            message : mensg;
 91        begin
 92            seek(out,n);
 93            read(out,message);
 94            writeln(YYSAIDA);
 95            writeln(YYSAIDA,'+++++ ',message.msg,' na linha ',l,
 96                            ' posicao ',p);
 97        end;
 98
 99
100
101   (*============================================================*)
102   (*                 MODULO TABELA DE SIMBOLOS                  *)
103   (*============================================================*)
104
105
106   %%CONSTANTS
107
108
109        (* tipo tabela de simbolos *)
110
111        INTEIRO    = 1;
112        LOGICO     = 2;
113
114        (*  classe tabela de simbolos *)
115
116        NAO_DECL   =  0;
117        VARIAVEL   =  1;
118        PAR        =  3;
119        PROC       =  4;
120        NMAX       = 29;
121        MAX        = 1000;
122        ALPHA1     = 9;
123        NPC        = 10;
124        IDTAM      = 8;    (* no. caracteres significativos ident *)
125
126
127   %%TYPES
128
129
130        charset         = set of char;
131        alfa            = array[1..IDTAM] of char;
132        alfa1           = array[1..ALPHA1] of char;
```

```
133
134       tipoalpha       = array[1..IDTAM] of char;
135
136       tscampos = record
137                      nome   :alfa;
138                      classe :integer;
139                      tipotam:integer;
140                      endoff :integer;
141                      nivel  :integer;
142                      col    :integer;
143                  end;
144
145  %%VARIABLES
146
147       (* tabela de simbolos *)
148
149       nivel,l       : integer;
150       ts            : array[1..MAX] of tscampos;
151       thash         : array[0..90] of integer;
152       escopo        : array[1..NMAX] of integer;
153
154
155  %%PROCEDURES
156
157       FUNCTION HASH(var simb:alfa):integer;
158       var
159           i,h:integer;
160       begin
161           h:= 0;
162           i := 1;
163           while (simb[i] <> ' ') AND (i < IDTAM) do
164           begin
165               h:=h + ord(simb[i]);
166               i := i + 1;
167           end;
168           hash := h mod 91;
169       end;  (*hash*)
170
171
172
173       FUNCTION INSTALA(var simb:alfa):integer;
174       label 2;
175       var
```

```
  176           n,k:integer;
```



```
  177       begin
  178           n:=hash(simb);
  179           k:=thash[n];
  180           while k >= escopo[nivel] do
  181           begin
  182               if simb = ts[k].nome then
  183               begin
  184                   erro(1,YYLINHA,YYPOS);
  185                   instala := k;
  186                   goto 2
  187               end
  188               else k:=ts[k].col
  189           end;
  190           if l = max + 1 then
  191           begin
  192               erro(2,YYLINHA,YYPOS);               (* estouro da TS *)
  193               instala := l;
  194               goto 2
  195           end;
  196           ts[l].nome :=simb;
  197           ts[l].nivel:=nivel;
  198           ts[l].classe:=NAO_DECL;
  199           ts[l].col:=thash[n];
  200           thash[n]:=l;
  201           instala := l;
  202           l:=l+1;
  203       2:end; (* instala *)
  204
  205
  206
  207       FUNCTION PROCURAID(var simb:alfa):integer;
  208       label 1;
  209       var
  210           n,k:integer;
  211       begin
  212           n :=hash(simb);
  213           k:=thash[n];
  214           while k <> 0 do
  215           begin
  216               if simb = ts[k].nome then
  217               begin
  218                   PROCURAID := k;
```

