Departamento de Ciência da Computação
D C C

S I C
SISTEMA DE IMPLEMENTAÇÃO DE COMPILADORES

MANUAL DO USUÁRIO
VERSÃO C 1.0

Mariza Andrade da Silva Bigonha
Roberto da Silva Bigonha
Marco Rodrigo Costa
Valeska Gonçalves Russo

RELATÓRIO TÉCNICO RT/020-96
DCC-ICEx-UFMG

U F M G

Universidade Federal de Minas Gerais

SINOPSE

Este relatório apresenta a especificação de uma ferramenta, SIC, destinada a assistir a implementação de compiladores através de uma linguagem especial, denominada SIC, que é baseada no Pascal. SIC gera compiladores nas linguagens Pascal e C. Este manual é destinado aos projetistas de compiladores na linguagem C.

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 intermediária.

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 ambíguas na produção de analisadores sintáticos determinísticos.

ABSTRACT

This report describes a programming tool, SIC, whose purpose is to assist the construction of compilers by means of a special purpose language, SIC, based on Pascal. SIC produces compilers in Pascal and C languages. This manual is adressed to the C compilers designers.

SIC possesses facilities to specify the syntax of programming laguages, associate semantic routines to grammar productions and allows explicit definition of attributes. It also provides, without loss of eficiency, facilities to implement interactive or batch compilers organized into one or more pass, where each pass operates directly on the source code, requiring no intermediate language.

From the given syntax especification, the SIC compiler produces a compressed LALR(1) table and a parser containing a language independent error handling and recovery routine, which automatically issues error messages.

The System also presents facilities to explicity solve LALR(1) conflicts resulting from the use of ambiguous grammar.

SUMÁRIO

1. SISTEMA DE IMPLEMENTAÇÃO DE COMPILADORES (SIC)

1.1. Histórico do SIC

O Sistema de Implementação de Compiladores, SIC, é uma ferramenta de auxílio à escrita de compiladores que foi desenvolvida pelo Grupo de Linguagens de Programação do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais.

A versão original do sistema gera compilador na linguagem Pascal, foi implementada também em Pascal e desenvolvida para plataforma MS-DOS. Posteriormente foi produzida uma nova versão gerando compiladores na linguagem C, padrão ANSI, e utilizando a como plataforma o ambiente operacional WINDOWS e o compilador DELPHI. Este manual trata da versão C do sistema SIC.

Mais detalhes sobre a versão em Pascal podem ser encontrados em [MBIGONHA & BIGONHA, 1989].

1.2. 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. A linguagem SIC possui mecanismos para a definição de um compilador na linguagem C, padrão ANSI [KERNIGHAN & RITCHIE, 1988], além dos mecanismos considerados importantes na escrita de um compilador.

Estes mecanismos incluem:

a) Os mecanismos para definir a gramática da linguagem a ser implementada juntamente com suas respectivas rotinas semânticas na linguagem C;

b) Os mecanismos para gerar a pilha semântica automaticamente a partir da declaração explícita de atributos [AHO & ULLMAN, 1972] associados a símbolos da linguagem;

c) As facilidades para agrupar declarações de constantes, tipos e variáveis da linguagem C com declarações de funções e procedimentos;

d) As facilidades para declarar explicitamente os terminais da linguagem a ser implementada;

e) Facilidades para a implementação de compiladores de vários passos onde, cada passo opera diretamente no fonte, eliminando a necessidade de linguagem intermediária;

f) Os mecanismos que definem a produção de compiladores interativos ou "batch";

g) 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 & ULLMAN, 1977];

h) As facilidades de se definir características dependentes de linguagem com o objetivo de se ter recuperação de erro mais precisa.

O funcionamento de todos estes mecanismos será visto em detalhe no Capítulo 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 C, 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 de C, ou seja, o arquivo CONSTSIC contém todas as declarações de constantes em ENTRADA; TYPESIC contém todas as declarações de tipo; e VARSIC recebe as declarações de variáveis.

As funções e 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.

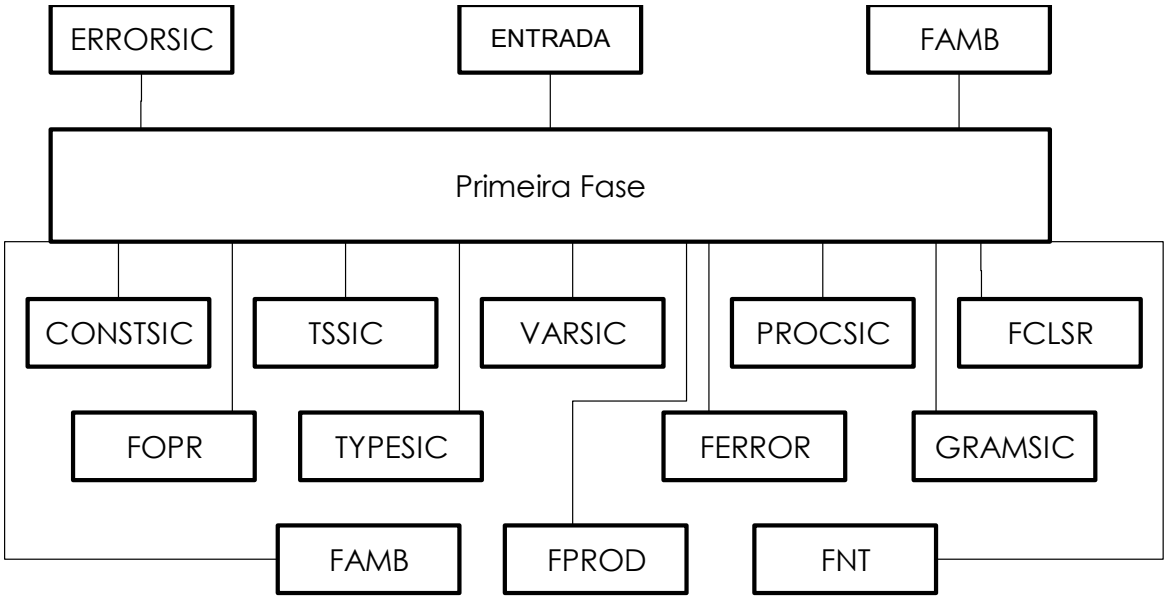


Figura 1: Primeira Fase do SIC

Outras operações realizadas nesta primeira fase dizem respeito a:

a) Leitura e transformação da gramática para uma forma interna;

b) Geração da regra inicial da gramática e subseqüente gravação em GRAMSIC;

c) Determinação de não-terminais que são inatingíveis a partir do símbolo inicial da gramática;

d) Determinação de não-terminais que produzem terminais;

e) Geração do arquivo FPROD contendo a tabela de produções;

f) Geração de constantes que representam símbolos terminais da linguagem;

g) Incorporação na tabela de símbolos TSSIC, das relações de precedência e associatividade associadas aos símbolos terminais da linguagem e às produções;

h) 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 de símbolos e/ou gramática após a rodada anterior.

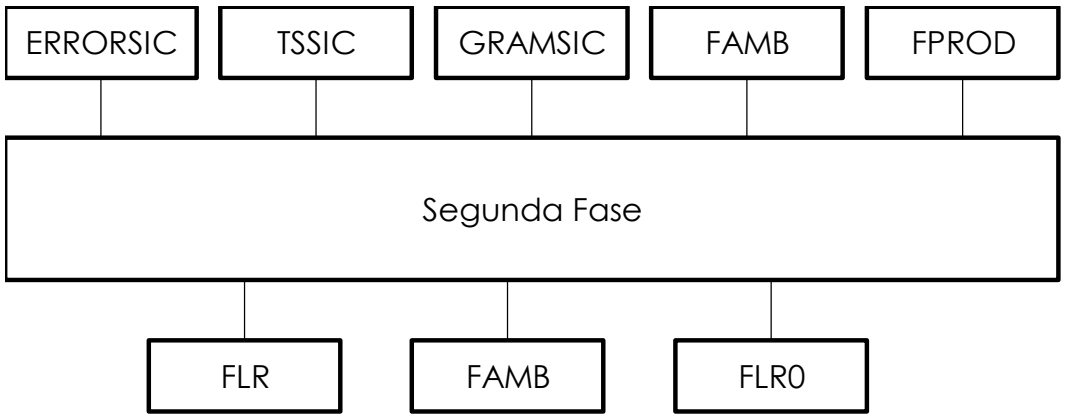


Figura 2: Segunda Fase do SIC

A terceira fase (Figura 3), por sua vez, recebe como entrada:

a) Os arquivos PROCSIC e SEMSIC contendo os procedimentos e as rotinas semânticas da primeira fase;

b) Os arquivos de declarações gerados nas fases anteriores;
c) O arquivo CORPOSIC contendo o corpo do programa fonte;

d) O procedimento do analisador sintático, com ou sem um recuperador automático de erros sintáticos embutido, lido do arquivo PARSERSIC.