```
219                     goto 1
220                 end;
```



```
221                 k :=ts[k].col
222             end;
223             PROCURAID := INSTALA(simb);
224         1:end; (* procuraid*)
225
226
227
228         PROCEDURE ABLOCO;
229         begin
230             nivel := nivel + 1;
231             if nivel > nmax
232             then erro(3,YYLINHA,YYPOS)  (* estouro limite de niveis *)
233             else escopo[nivel]:=l;
234         end;(*abloco*)
235
236
237
238         PROCEDURE FBLOCO;
239         var
240             s,b,k:integer;
241         begin
242             s:=l;
243             b:=escopo[nivel];
244             (* desfaz hash *)
245             while s > b do begin
246                   s := s - 1;
247                   k:=hash(ts[s].nome);
248                   thash[k]:=ts[s].col;
249             end;
250             nivel:=nivel - 1;
251         end;(*fbloco*)
252
253
254
255         PROCEDURE TAMANHO(proced,addr:integer);
256         begin
257             ts[proced].endoff:=addr;
258         end; (* tamanho *)
259
260
261
```

```
  262       PROCEDURE DECLARA(k,class,tipot,offset:integer);
  263       begin
  264           ts[k].classe:=class;
```



```
  265           ts[k].tipotam := tipot;
  266           ts[k].endoff := offset
  267       end; (* declara *)
  268
  269
  270
  271       FUNCTION FOIDECLARADO(k:integer):boolean;
  272       begin
  273           if ts[k].classe <> NAO_DECL
  274           then FOIDECLARADO := true
  275           else FOIDECLARADO := false
  276       end; (*foideclarado*)
  277
  278
  279   (*=============================================================*)
  280   (*                MODULO   OFFSET                              *)
  281   (*=============================================================*)
  282
  283
  284   %%VARIABLES
  285
  286
  287       poff            : array[1..7] of integer;
  288       toff            : integer;
  289       offset          : integer;
  290
  291
  292   %%PROCEDURES
  293
  294
  295       PROCEDURE POPOFF(offst:integer);
  296       begin
  297           if toff < 1 then erro(4,YYLINHA,YYPOS)
  298           else begin
  299                offst := poff[toff];
  300                toff:=toff-1;
  301           end;
  302       end; (* popoff *)
  303
  304
```

```
  305
  306        PROCEDURE PUSHOFF(offst:integer);
  307        begin
  308            if toff > 7 then erro(5,YYLINHA,YYPOS)
```



```
  309            else begin
  310                 toff:=toff +1;
  311                 poff[toff]:=offst;
  312                 offst:=1;
  313            end;
  314        end; (* pushoff *)
  315
  316
  317        FUNCTION TEMP:integer;
  318        begin
  319            TEMP := temps;
  320            temps:= temps - 1
  321        end;
  322
  323
  324   (*=====================================================================*)
  325   (*                  MODULO   DE ANALISE LEXICA                         *)
  326   (*=====================================================================*)
  327
  328
  329   %%CONSTANTS
  330
  331
  332        (**** GETCHAR ****)
  333
  334        LINHAMAX  = 66;
  335        SPACELINE = 1;
  336
  337        (***** YYSCAN ******)
  338
  339        ENDFILE      = 0;
  340        NEWLINE      = 13;
  341        HTAB         = 9;
  342        LINEFEED     = 10;
  343        CTLZ         = 26;
  344        BACKSPACE    = 8;
  345        VT           = 11;
  346        FF           = 12;
  347        TLE          = 72;
```

```
  348
  349
  350
  351   %%VARIABLES
  352
```



```
  353        errofatal       : boolean;
  354        FONTE           : text;
  355
  356        (*YYSCAN*)
  357
  358        ii              : integer;
  359        ctx             : integer;
  360        letra           : charset;
  361        letraminuscula  : charset;
  362        letramaiuscula  : charset;
  363        key             : array[1..11] of tipoalpha;
  364        tipo_pal_res    : array[1..11] of integer;
  365
  366        (***** GETCHAR ******)
  367
  368        ch              : char;
  369        ll,cc           : integer;
  370        linha           : array[1..tle] of char;
  371        pagina          : integer;
  372        linenum         : integer;
  373        pagcomp         : integer;
  374        marginf         : integer;
  375        margem          : integer;
  376        margsup1        : integer;
  377        margsup2        : integer;
  378        margsup         : integer;
  379
  380
  381   %%PROCEDURES
  382
  383
  384
  385        PROCEDURE MONTATABELAS;
  386        begin
  387            (* tabela de palavras reservadas *)
  388            key[1] := 'begin   ';
  389            key[2] := 'boolean ';
  390            key[3] := 'do      ';
```

```
391             key[4] := 'else    ';
392             key[5] := 'end     ';
393             key[6] := 'if      ';
394             key[7] := 'integer ';
395             key[8] := 'procedur';
396             key[9] := 'program ';
```



```
397             key[10] := 'then    ';
398             key[11] := 'while   ';
399
400             tipo_pal_res[1]:=xbegin;
401             tipo_pal_res[2]:=xboolean;
402             tipo_pal_res[3]:=xdo;
403             tipo_pal_res[4]:=xelse;
404             tipo_pal_res[5]:=xend;
405             tipo_pal_res[6]:=xif;
406             tipo_pal_res[7]:=xinteger;
407             tipo_pal_res[8]:=xprocedu;
408             tipo_pal_res[9]:=xprogram;
409             tipo_pal_res[10]:=xthen;
410             tipo_pal_res[11]:=xwhile;
411         end; (* montatabelas *)
412
413
414
415         PROCEDURE INICIA;
416         begin
417             ch := ' ';
418             ll := 0;  cc := 0;
419             pagina  := 0;
420             linenum := 0;
421             pagcomp := linhamax;
422             margem  := 4;
423             margsup1:= 2;
424             margsup2:= 2;
425             margsup := margsup1 + margsup2 + 3;
426             marginf := pagcomp - margem;
427             ctx     := 1;
428             l       := 2;
429             ts[1].nome    := '        ';
430             ts[1].nivel   := 1;
431             ts[1].classe  := NAO_DECL;
432             ts[1].tipotam := INTEIRO;
433             ts[1].endoff  := 0;
```

```
434             letramaiuscula := ['A'..'Z'];
435             letraminuscula := ['a'..'z'];
436             letra := letramaiuscula + letraminuscula;
437             for ii:=1 to 1000 do ts[ii].col:=0;
438             for ii:=0 to 90 do thash[ii]:=0;
439             montatabelas;
440             temps := -1;
```



```
441             nivel := 0;
442             ABLOCO;
443         end;
444
445
446
447         PROCEDURE WRITEFONTE;
448         var i : integer;
449         begin
450             if linenum > marginf then
451             begin
452                 write(YYSAIDA,' ':4);
453                 for i :=1 to 15 do write(YYSAIDA,'----+');
454                 for i:=1 to margem-1 do writeln(YYSAIDA);
455                 linenum := 0;
456             end;
457             if linenum = 0 then        (* comeca nova pagina *)
458             begin
459                 pagina := pagina + 1;
460                 for i := 1 to margsup1 do writeln(YYSAIDA);
461                 write(YYSAIDA,' ':4);
462                 for i := 1 to 15 do write(YYSAIDA,'----+');
463                 writeln(YYSAIDA);
464                 writeln(YYSAIDA,'LINHA',' ':8,'TEXTO',' ':40,'PAGINA '
465                    ,pagina:3);
466                 for i := 1 to margsup2 do writeln(YYSAIDA);
467                 linenum := margsup + 3;
468                 write(YYSAIDA,YYLINHA:1,' ':3);
469                 for i := 1 to ll-1 do write(YYSAIDA,linha[i]);
470                 linenum := linenum + spaceline;
471                 for i:= 1 to spaceline do writeln(YYSAIDA)
472             end
473             else begin
474                  write(YYSAIDA,YYLINHA:1,' ':3);
475                  for i := 1 to ll-1 do write(YYSAIDA,linha[i]);
476                  linenum := linenum + spaceline;
```

```
477                 for i:= 1 to spaceline do writeln(YYSAIDA)
478             end;
479         end;
480
481
482         PROCEDURE GETCHAR(var c:char);
483         label 1;
484         begin
```