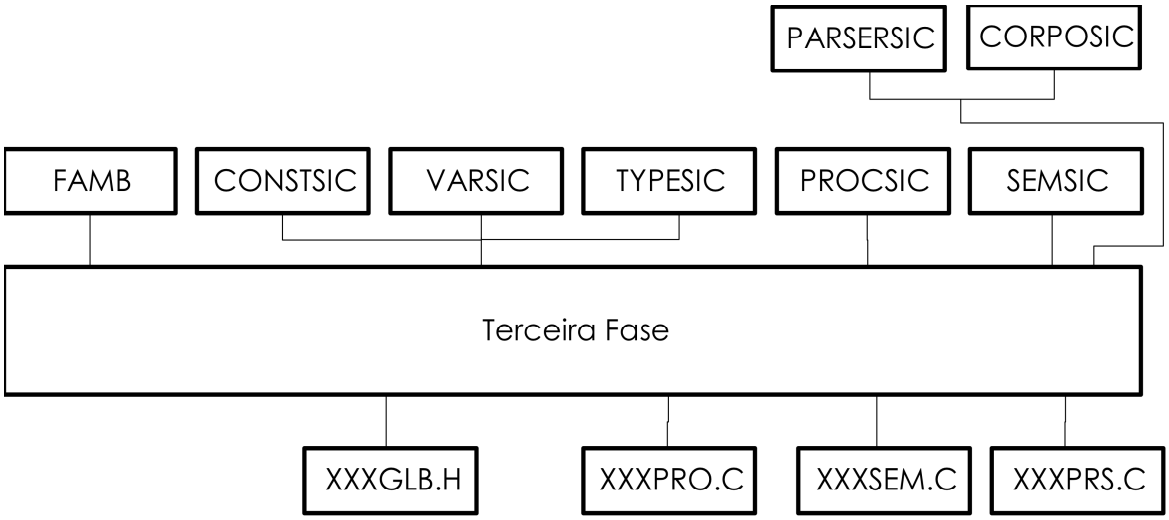


Figura 3: Terceira fase do SIC

Esta fase, entre outras tarefas, 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 C. Este programa é dividido em quatro arquivos gerados da seguinte forma: os arquivos de declarações são combinados em um único arquivo de extensão "h". Neste arquivo são incluídos os protótipos das funções e procedimentos definidos pelo usuário e também aqueles gerados pelo SIC. Os arquivos contendo as rotinas semânticas e os procedimentos já estão em dois arquivos separados, ambos com extensão "c". Os arquivos contendo o analisador sintático e o corpo do programa fonte são unidos em um arquivo também de extensão "c". A formação dos nomes dos quatro arquivos é feita obedecendo o seguinte critéiro: usa-se os três primeiros caracteres do nome do compilador do usuário, ou todos os caracteres se o nome tiver menos que três caracteres, XXX, seguidos pelos sufixos GLB.H, SEM.C, PRO.C ou PRS.C, onde:

a) XXXGLB.H representa o arquivo de definições **gl**o**b**ais: constantes, tipos, variáveis globais e protótipos de funções e procedimentos;

b) XXXSEM.C representa o arquivo de rotinas **sem**ânticas;

c) XXXPRO.C representa o arquivo de **pro**cedimentos e funções;

d) XXXPRS.C representa o arquivo do **p**a**rs**er e do programa principal,

- Exemplos:

1) Compilador: EXEMPLO.SIC
   Arquivos: EXEGLB.H, EXESEM.C, EXEPRO.C, EXEPRS.C

2) Compilador: EX.SIC
   Arquivos: EXGLB.H, EXSEM.C, EXPRO.C, EXPRS.C

3) Compilador: E.SIC
   Arquivos: EGLB.H, ESEM.C, EPRO.C, EPRS.C

Finalmente, para executar o compilador gerado pelo SIC na lingugem C, o usuário deverá compilar cada arquivo e ligá-los usando, para isto, um compilador C que aceite o padrão ANSI (Figura 4). A maneira como esta etapa deve ser feita é diferente para cada compilador C, e, portanto, o usuário deve verificar como os arquivos são compilados e ligados no ambiente em que o compilador estiver sendo utilizado.

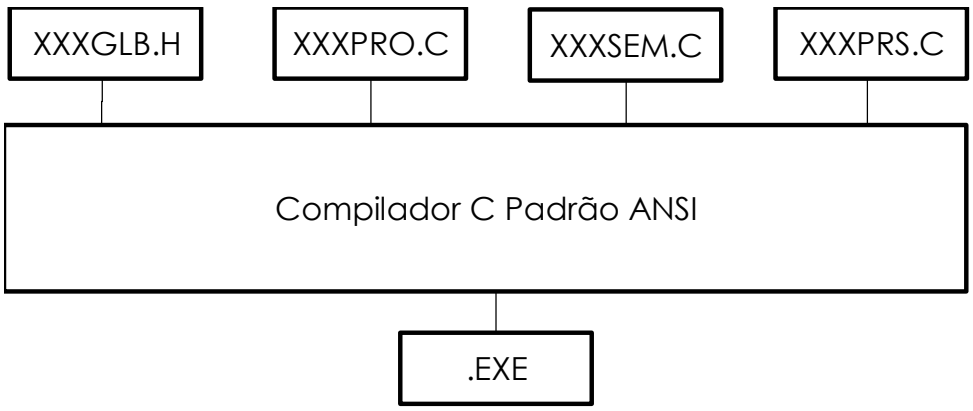


Figura 4: Programa SIC Compilado e Ligado

1.3 EXECUÇÃO DO SIC

1.3.1 Interação com o Usuário

As requisições mínimas necessárias para executar o SIC são:

a) Ambiente operacional WINDOWS 3.1 ou WINDOWS 3.11;
b) Processador 80386 ou superior;
c) 4MB de memória RAM ou superior (recomenda-se 8MB);
d) 1MB livres de disco rígido. Um espaço adicional pode ser necessário dependendo do tamanho da aplicação.

1.3.1.1 Instalação do SIC

Para instalar o SIC proceda da seguinte maneira:

a) Copie os arquivos SIC.EXE, e o arquivo SICPRS0B.DAT em um diretório de sua preferência, por exemplo, no diretório C:\SIC;

b) Dentro do WINDOWS:
- Crie um novo item de programa no item de grupo de sua preferência;
- Coloque SIC para a descrição
- Coloque C:\SIC\SIC.EXE para a linha de comando.

O arquivo SICPRS0B.DAT corresponde ao analisador sintático. Ele deve estar no mesmo diretório em que está o compilador fonte, .SIC. A ausência do analisador sintático interromperá a execução da terceira fase do sistema e a mensagem aparecerá na tela. O usuário deve então sair do sistema e providenciar a cópia do analisador sintático.

Se a opção salvar cores (veja Seção 1.3.1.3) for utilizada, o SIC gera um arquivo denomindado AMBIENTE.DAT. Se este arquivo for apagado a alteração de cores do sistema não fará efeito.

Para a ativação do sistema dê um duplo clique no ícone do SIC.

1.3.1.2 Noções Básicas

A janela principal do SIC (Figura 5) é formada por vários *menus*. O acesso às opções de cada *menu* é feito posicionando o cursor no nome do *menu* e clicando o botão esquerdo *mouse*. Alternativamente, pode-se usar o teclado da seguinte forma: pressiona-se a tecla ALT juntamente com a tecla correspondente ao caracter que estiver sublinhado no nome do *menu.* Não há distinção entre letras maiúsculas e minúsculas, e portanto, pressionar ALT + a causa o mesmo efeito de pressionar ALT + A.

- Exemplos:

Nome do *menu:*	Teclas:
Arquivo	ALT + a
Editar	ALT + e
Compilar	ALT + c
Listar	ALT + l
Opções	ALT + o
Ajuda	ALT + u

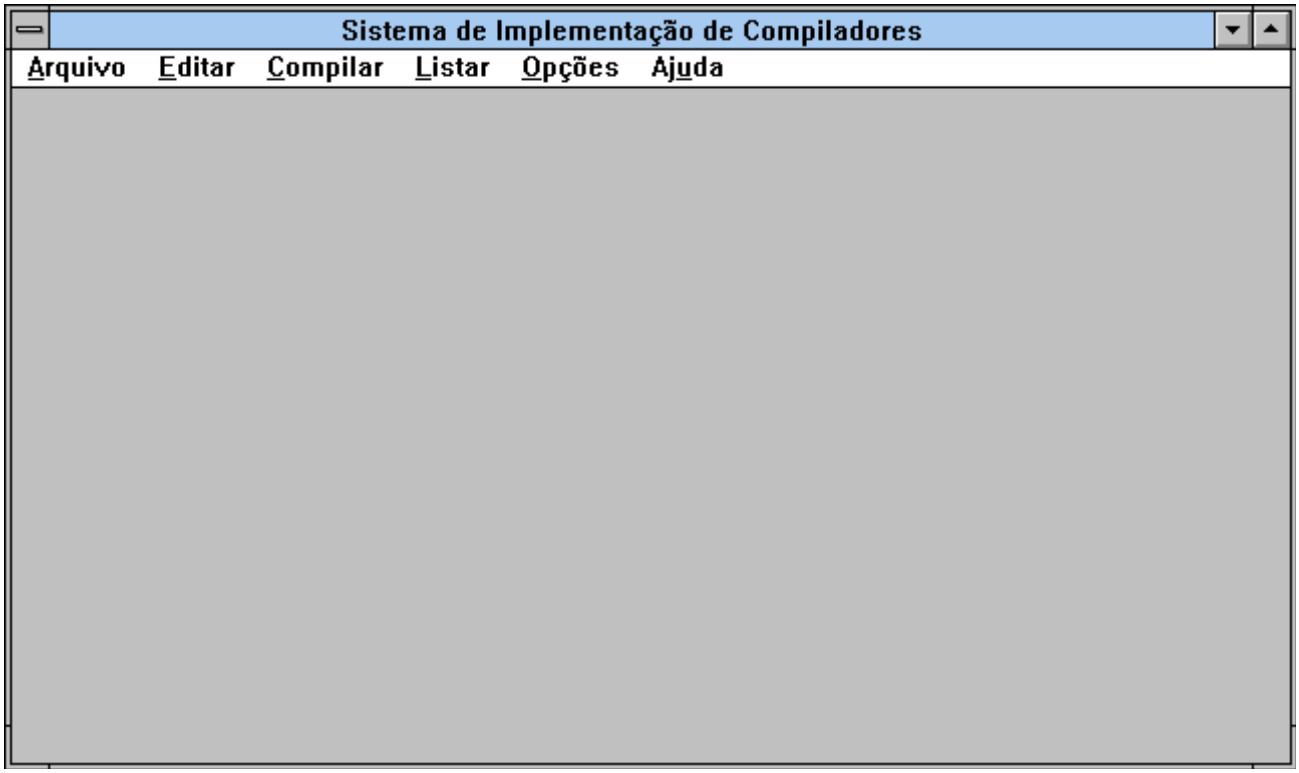


Figura 5: Janela Principal do SIC

Após o acesso a um dos *menus* pode-se também mover de um *menu* para o outro utilizando as setas para-a-esquerda e para-a-direita do teclado.

Ao clicar um menu são exibidas as opções presentes (Figura 6). Para utilizar uma opção basta clicar no seu nome ou então pode-se mover de uma opção para outra utilizando as setas para-cima e para-baixo do teclado e selecionar uma delas pressionando a tecla ENTER.

Figura 6: Um *menu* selecionado

Quando o SIC é iniciado, a janela principal aparece maximizada, ou seja, ocupando toda a tela. Se o usuário desejar restaurar o tamanho original da janela basta clicar o botão restaurar que fica no canto superior direito da janela (Figura 7). Para maximizar novamente a janela basta clicar no botão maximizar (Figura 8) que surge no canto superior direito quando a janela está no seu tamanho original Para minimizar a janela, ou seja, reduzí-la ao tamanho de um ícone, basta clicar no botão minimizar que também fica no canto superior direito.

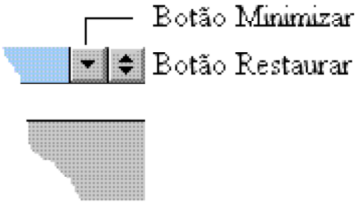


Figura 7: Janela maximizada

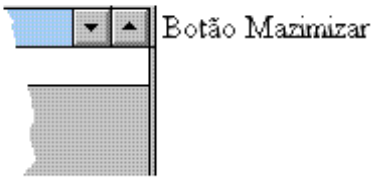
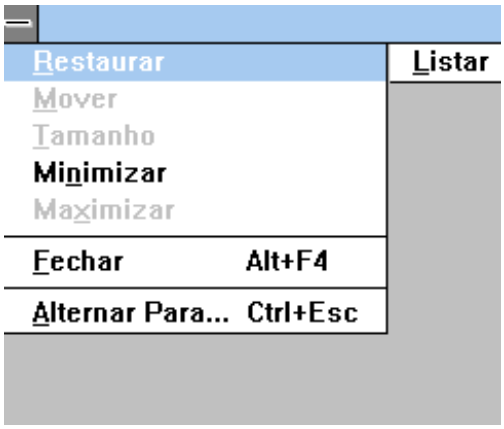


Figura 8: Janela restaurada

Para alternar entre vários aplicativos do WINDOWS, pressione a tecla ALT juntamente com a tecla TAB.



O botão que fica no canto superior esquerdo da janela, denominado botão de controle (Figura 9), é responsável por várias funções como minimizar, maximizar, restaurar a janela, fechar o aplicativo, alternar entre aplicativos do WINDOWS, etc. Para ter acesso a este botão basta clicá-lo do mesmo modo que é feito para um *menu*.

Figura 9: Funções do botão de controle

1.2.1.3 Utilização do SIC

A janela principal do sistema (Figura 5) possui os seguintes *menus*:

a) Arquivo, com a opção Sair;
b) Editar

c) Compilar com a opção Compilar Compilador;
d) Listar com as opções:
- Listar Compilador;
- Listar Gramática;
- Listar Coleção LR(0);
- Listar Tabela LALR(1) compactada;
- Arquivos Gerados;

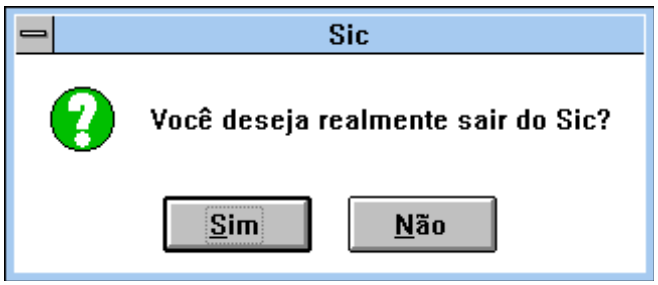
e) Opções com as opções:
- Ambiente;
- Informações sobre LALR(1);

f) Ajuda com as opções:
- Parâmetros;
- Sobre o SIC.

As funções de cada opção dos *menus* estão descritas abaixo.

a) Arquivo

Este *menu* possui uma única opção: Sair. Esta opção é responsável pelo término de uma sessão do SIC. Ao clicar esta opção, surgirá uma janela (Figura 10) confirmando o término da execução do SIC. O usuário deve clicar o botão Sim para sair ou o botão Não para permanecer no sistema.



Pode-se também utilizar o teclado para selecionar um botão: move-se de um botão para outro utilizando a tecla TAB e ao pressionar a tecla ENTER o botão é selecionado.

Figura 10: Confirmação de saída do SIC.

Outra forma de sair do SIC é dando um duplo clique com o botão esquerdo do *mouse* no botão de controle (Figura 11) da janela que fica no canto superior esquerdo ou então selecionar a opção Fechar deste botão (Figura 9).

Figura 11: Botão de Controle

Todas as janelas do sistema podem ser fechadas da mesma forma que a janela principal.

b) Editar

Nesta versão do SIC a opção de edição ainda não foi implementada, portanto ao clicar este *menu* surgirá uma mensagem como ilustra a Figura 12:

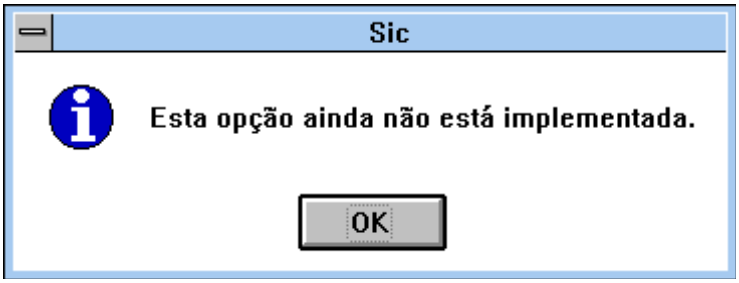


Figura 12

Para retornar à janela principal o usuário deve clicar o botão OK ou pressionar a tecla ENTER.

c) Compilar

Este *menu* possui a opção Compilar Compilador cuja função é executar as três fases descritas na seção 1.2.

Ao selecionar esta opção, seja através do *mouse* ou do teclado, surgirá uma janela (Figura 13) para que o usuário forneça o nome do compilador a ser compilado.

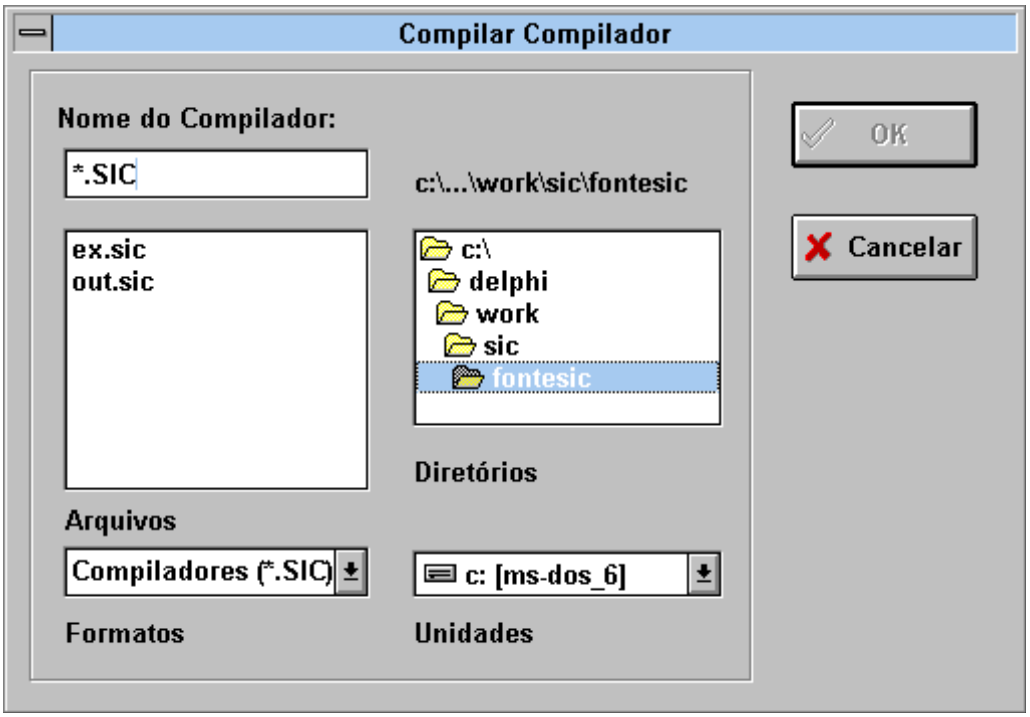
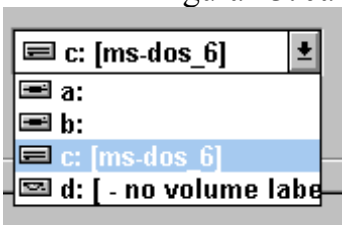


Figura 13: Janela para entrada do nome do compilador a ser compilado



Para saber as unidades disponíveis no sistema, basta clicar na seta da caixa de unidades (Figura 14). O usuário deve então selecionar a unidade onde se encontra o compilador dando um duplo clique sob o nome desta.