```
485             if cc = ll then
486             begin
487                 if eof(FONTE) then
488                 begin
489                     c  := chr(endfile);
490                     goto 1;
491                 end;
492                 ll := 0; cc:= 0;
493                 while not eoln(FONTE) do
494                 begin
495                     ll := ll + 1;
496                     read(FONTE,linha[ll]);
497                 end;
498                 ll := ll + 1; linha[ll] := CHR(NEWLINE);
499                 readln(FONTE);
500                 YYLINHA := YYLINHA + 1;
501
502                 (* impressao do texto de entrada *)
503                 WRITEFONTE;
504             end;
505             cc := cc + 1;
506             c := linha[cc];
507         1:end; (*GETCHAR*)
508
509
510         PROCEDURE YYSCAN;
511         label 1,2;
512         var
513             inum,i,j,k,kk:integer;
514             tam:integer;
515             ident : alfa;
516         begin
517             for k := 1 to IDTAM do ident[k] := ' ';
518             1:while(ch = ' ') or (ch = chr(htab)) or (ch = chr(newline))
519               or (ch = chr(backspace)) or (ch = chr(vt))
```

```
520           or (ch = chr(ff)) or (ch = chr(linefeed)) do getchar(ch);
521
522           YYPOS := cc;
523           if ch = chr(endfile) then
524           begin
525               YYSIMB := YYEOF;
526               goto 2;
527           end
528           else if ch = '{' then
```



```
529                begin
530                    GETCHAR(ch);
531                    while ch <> '}' do
532                    GETCHAR(ch);
533                    GETCHAR(ch);
534                    goto 1;
535                end
536           else if ch in letra  then
537                begin (*palavra*)
538                    tam:=0;
539                    repeat if tam < IDTAM then
540                           begin
541                               {converte carater lido para minuscula }
542                               if(ch in letramaiuscula)
543                               then ch:=chr(ord(ch)+32);
544                               tam := tam + 1;
545                               ident[tam]:=ch
546                           end;
547                           GETCHAR(ch);
548                    until not(ch in ['a'..'z','A'..'Z','0'..'9']);
549                    (* procura por palavra chave *)
550                    i:=1;
551                    j:=npc;
552                    repeat k:=(i+j) div 2;
553                           if ident <= key[k] then j:=k-1;
554                           if ident >= key[k] then i:=k+1;
555                    until i > j;
556                    if i-1>j then  YYSIMB := tipo_pal_res[k]
557                    else begin
558                         YYSIMB:=xid;
559                         if ctx = 1 then
560                            "id".valor := instala(ident)
561                         else "id".valor := procuraid(ident);
562                    end;
```

```
563                end
564                else if ch in ['0'..'9'] then
565                     begin (*numero*)
566                         inum:=0;
567                         YYSIMB := xcte;
568                         repeat kk:=ord(ch) - ord('0');
569                             if (inum > 3276) or ((inum = 3276)
570                                                  and (kk > 7))
571                             then erro(6,YYLINHA,YYPOS)
572                             else begin
```



```
573                                 inum:=inum *10+kk;
574                                 GETCHAR(ch)
575                             end;
576                         until not(ch in ['0'..'9']);
577                         "cte".valor := inum;
578                     end
579                     else case ch of
580                          ':' : begin
581                                    GETCHAR(ch);
582                                    if ch = '=' then
583                                    begin
584                                        YYSIMB:=xatr;
585                                        GETCHAR(ch)
586                                    end
587                                    else YYSIMB:=xdp
588                                end;
589
590                         ';' :  begin
591                                    GETCHAR(ch);
592                                    YYSIMB:=xpv
593                                end;
594
595                         '(' :  begin
596                                    GETCHAR(ch);
597                                    YYSIMB:=xap
598                                end;
599
600                         ')' :  begin
601                                    GETCHAR(ch);
602                                    YYSIMB:=xfp;
603                                end;
604
605                         '+' :  begin
```

```
606                                   GETCHAR(ch);
607                                   YYSIMB:=xmais
608                                end;
609
610                           '-' :  begin
611                                     GETCHAR(ch);
612                                     YYSIMB:=xmenos
613                                   end;
614
615                           '*' :  begin
616                                     GETCHAR(ch);
```