Figura 14: caixa de unidades

Para selecionar o diretório deve-se dar um duplo clique no nome do diretório desejado. Se houver algum arquivo com extensão .SIC no diretório selecionado, este aparecerá na caixa de arquivos. Na figura 13 por exemplo, a unidade selecionada é C, o diretório é C:\DELPHI\WORK\SIC\FONTESIC e neste diretório há dois arquivos de extensão .SIC: EX.SIC e OUT.SIC.

O usuário pode selecionar um dos arquivos listados na caixa de arquivos apenas clicando no nome desejado. Pode-se alternativamente digitar o nome do arquivo utilizando o teclado. A tecla TAB serve para mover de um controle para outro (caixas, botões, etc) dentro de uma janela.

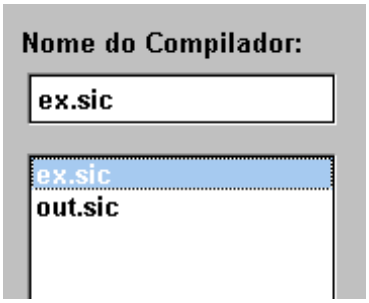


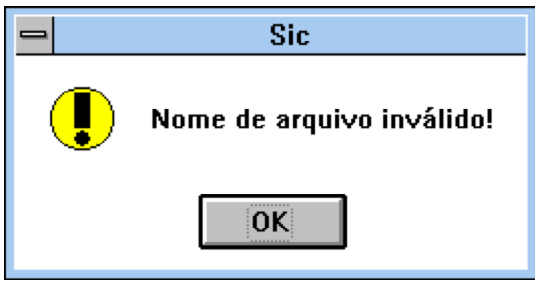
Figura 15

Ao digitar o nome do arquivo o usuário deve seguir as regras abaixo:

a) O nome do arquivo deve conter no máximo oito caracteres;
b) Os caracteres ^ , * , + , = , [,] , ; , : , " , \ , | , , , / , ? , < , > não podem aparecer no nome do arquivo;
c) O nome do arquivo pode ou não ter uma extensão. Se houver extensão esta deve ser SIC.

Para compilar o arquivo selecionado, o usuário deve clicar o botão OK ou pressionar a tecla ENTER. O usuário pode desistir da compilação clicando o botão Cancelar ou pressionando a tecla ESC.

Se o nome do arquivo não estiver correto, ao clicar o botão OK surgirá a seguinte mensagem:



O usuário deve então clicar o botão OK e digitar ou selecionar um arquivo de nome válido.

Figura 16: Nome de arquivo inválido.

Se tudo estiver correto, ao clicar o botão OK na janela Compilar Compilador (Figura 13) surgirá uma janela (Figura 17) indicando que o SIC está executando a primeira fase da compilação (veja Seção 1.2).

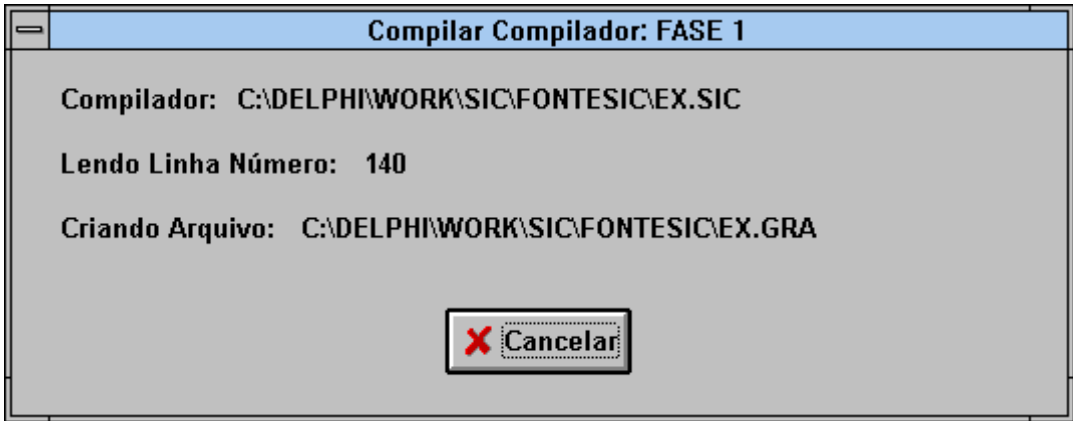


Figura 17: Primeira fase da compilação

A constante que aparece nesta janela indica o número da linha do compilador do usuário que está sendo processado pelo SIC. Nesta Figura, o compilador do usuário se chama EX.SIC e a linha é a de número 140.

O usuário pode cancelar esta fase clicando o botão Cancelar ou pressionando a tecla ESC. Se isto ocorrer, surgirá uma mensagem de confirmação (Figura 18) e o usuário deve clicar o botão Sim para cancelar ou Não se desistir do cancelamento.

Se correr tudo bem na primeira fase e o usuário não tiver cancelado a execução terá início a segunda fase da compilação. Na segunda fase aparecerá uma outra janela (Figura 19) mostrando em que ponto está a execução.

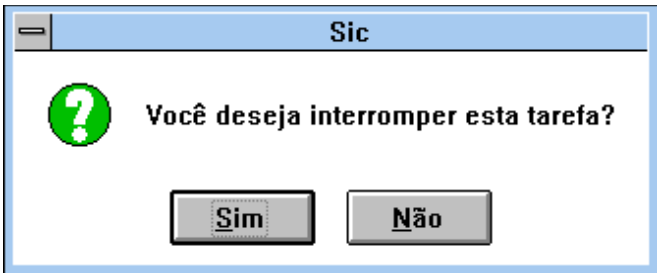
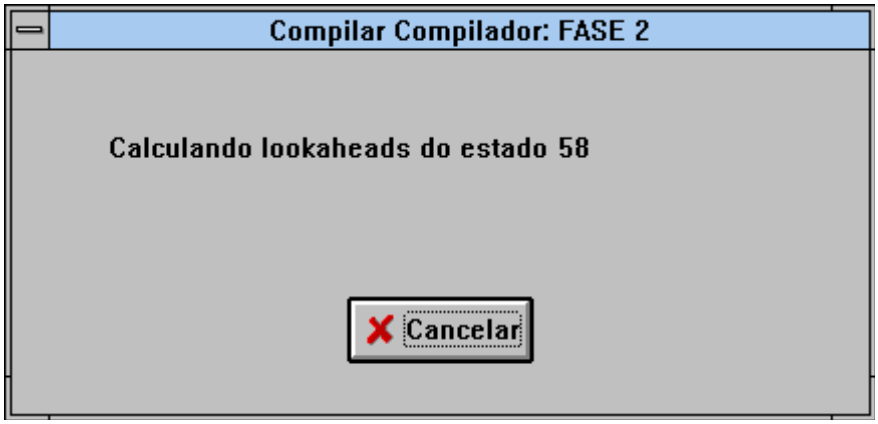


Figura 18



Na figura ao lado, o SIC se encontra calculando os lookaheads do estado 58.

Nesta fase o usuário também pode cancelar a execução.

Figura 19: Segunda fase da compilação.

A janela da terceira fase pode ser vista na Figura 20. Esta janela mostra os arquivos que estão sendo criados para o compilador do usuário. Estes arquivos gerados pelo SIC são escritos no mesmo diretório em que se encontra o compilador que está sendo compilado. O usuário pode mudar a unidade de trabalho do compilador através do *menu* Opções antes de realizar uma compilação.

Figura 20: Terceira fase da compilação

Ao terminar a terceira fase o SIC exibe uma janela (Figura 21) com as informações sobre a geração das tabelas para o compilador. Nesta janela, o usuário pode clicar o botão Arquivos Gerados para saber o nome dos arquivos que foram cridados para o seu compilador (Figura 22). Esta opção também está disponível no *menu* Listar.

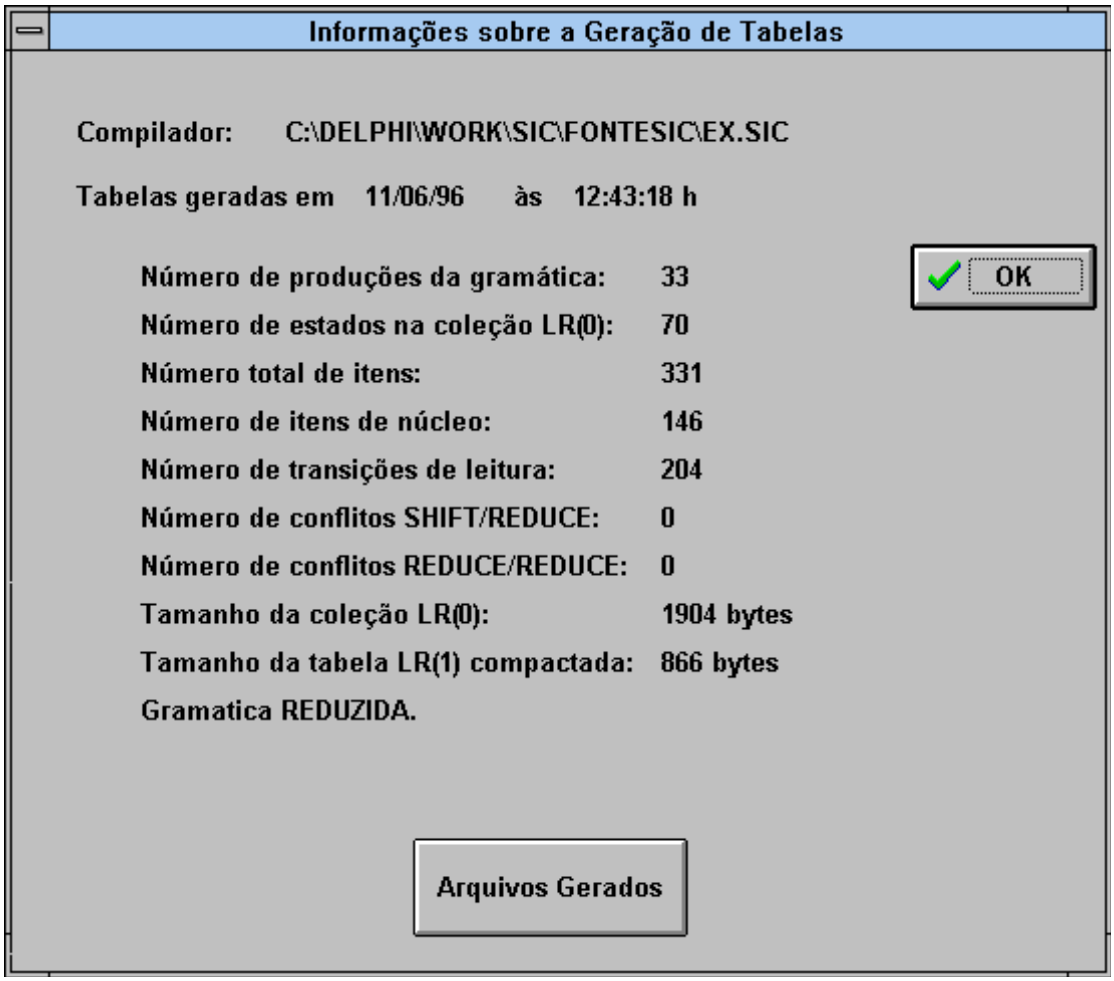


Figura 21: Informações sobre a geração das tabelas

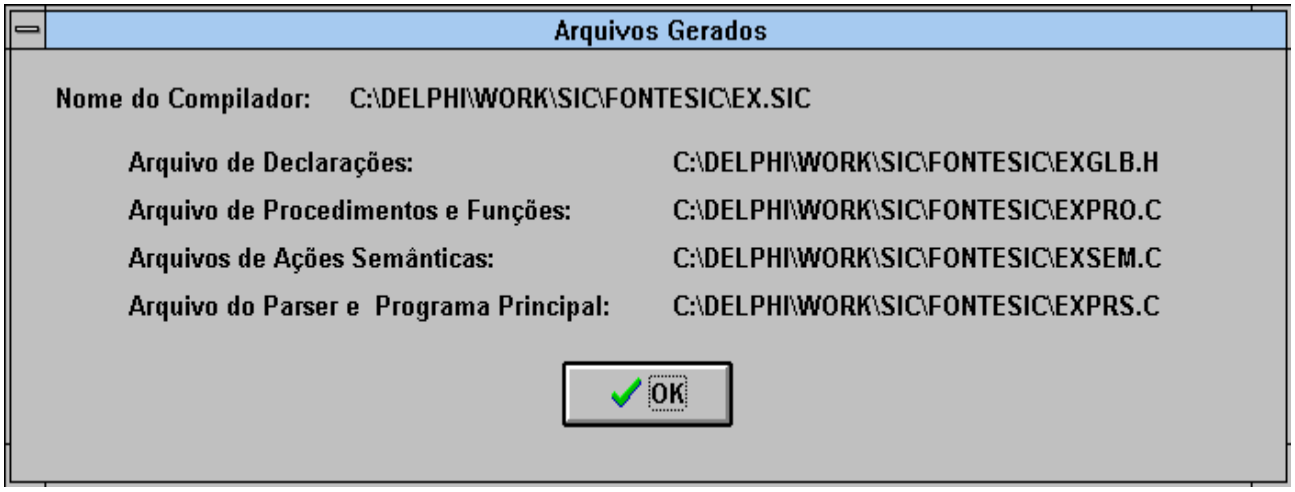


Figura 22: Arquivos gerados para o compilador

d) Listar

Este *menu* possui quatro opções que se referem à listagem em arquivo ou impressora, do programa fonte, da gramática, da coleção canôncia LR(0) e da tabela LALR(1) compactada e da gramática. A quinta opção exibe somente na tela os nomes dos arquivos gerados pelo SIC para o compilador.

Todas estas opções só ficam disponíveis após a compilação de um compilador.

- Opção Listar Compilador

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

Ao clicar qualquer uma destas opções de listagem surge a seguinte janela:

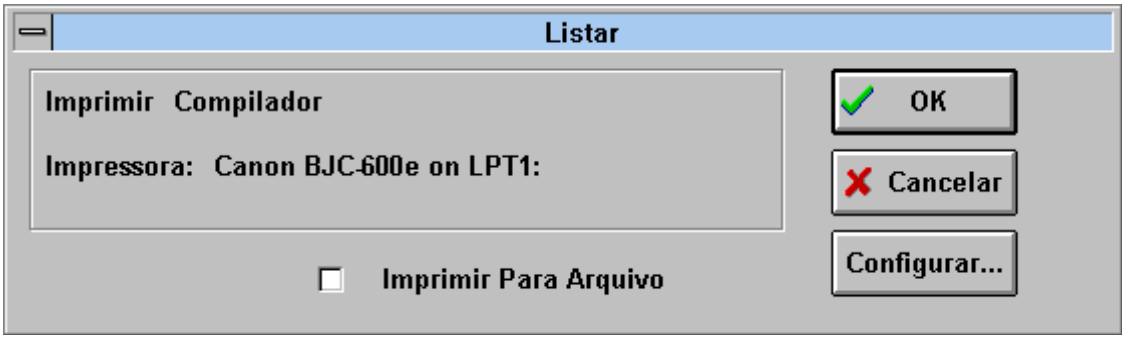


Figura 23: Listar Compilador

O botão Configurar... ao ser clicado exibe uma janela onde o usuário pode definir uma outra impressora, definir o tamanho do papel e a orientação de impressão (retrato ou paisagem). Nesta janela de configuração o botão Opções faz aparecer opções específicas da impressora selecionada como por exemplo, o tamanho do papel, a fonte do papel, cor da impressão, orientação, etc.

Retornando à janela principal de impressão (Figura 23), o usuário poderá clicar o botão OK ou pressionar a tecla ENTER para listar o compilador. Se a opção Imprimir Para Arquivo não estiver selecionada, a listagem será feita para a impressora definida. Na figura 23, a impressora definida é a Canon BJC 600e. Para desistir da impressão basta clicar o botão Cancelar ou pressionar a tecla ESC.

Para gravar o compilador em um arquivo, o usuário deve selecionar a opção Imprimir Para Arquivo clicando no quadradinho mostrado na figura 1.23.

Figura 24

Se esta opção estiver selecionada, ao clicar o botão OK ou pressionar a tecla ENTER a janela de impressão para arquivo surgirá como mostra a Figura 25.

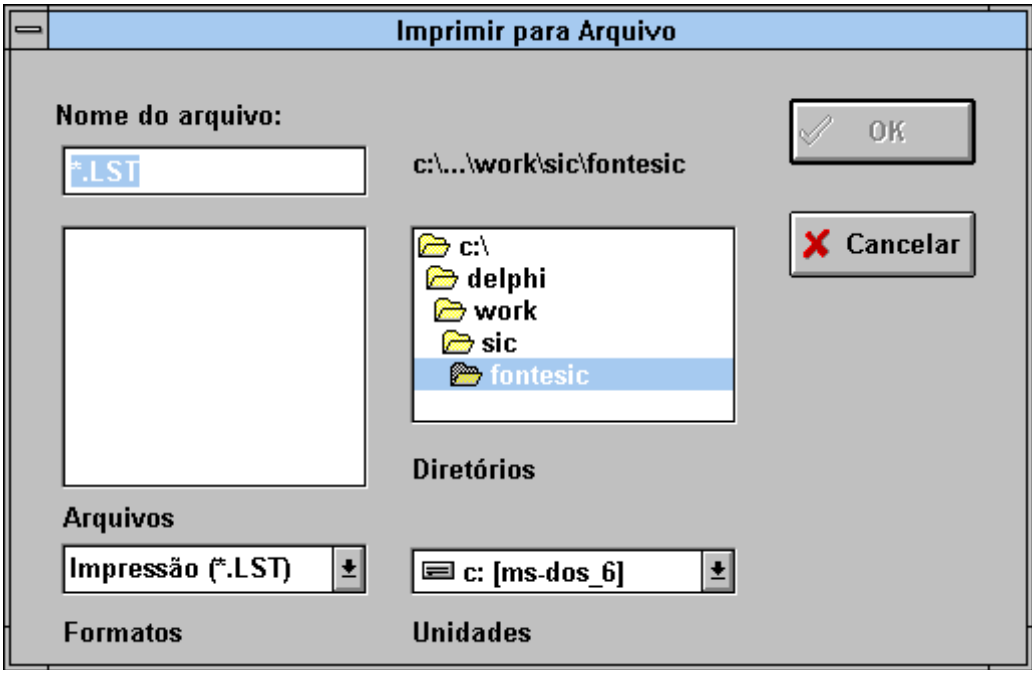


Figura 25: Imprimir para Arquivo

Nesta janela o usuário deverá definir o nome do arquivo de saída, o diretório e a unidade (veja *menu* Compilar para maiores informações sobre definição de diretórios e unidades).

O nome do arquivo de saída deverá seguir as seguintes regras:

a) Deve ter no máximo oito caracteres;
b) Os caracteres ^ , * , + , = , [,] , ; , : , " , \ , | , , , / , ? , < , > não podem aparecer no nome do arquivo;
c) Deve possuir uma extensão de no máximo três caracteres. O SIC sugere a extensão LST, mas o usuário pode usar qualquer extensão que desejar.