```
617                                     if ch = '*' then
618                                     begin
619                                         YYSIMB:=xpot;
620                                         GETCHAR(ch)
621                                     end
622                                     else YYSIMB :=xvezes
623                                end;
624
625                           '/' : begin
626                                     GETCHAR(ch);
627                                     YYSIMB := xdiv;
628                                end;
629
630                           '=' : begin
631                                     GETCHAR(ch);
632                                     YYSIMB := xigual;
633                                end;
634
635                           else  begin
636                                     WRITELN(YYSAIDA,'CHAR=',ord(ch),ch);
637                                     YYSIMB := xerror;  {40;}
638                                     GETCHAR(ch);
639                                     writeln(YYSAIDA,'char=',ord(ch):1,
640                                     ch:1);
641                                end;
642                      end; (* case *)
643       2:end; (*yyscan*)
644
645   (*================================================================*)
646   (*                 MODULO CODIGO INTERMEDIARIO                    *)
647   (*================================================================*)
648
649
650   %%CONSTANTS
651
652        (* operadores para opquad *)
```

```
653
654         CPROCEND        = 1;
655         CPROCBEGIN      = 2;
656         CATRIB          = 3;
657         CPARAM          = 4;
658         CCALL           = 5;
659         CIGUAL          = 6;
660         CGOTO           = 7;
```



```
661         CMAIS           = 8;
662         COU             = 13;
663         CVEZES          = 9;
664         CE              = 14;
665         CMENOS          = 10;
666         CINVERTE        = 15;
667         CNEGA           = 16;
668         CDIV            = 11;
669         CPOT            = 12;
670         CDSVF           = 17;
671         CPROGEND        = 18;
672         CPROGBEGIN      = 19;
673
674   %%TYPES
675
676
677         (* gen *)
678
679         tquad = record
680                     opquad,opn1quad,opn2quad,opn3quad : integer;
681                 end;
682
683
684   %%VARIABLES
685
686         (* gen *)
687         op    : integer;
688         quad  : array[1..500] of tquad;
689         code  : file of tquad;
690         proxq : integer;
691         opnome: array[1..19] of tipoalpha;
692         temps : integer;
693
694
695   %%PROCEDURES
```

```
696
697
698         PROCEDURE GEN(operador,opn1,opn2,opn3:integer);
699         begin
700             quad[proxq].opquad   := operador;
701             quad[proxq].opn1quad := opn1;
702             quad[proxq].opn2quad := opn2;
703             quad[proxq].opn3quad := opn3;
704             proxq:=proxq + 1
```



```
705         end; (* gen *)
706
707
708
709         PROCEDURE CONSERTA(cond,addr:integer);
710         begin
711             quad[cond].opn2quad := addr;
712         end; (* conserta *)
713
714
715
716         PROCEDURE SALVA(i,k : integer);
717         var j : integer;
718         begin
719             for j := i to k do write(code,quad[j])
720         end;
721
722
723
724         PROCEDURE MONTAOPNOME;
725         begin
726             opnome[1 ] := 'PROCEND ';
727             opnome[2 ] := 'PRCBEGIN';
728             opnome[3 ] := ':=      ';
729             opnome[4 ] := 'PARAM   ';
730             opnome[5 ] := 'CALL    ';
731             opnome[6 ] := '=       ';
732             opnome[7 ] := 'GOTO    ';
733             opnome[8 ] := '+       ';
734             opnome[9 ] := '*       ';
735             opnome[10] := '-       ';
736             opnome[11] := '/       ';
737             opnome[12] := '**      ';
738             opnome[13] := 'OU      ';
```

```
739             opnome[14] := 'E       ';
740             opnome[15] := '-       ';
741             opnome[16] := 'NEGA    ';
742             opnome[17] := 'DSVF    ';
743             opnome[18] := 'PROGEND ';
744             opnome[19] := 'PRGBEGIN';
745         end;
746
747
748
```