Se o usuário selecionar um arquivo da caixa de arquivos, ao clicar o botão OK ou pressionar a tecla ENTER, o SIC perguntará se o arquivo deve ser substituído. Se o usuário clicar o botão Sim o compilador será listado por cima do arquivo já existente. Se clicar

Não retorna-se à janela Imprimir para Arquivo (Figura 25) para a digitação de um novo nome.

O usuário após a definição de um nome pode clicar o botão OK ou pressionar a tecla ENTER para a listagem do compilador para o arquivo. O usuário pode desistir da impressão clicando o botão Cancelar ou pressionando a tecla ESC.

Durante a listagem do compilador seja para o arquivo ou impressora surge uma janela (Figura 26) onde o usuário pode cancelar a listagem clicando o botão Cancelar ou pressionando a tecla ESC.

Figura 26: Listando o compilador

· Opção Listar Gramática

Esta opção é responsável pela listagem, em arquivo ou impressora, da gramática, com as produções numeradas para facilitar a depuração. Além disto ela é responsável pela impressão da tabela de referências cruzadas, 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.

A definição do arquivo ou impressora para a listagem é feita da mesma forma como descrito para a opção Listar Compilador.

· Opção Listar Coleção LR(0)

Esta opção é responsável pela listagem em arquivo ou 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.

A definição do arquivo ou impressora para a listagem é feita da mesma forma como descrito para a opção Listar Compilador.

· Opção Listar Tabela LALR(1) Compactada

Esta opção é responsável pela listagem em arquivo ou impressora, da tabela LALR(1) assim como informações referentes ao seu tamanho.

A definição do arquivo ou impressora para a listagem é feita da mesma maneira como descrito para a opção Listar Compilador.

· Opção Arquivos Gerados

Esta opção é responsável pela exibição na tela dos nomes dos arquivos gerados pelo SIC para o compilador (veja Figura 22).

e) Opções

Este menu possui duas opções: Ambiente e Informações sobre LALR(1).

· Opção Ambiente

Ao clicar esta opção surgirá um *sub-menu* (Figura 27) com as seguintes opções: Unidades, Cores, Salvar Cores.

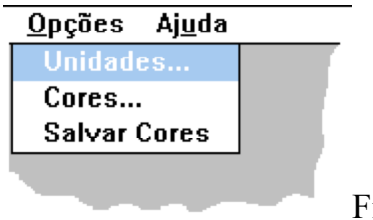


Figura 27

- Opção Unidades

Esta opção permite ao usuário definir novas unidades de trabalho para a compilação. Através da janela de unidades (Figura 28) o usuário pode definir:

a) A unidade de entrada do compilador, ou seja, a unidade onde está o arquivo .SIC;
b) A unidade de trabalho do compilador, ou seja, a unidade onde serão escritos os diversos arquivos gerados pelo SIC;
c) A unidade de saída do compilador, ou seja, a unidade onde serão escritos os arquivos gerados pelo SIC que formam o compilador do usuário.

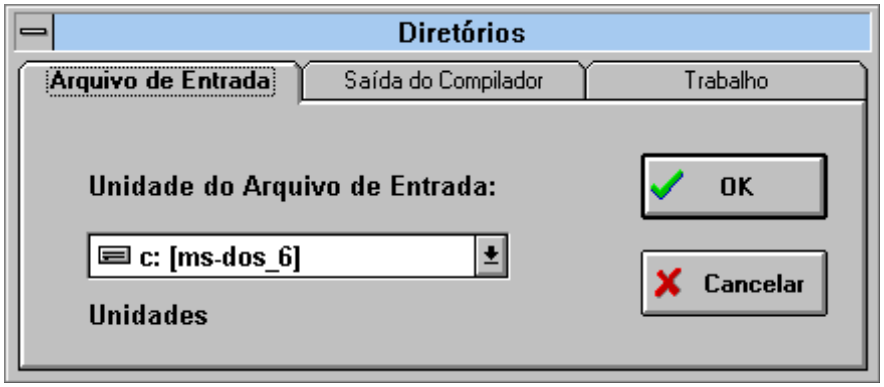


Figura 28: Alteração das Unidades

Para alternar entre as três opções descritas basta clicar no nome da opção, por exemplo, para alterar a unidade de trabalho clique primeiro no nome trabalho. Para saber as unidades disponíveis basta clicar na seta da caixa de unidades e para selecionar uma unidade, basta clicar no nome correspondente.

Ao clicar OK ou pressionar a tecla ENTER, as informações que foram definidas para as três opções são gravadas e utilizadas para os próximos compiladores compilados dentro de uma mesma sessão do SIC, até que as unidades sejam novamente modificadas. Para cancelar a alteração das unidades, basta clicar o botão Cancelar ou pressionar a tecla ESC.

- Opção Cores

O usuário através desta opção pode alterar a cor das janelas, do texto das janelas, do texto dos botões, etc. Na janela de cores (Figura 29) o usuário deve primeiramente selecionar qual objeto deseja alterar a cor e em seguida selecionar uma cor clicando no quadrado correspondente.

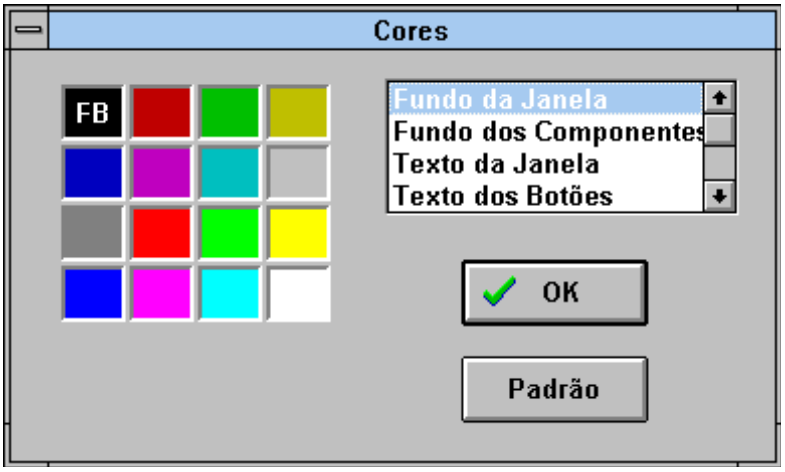


Figura 29: Alteração de Cores

O botão OK faz com que todas as janelas do SIC sejam alteradas. O botão Padrão restaura as cores originais do SIC (padrão do WINDOWS).

- Opção Salvar Cores

Esta função armazena as informações de cores do SIC para que em uma próxima sessão as mesmas cores sejam utilizadas. Esta função só precisa ser utilizada no caso do usuário ter alterado as cores do ambiente utilizando a opção Cores e desejar utilizar sempre as mesmas cores nas próximas sessões do SIC.

Se esta opção não for utilizada, as alterações de cores só valem para a sessão atual.

· Opção Informações sobre LALR(1)

Esta função exibe as informações referentes às tabelas geradas na segunda fase de compilação (veja Figura 21).

Se a gramática do compilador fornecida pelo usuário não for reduzida, outras opções podem aparecer na janela de informações , tais como:

a) Número de conflitos SHIFT/REDUCE ...
b) Número de conflitos REDUCE/REDUCE ...
c) Não-terminais inatingíveis a partir do símbolo de partida =
d) Não-terminais que não produzem strings de símbolos terminais =

f) Ajuda

Este *menu* contém somente as opções Parâmetros e Sobre o SIC.

- Opção Parâmetros

Esta opção indica os parâmetros da versão C 1.0:

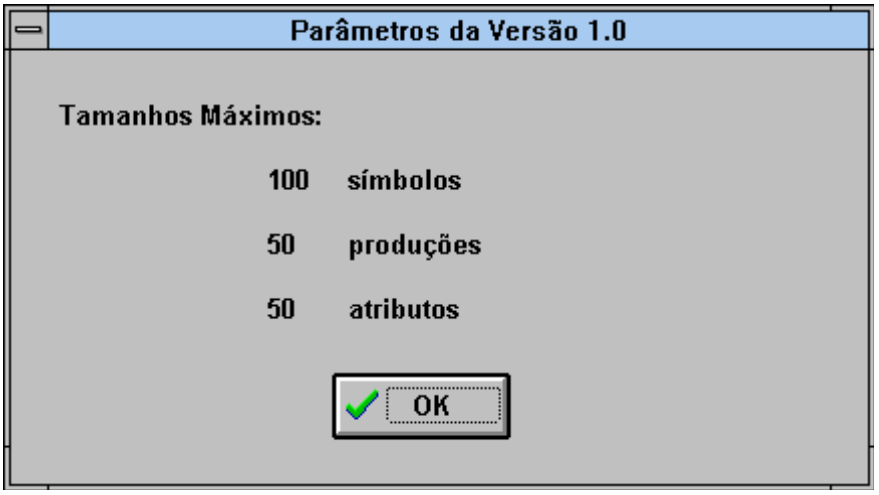


Figura 30: Parâmetros da Versão 1.0

- Opção Sobre o SIC

Esta opção mostra informações sobre a autoria do SIC:

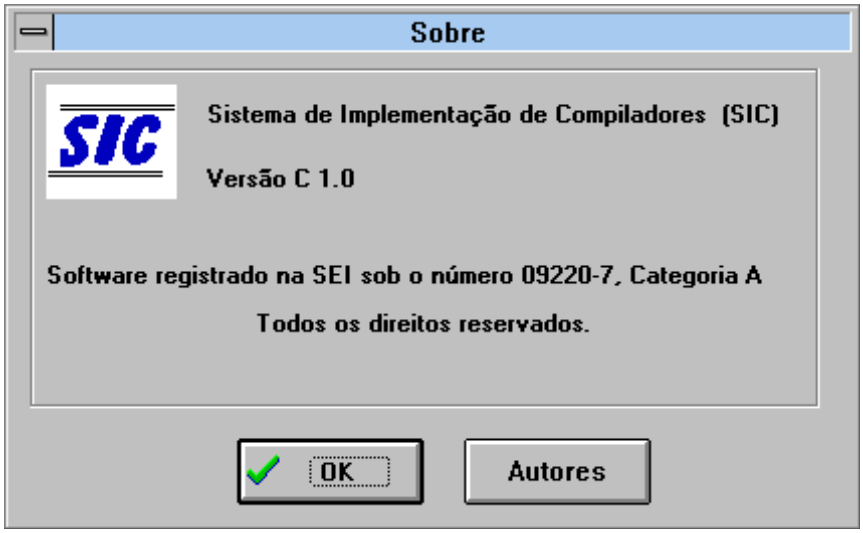


Figura 31

1.3.2 Arquivos do SIC

O nome do arquivo do compilador do usuário é usado na formação dos nomes dos arquivos físicos criados pelo SIC. Se o arquivo fonte for EX.SIC, os demais arquivos terão os seguintes nomes:

NOME LÓGICO	NOME FÍSICO
FAMB	EX.AMB
ENTRADA	EX.SIC
ERRORSIC	EX.ERR
CONSTSIC	EX.CON
TYPESIC	EX.TYP
VARSIC	EX.VAR
CORPOSIC	EX.COR
GRAMSIC	EX.GRA
USELSSNT	EX.SNT
GRAMOCUR	EX.OCR
TSSIC	EX.TAB
YYSAIDA	EX.LR0 EX.LST
YYFLR	EX.LRT
YYFPROD	EX.PRD
YYFOPR	EX.OPR
YYFCLSR	EX.CLS
YYFNT	EX.NTT
PROCSIC	EXEPRO.C
SEMSIC	EXESEM.C
OBJ	EXEGLB.H EXEPRS.C

O arquivo EX.AMB contém informações necessárias para comunicação durante a execução do SIC. O arquivo EX.ERR contém as mensagens de erro do compilador de SIC. No final de cada execução, somente os arquivos de extensão .AMB, .C, .H, .ERR, .SIC, .TAB, .GRA, .LRT, .PRD, .OPR, .CLS, .NTT e .DAT devem ser preservados. Os demais são apagados.

2LINGUAGEM DE PROGRAMAÇÃO DE COMPILADORES (SIC)

2.1 Definição da SIC

2.1.1 Notação

Para descrever a sintaxe de SIC [MBIGONHA, 1985] [MBIGONHA & BIGONHA, 1988] 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 <= n, tem a forma F1 F2 ... Fm (m > 0)

onde cada elemento Fi, 0 < i <= m, pode ser:

a) um símbolo terminal
b) um símbolo não-terminal
c) { T }
d) [T]
e) (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.

```
símbolos_básicos    = token
                    | nt
                    | índice
                    | texto
                    | identificador
                    | palavra_chave ;

token               = """sequência de caracteres diferente de aspas""";

nt                  = identificador ;

índice              = " [ "   número  " ] "  "." ;
```

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 da linguagem C.

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
                    | dígito
                    | "_" ;


dígito              =       "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 inteira representando o código da marca de fim de arquivo (veja Seção 2.1.4).

YYSAIDA

Arquivo do tipo texto, 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) [BIGONHA & MBIGONHA, 1983] e a gramática. O usuário é responsável pela abertura deste arquivo.

YYSIMB

Variável global do tipo inteira 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 inteira 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 inteira 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 inteira que pode assumir os seguintes valores: YYSINTAXEOK, YYERROFATAL, YYERROSINTAXE. É 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.

Existem duas classes de identificadores, identificados pelos prefixos YY e ZZ, que têm seus papéis pré-definidos no SIC:

a) Identificadores com funções específicas no compilador gerado pelo SIC: o usuário deve 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	

b) Identificadores usados pelo analisador sintático YYPARSER, gerado pelo SIC. o usuário não deve declarar estes identificadores para evitar resultados indesejáveis:

ZZpadrao	ZZanalisa	Zzmintopo	Zztopo1
ZZttopo	ZZPSZPE	ZZreduction	Zzratr
ZZs	ZZA	Zzp	Zznumerodeerros
ZZlinha	ZZpos	ZZAcheProximo	ZZPushPE
ZZFazReducoesAtrasadas		ZZInitYYPARSER	

Além das palavras-chave da linguagem C [KERNIGHAN & RITCHIE, 1988], 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 *).

Dentro de texto C, os comentários devem ser delimitados pelo par /* */, mas fora deste contexto somente o par (* *) é permitido.

Números que ocorrem fora de texto C não possuem sinal, não podem conter brancos entremeados e são sempre do tipo inteiro.

```
número = dígito { dígito } ;
```

2.1.3 Programa em SIC

A especificação de um programa em SIC é a seguinte:

```
SIC =   "%%COMPILER"  [texto]  opções [passos]
            seção_um
            [seção_dois]
            corpo
        "%%END"
```

O texto após o a palavra chave %%COMPILER pode ou não conter a palavra "main" ou o nome de um procedimento. A presença de "main" indica que o compilador gerado será um programa principal em C, enquanto o nome de um procedimento indica que o compilador gerado será um procedimento. Às vezes é conveniente declará-lo como procedimento, pois quando o programa se torna muito grande, é mais fácil depurá-lo isoladamente. Se não houver texto após a palavra-chave %%COMPILER o SIC assume que o compilador será um programa principal.

- Exemplos:

```
%%COMPILER main %%BATCH
%%COMPILER MeuCompilador  %%BATCH
%%COMPILER %%BATCH
```

Se o usuário desejar incluir o compilador como procedimento em um programa principal existente, deve incluir no mesmo as seguintes linhas, nesta ordem:

```
#define OUTROS 1
#include "XXXGLB.H" /* arquivo header de globais */
```

onde #define OUTROS 1 garante que certas variáveis inicializadas, geradas pelo SIC, sejam declaradas corretamente.

- Exemplo:

Nome do fonte SIC: EX.SIC
Linhas a serem inseridas:
```
#define OUTROS 1
#include "EXGLB.H"
```

A Seção 4 apresenta um exemplo completo de um programa em SIC.

2.1.3.1 A Seção Opções de Compilação

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

```
opções =    "%%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.

Somente as opções %%BATCH estão implementadas na versão C.

2.1.3.2 A Seção 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 main %%BATCH
%%COMPILER MeuCompilador %%INTERACTIVE NOREC
%%COMPILER main %%BATCH %%FIRST PASS
%%COMPILER %%INTERACTIVE REC %%FIRST PASS
%%COMPILER %%BATCH %%OTHER PASS
%%COMPILER main %%OTHER PASS
```

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

2.1.3.3 A Seção Seção_Um

A Seção seção_um tem por objetivos:

a) Especificar os símbolos terminais da linguagem sendo implementada;

b) Declarar os atributos associados aos símbolos não-terminais e terminais da gramática dessa linguagem;

c) Declarar os mapas de escopo, isto é, os símbolos terminais que atuarão como abridores de escopo e seus respectivos fechadores (veja Seção 2.1.6);

d) Declarar os símbolos não-terminais que poderão ser inseridos ou substituídos na recuperação de erro;

e) Especificar as declarações da linguagem C.

```
seção_um    =    terminais | [atributos] | [mapadeescopo]
             |    [ntmapa] | [declarações] ;
```

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 C podem 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 C, a definição desses terminais deve aparecer antes, para diferenciar texto escrito entre aspas de símbolos terminais (veja seção 2.1.8).

2.1.3.4 A Seção Seção_Dois

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

seção_dois = gramática resolução_de_conflitos ;

2.1.4 A 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 fornece o tipo léxico da marca de final do texto de entrada.

- Exemplo:

```
%%TOKENS
    "id"      = xid;
    "cte"     = xcte;
    "begin"   = xbegin;
    "eof"     = YYEOF;
    "<"       = xmenor;
    "*"       = xvezes
```

```
";"      = xpv
"end"    = xend
"("      = xap
")"      = xfp
```

2.1.5 A Seção de Atributos

O SIC implementa o esquema clássico de geração pós-fixada de código dirigida por sintaxe [AHO & ULLMAN, 1972]. Neste método, aos símbolos da gramática devem ser associados atributos sintetizados [KNUTH, 1968, 1973] os quais têm seus valores definidos à 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 em uma 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" [número] "OF" "ATTRIBUTES" atributo ";" ;

atributo        = def_atrib { def_atrib } ;

def_atrib       = símbolo "=" "(" lista_de_atrib ")"
                  [ decl_ação ] [ ";" ];

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_ação       = " { " [ texto ] " } " ;

símbolos        = nt
                | token ;
```

- Exemplos:

```
%%STACK 200 OF ATTRIBUTES

exp   = (R,tipo : int)   { exp.r = 0; exp.tipo = 0;};

"id"  = (valor  : int)   { GERATEMPORARIO(t); "id".valor = INSTALA(T); };

"cte" = (valor  : int)   { "cte".valor = 0; };

"*"   = (P      : alfa);
```

```
cmd    =  (B,XX,K : unsigned int);
"+"    =  (KK,ZZ  : alfa)
dcl    =  (DD     : int; A,B,C : char)
cond   =  (quad   : int)      { 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 normalmente, mesmo após erros de sintaxe haverem sido detectados durante a compilação.