```
749         PROCEDURE IMPRIMECODIGO;
750         var i,j   : integer;
751             quad  : tquad;
752         begin
753             MONTAOPNOME;
754             writeln(YYSAIDA,' ':4,'============CODIGO GERADO==========');
755             writeln(YYSAIDA);
756             writeln(YYSAIDA,' ':6,'opquad',' ':2,'opn1quad ','opn2quad ',
757                             'opn3quad');
758             j := 1;
759             while not eof(code) do begin
760                 read(code,quad);
761                 writeln(YYSAIDA);
762                 write(YYSAIDA,j:4,' ':2,opnome[quad.opquad]);
763                 case quad.opquad of
764
765                   CPROCEND    : ;
766
767                   CPROCBEGIN  : write(YYSAIDA,' ':4,quad.opn1quad);
768
769                   CATRIB      : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
770                                                     quad.opn2quad);
771
772                   CPARAM      : write(YYSAIDA,' ':4,quad.opn1quad);
773
774                   CCALL       : write(YYSAIDA,' ':4,quad.opn1quad);
775
776                   CGOTO       : write(YYSAIDA,' ':12,quad.opn2quad);
777
778                   CMAIS       : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
779                                       quad.opn2quad,' ':8,quad.opn3quad);
780
781                   COU         : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
```

```
782                                     quad.opn2quad,' ':8,quad.opn3quad);
783
784                 CIGUAL      : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
785                                     quad.opn2quad,' ':8,quad.opn3quad);
786
787                 CDIV        : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
788                                     quad.opn2quad,' ':8,quad.opn3quad);
789
790                 CVEZES      : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
791                                     quad.opn2quad,' ':8,quad.opn3quad);
792
```



```
793                 CE       : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
794                                     quad.opn2quad,' ':8,quad.opn3quad);
795
796                 CPOT        : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
797                                     quad.opn2quad,' ':8,quad.opn3quad);
798
799                 CMENOS      : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
800                                     quad.opn2quad,' ':8,quad.opn3quad);
801
802                 CINVERTE    : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
803                                     quad.opn2quad);
804
805                 CNEGA       : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
806                                     quad.opn2quad);
807
808                 CDSVF       : write(YYSAIDA,' ':4,quad.opn1quad,' ':6,
809                                                   quad.opn2quad);
810                 CPROGEND    : ;
811
812                 CPROGBEGIN  : ;
813
814             end; (* case *)
815             j := j + 1;
816           end; (* while *)
817
818           (* imprime tabela de simbolos *)
819           writeln(YYSAIDA);
820           writeln(YYSAIDA,' ':4,'=========TABELA DE SIMBOLOS=======');
821           writeln(YYSAIDA,'nome',' ':4,'classe',' ':4,'tipotam',' ':4,
822                           'endoff',' ':4,'nivel');
823           for i := 1 to l-1 do
824               with ts[i] do
```

```
825                     writeln(YYSAIDA,nome:6,classe:4,tipotam:8,
826                     endoff:12,nivel:12);
827         end;
828
829   (*================================================================*)
830   (*              MODULO ROTINAS SEMANTICAS                         *)
831   (*================================================================*)
832
833
834   %%VARIABLES
835
836
```



```
837         temporario      : integer;
838         classe          : integer;
839
840
841
842   %%GRAMMAR    program  AND  SEMANTICS
843
844
845   program  = proghead dcls cmdc
846                       {SALVA(1,proxq-1);
847                        GEN(CPROGEND,0,0,0); };
848
849   proghead = "program"
850              { toff    := 0;
851                offset  := 0;
852                proxq   := 1;
853                 GEN(CPROGBEGIN,0,0,0); };
854
855   dcls  = dcl
856           {    }
857         |  dcls ";" dcl
858           {    } ;
859
860   dcl  = "id" ":" "integer"
861          { DECLARA("id".valor,VARIAVEL,INTEIRO,offset);
862            offset := offset +1};
863
864   dcl  = "id" ":" "boolean"
865          { DECLARA("id".valor,VARIAVEL,LOGICO,offset);
866            offset := offset +1};
867
```

```
868   dcl  = prochead "(" par ")" dcls cmdc
869          { TAMANHO(prochead.valor,offset);
870            POPOFF(offset);
871            ctx := 1;
872            GEN(CPROCEND,0,0,0);
873            FBLOCO;
874            SALVA(prochead.inicio,proxq-1);
875            proxq := prochead.inicio; };
876
877   par  = "id" ":" "integer"
878          { DECLARA("id".valor,PAR,INTEIRO,offset);
879            offset := offset +1};
880
```



```
881   par  = "id" ":" "boolean"
882          { DECLARA("id".valor,PAR,LOGICO,offset);
883            offset := offset +1};
884
885   prochead  = "id" ":" "procedure"
886               {DECLARA("id".valor,PROC,0,proxq);
887                prochead.inicio := proxq;
888                GEN(CPROCBEGIN,
889                   "id".valor,0,0);
890               prochead.valor := "id".valor;
891               ABLOCO;
892               PUSHOFF(offset)};
893
894   cmdc  = c "begin" cmds "end"
895              {    };
896
897   c  =
898           {ctx := 2 (* CONTEXTO DE COMANDOS  *)};
899
900   cmds  = cmd
901           {    }
902         |  cmds ";" cmd
903           {    };
904
905   cmd  = "id" ":=" exp
906          {classe := ts["id".valor].classe;
907           if(FOIDECLARADO("id".valor))and
908             ((classe=VARIAVEL) or ( classe = PAR)) then
909              GEN(CATRIB,"id".valor,exp.r,0)
910         else ERRO(7,"id".YYLINHA,
```

```
911                "id".YYPOS) };
912
913    cmd  = "id" "(" exp ")"
914           {if (FOIDECLARADO("id".valor)) and
915               (ts["id".valor].classe = proc)
916            then begin
917                 GEN(CCALL,"id".valor,0,0);
918                 GEN(CPARAM,exp.r,0,0);
919            end
920            else ERRO(8,"id".YYLINHA,
921                 "id".YYPOS) };
922
923   cmd  = "if" cond "then" cmd  "else" n cmd
924          {CONSERTA(cond.quad,n.quad+1);
```



```
925           CONSERTA(n.quad,proxq) };
926
927   cmd  = "while" marca cond "do"  cmd
928          {GEN(CGOTO,marca.quad,0,0);
929          CONSERTA(cond.quad,proxq)   };
930
931   cmd  = cmdc
932          {     };
933
934   cond  = exp
935           {if exp.tipo = LOGICO then
936               GEN(CDSVF,exp.r,0,0)
937            else erro(10,exp.YYLINHA,exp.YYPOS);
938            cond.quad := proxq; };
939
940   n  =
941           {n.quad := proxq;
942            GEN(CGOTO,0,0,0) };
943
944   marca  =
945           { marca.quad := proxq};
946
947   exp  = exp "+" exp
948          { if exp[2].tipo <> exp[3].tipo then
949            erro(9,exp[2].YYLINHA,exp[2].YYPOS);
950            if exp[2].tipo = INTEIRO
951            then op := CMAIS
952            else op := COU;
953            temporario := TEMP;
```

```
954           GEN(OP,temporario,exp[2].r,exp[3].r);
955           exp.r:=temporario;
956           exp.tipo := exp[2].tipo;  };
957
958   exp  = exp "=" exp
959          { if (exp[2].tipo <> exp[3].tipo) or
960               (exp[2].tipo <> INTEIRO) then
961           erro(9,exp[2].YYLINHA,exp[2].YYPOS);
962           temporario := temp;
963           GEN(CIGUAL,temporario,exp[2].r,exp[3].r);
964           exp.r := temporario;
965           exp.tipo := LOGICO; };
966
967   exp  = exp "/" exp
968          { if (exp[2].tipo <> exp[3].tipo) or
```



```
969              (exp[2].tipo <> INTEIRO) then
970           erro(9,exp[2].YYLINHA,exp[2].YYPOS);
971           temporario := temp;
972           GEN(CDIV,temporario,exp[2].r,exp[3].r);
973           exp.r := temporario;
974           exp.tipo := exp[2].tipo; };
975
976   exp  = exp "*" exp
977           { if exp[2].tipo <> exp[3].tipo then
978                erro(9,exp[2].YYLINHA,exp[2].YYPOS)  ;
979             if exp[2].tipo = INTEIRO
980             then op := CVEZES
981             else op := CE;
982             temporario:= temp;
983             GEN(OP,temporario,exp[2].r,exp[3].r);
984             exp.r  := temporario;
985             exp.tipo := exp[2].tipo; };
986
987   exp  = exp "**" exp
988          { if (exp[2].tipo <> exp[3].tipo) or
989               (exp[2].tipo <> INTEIRO) then
990           erro(9,exp[2].YYLINHA,exp[2].YYPOS);
991           temporario := temp;
992           GEN(CPOT,temporario,exp[2].r,exp[3].r);
993           exp.r := temporario;
994           exp.tipo := exp[2].tipo };
995
996    exp = exp "-" exp
```

```
 997          { if (exp[2].tipo <> exp[3].tipo) or
 998               (exp[2].tipo <> INTEIRO) then
 999            erro(9,exp[2].YYLINHA,exp[2].YYPOS);
1000            temporario := temp;
1001            GEN(CMENOS,temporario,exp[2].r,exp[3].r);
1002            exp.r := temporario;
1003            exp.tipo := exp[2].tipo;  };
1004
1005   exp  = "-" exp   %%PREC "*"
1006          { temporario := temp;
1007            if exp[2].tipo = INTEIRO
1008            then op := CINVERTE
1009            else op := CNEGA;
1010            gen(OP,temporario,exp[2].r,0);
1011            exp.r := temporario;
1012            exp.tipo := exp[2].tipo; }
```



```
1013
1014        | "(" exp ")"
1015          {exp.r := exp[2].r;
1016           exp.tipo := exp[2].tipo; };
1017
1018   exp  = "id"
1019          {classe := ts["id".valor].classe ;
1020           exp.r  := "id".valor;
1021           if FOIDECLARADO("id".valor) and
1022              ((classe = VARIAVEL) or
1023               (classe = PAR))
1024           then begin
1025                exp.tipo := ts["id".valor].tipotam;
1026           end
1027           else begin
1028                ERRO(7,"id".YYLINHA,"id".YYPOS) ;
1029                exp.tipo := INTEIRO;
1030           end;      };
1031
1032   exp  = "cte"
1033          { exp.r :="cte".valor;
1034            exp.tipo := INTEIRO; };
1035
1036
1037
1038   %%CONFLICTS
1039
```

```
1040
1041         %%RIGHT   "=" ;
1042         %%LEFT     "+","-" ;
1043         %%LEFT     "*","/"
1044         %%RIGHT   "**"
1045
1046    (*=================================================================*)
1047    (*                  MODULO PRINCIPAL                               *)
1048    (*=================================================================*)
1049
1050
1051    %%PROGRAM
1052
1053         assign(FONTE,'entrada');reset(FONTE);
1054         assign(YYSAIDA,'LST:');rewrite(YYSAIDA);
1055         assign(OUT,'EMITEMSG.ERR'); reset(OUT);
1056         assign(CODE,'EXEMPLO.COD'); rewrite(CODE);
```



```
1057         INICIA;
1058         YYPARSER;
1059         close(CODE); reset(CODE);
1060         IMPRIMECODIGO;
1061         case YYRESULTADO of
1062              YYSINTAXEOK   : write(YYSAIDA,' Parse completed,',
1063                                             ' sintaxe ok ');
1064              YYERROFATAL   : write(YYSAIDA,
1065                                ' erro fatal, programa cancelado');
1066              YYERROSINTAXE : write(YYSAIDA,' erro de sintaxe ');
1067         end;
1068         close(YYSAIDA); close(OUT);
1069         close(CODE);
1070    %%END
1071
1072
1073

 0 erros encontrados no passo 1
```