Todo símbolo, terminal ou não-terminal possui implicitamente 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 A Seção de Abridores e Fechadores de Escopo

A Seção mapadeescopo tem por finalidade prover informações úteis à recuperação de escopo. [MBIGONHA & BIGONHA, 1995]

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 A Seção Mapa de Não-Terminais

A Seção mapa de não-terminais, ntmapa, 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. [MBIGONHA & BIGONHA, 1985]

```
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 A Seção de Declarações

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

```
declarações  =  membros  ;

membros      =    "%%TYPES"      declaração de tipos
             |    "%%CONSTANTS"   declaração de constantes
             |    "%%VARIABLES"   declaração de variáveis
```

| "%%PROCEDURES" declaração de procedimentos ;

declaração de tipos = texto ;
declaração de constantes = texto ;
declaração de variáveis = texto ;
declaração de procedimentos = texto ;

- Exemplos

```
%%CONSTANTS
        #define TMID 8
        #define A 2
        #define B 3
%%VARIABLES
        alpha x;
        int y;
%%TYPES
        typedef char alpha[TMID];
%%PROCEDURES
        void YYSCAN(void)
        {       ...
        }
        int INSTALA( ... );
        {       ...
        }
        int PROCURA( ... );
        {       ...
        }
%%VARIABLES
        char ch;
```

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 deve ser um procedimento sem parâmetros que será 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, 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 "int", YYSCAN poderá retornar o valor *inum* de uma constante inteira lida mediante a atribuição

"cte".valor = inum ;

- Exemplo:

```
void GETCHAR(char *c)
{
 if (cc == ll)
 {
        ...
         ll = 1;
         cc = 0;
         fscanf(FONTE,"%c",&linha[ll]);
         while (linha[ll] != '\n')
         {
          ll++;
          fscanf(FONTE,"%c",&linha[ll]);
         }
         ll++;
         linha[ll] = NEWLINE;
         fscanf(FONTE,"\n");
         YYLINHA++;
 }
 cc++;
 *c = linha[cc];
} /*GETCHAR*/

void YYSCAN(void)
{
        ...
        YYPOS = cc;
        if (isalpha(ch))        /* palavra */
        {
                ...
                YYSIMB = xid;
                "id".valor = INSTALA(ident);
        }
        else    if (isdigit(ch))    /* numero */
                {
                        ...
                        YYSIMB = xcte;
                        "cte".valor = inum;
                }
                else    if (ch==';')
                        {
                                GETCHAR(&ch);
                                YYSIMB = xpv;
                        }
        ...
} /*YYSCAN*/
```

2.1.9 A 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.

```
gramática      = "%%GRAMMAR"  nt  [ "AND" "SEMANTICS"]  produções ;

produções      =  nt  "="   def_prod  { "|" def_prod} ";"  ;

def_prod               = ldireito  [ precedência ]
                           [ rotina_semântica ];

ldireito               =  { símbolos };

precedência    =  "%%PREC"   token  |  "%%PREC" identificador ;

rotina_semântica      =  " { "   [ 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:

```
EMPTY = ;
```

A rotina ou ação semântica, associada a cada produção é definida por um ou mais comandos em C colocados entre chaves, e como tal, pode fazer entrada e saída, invocar procedimentos, alterar valores de variáveis e vetores externos; salvar tabela de símbolos, etc. É importante ressaltar aqui dois pontos:

a) Os comentários dentro deste contexto devem ser delimitados pelo par /* */.
b) O usuário deve estar atento para o uso de "{" (abre-chaves) e "}" (fecha-chaves) dentro do texto C. Se houver um "{" ou um "}" não balanceado pode haver má interpretação quanto a delimitação da ação semântica (veja Seção 2.1.12).

· Exemplo de regras com ações semânticas associadas:

```
a) cmd = "id"  ":="  exp
            {
             classe = ts["id".valor].classe;
             if ((FOIDECLARADO("id".valor)) && ((classe==VARIAVEL) ||
                    ( classe == PAR)))
                    GEN(CATRIB,"id".valor,exp.r,0);
             else ERRO(7,"id".YYLINHA,"id".YYPOS);
            };

b) exp  =  "cte"
            {
             exp.r ="cte".valor;
             exp.tipo = INTEIRO;
            };
```

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 ambigüidades, 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:

a)
```
exp = exp + exp
    {
    if (exp[2].tipo != exp[3].tipo)
        ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
    if (exp[2].tipo == INTEIRO)
        op = CMAIS;
    else op = COU;
    temporario = TEMP();
    GEN(op,temporario,exp[2].r,exp[3].r);
    exp.r = temporario;
    exp.tipo = exp[2].tipo;
    };
```

b)
```
exp = exp * exp
    {
    if (exp[2].tipo != exp[3].tipo)
        ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
    if (exp[2].tipo == INTEIRO)
        op = CVEZES;
    else op = CE;
    temporario = TEMP();
    GEN(op,temporario,exp[2].r,exp[3].r);
    exp.r = temporario;
    exp.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 A 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 & ULLMAN, 1979]. Entretanto, várias construções sintáticas comuns em linguagens de programação 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 & ULLMAN, 1977] e [AHO, SETHI & ULLMAN, 1986].

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.

```
resolução_de_conflito       = "%%CONFLICTS" resoluçãoconflitos
                              { [ ";" ] resoluçãoconflitos}
                              [ ";" ] ;
resoluçãoconflitos  = associa_se_a símbolos
                              { "," símbolos };
associa_se_a        = "%%RIGHT"
                    |  "%%LEFT"
                    |  "%%NONE"
```

A declaração de precedência e associatividade é feita através de uma série de claúsulas começando com as palavras chaves %%RIGHT, %%LETF 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 três 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 resolução de conflitos, e use tal identificador na cláusula %%PREC correspondente.

- Exemplo:

```
%%CONFLITS

        %%LEFT "+" "-"
        %%RIGHT WW
        %%LEFT "*" "/"

exp = exp "+" exp %%PREC WW
```

O exemplo acima indica que a regra da gramática, exp = exp "+" exp terá uma precedência (a do identificador WW) maior que "+" e menor que "*".

2.1.11 A Seção 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 C, exigindo-se apenas que o procedimento YYPARSER seja ativado em algum ponto.

2.1.12 Erros na Especificação da Sintaxe do Compilador

Na presença de um erro, o usuário poderá utilizar a função Listar Compilador, disponível no menu principal. Esta função listará o programa fonte no disco ou impressora juntamente com as mensagens de erro. Porém, há certos tipos de erros oriundos de uma especificação mal formada, que não geram mensagens e outros que geram mensagens não muito explicativas. Alguns destes erros são descritos nesta seção.

O primeiro deles ocorre quando o usuário coloca um "{" (abre-chaves) ou um "}" (fecha-chaves) não balanceado dentro de uma rotina semântica (veja.seção 2.1.9). Quando isto acontece não há meios de distinguir no SIC quais são os delimitadores da rotina semântica e quais são os "{" e "}" do programa em C. Assim, muito embora sejam geradas mensagens de erro, as mesmas não mostram que o erro ocorrido foi este. Portanto, o usuário deve estar atento para o uso correto dos pares "{" e "}". O exemplo abaixo mostra como um erro deste pode ocorrer:

```
%%GRAMMAR program AND SEMANTICS

program = proghead dcls cmdc
          {
                {                                   /* ERRO!!! */
                SALVA(1,proxq-1);
                GEN(CPROGEND,0,0,0);
          };
```

O trecho abaixo foi extraído da listagem do programa fonte utilizando a função Listar Compilador:

```
...
844  %%GRAMMAR program AND SEMANTICS
     845
     846
     847  program  = proghead dcls cmdc
     848              {
     849                   {
     850                   SALVA(1,proxq-1);
     851                   GEN(CPROGEND,0,0,0);
     852              };
     853
     854  proghead = "program"
     855              {
     856                   toff   = 0;
     857                   offset  = 0;
     858                   proxq   = 1;
     859                   GEN(CPROGBEGIN,0,0,0);
     860              };
     861
     862  dcls  = dcl
     863            {   }
     864     |  dcls ";" dcl
     865            {   } ;
     866
     867  dcl  = "id" ":" "integer"
     868       {
     869            DECLARA("id".valor,VARIAVEL,INTEIRO,offset);
    ----------------------------^
    <<Estourou o indice>>
     870            offset++;
     871       };
     872
     873  dcl  = "id" ":" "boolean"
     874       {
     875            DECLARA("id".valor,VARIAVEL,LOGICO,offset);
    ----------------------------^
    <<Estourou o indice>>
     876            offset++;
     877       };
     878    ...
```

Podemos notar que a mensagem de erro gerada “Estourou o Índice” não fornece pistas quanto ao real motivo do erro.

b) Existe outro tipo de erro que pode ocorrer pelo mesmo motivo: "{" ou "}" não balanceados em uma função definida pelo usuário. O problema neste caso é mais sério porque não há geração de nenhuma mensagem de erro e o programa gerado pelo SIC parece então estar correto. Mas ao compilá-lo em C, algumas mensagens de erro podem ocorrer.

Este problema ocorre devido ao seguinte fato: se houve um "{" ou "}" não balanceado, os protótipos das funções definidas pelo usuário não são escritos corretamente no arquivo de definições globais. Assim, se houver algum problema ao compilar o programa em C, e o SIC não tiver reportado erros de sintaxe na entrada, o usuário deve verificar se os "{" e "}" estão balanceados.

- Exemplo:

```
...
%%PROCEDURES

    void ERRO(int n,int l,int p)
   {
          mensg message;
          fseek(out,sizeof(mensg)*n,SEEK_SET);
          {                                                    /* ERRO!!! */
          fread(&message,sizeof(mensg),1,out);
          fprintf(YYSAIDA,"\n");
          fprintf(YYSAIDA,"%5c%s%s%d%s%d\n",'+',message.msg,"na linha", l, "posicao",
                p);

   } /* ERRO */
...
```

Ao compilar o arquivo EXSEM.C são geradas as mensagens abaixo:

```
...
Compiling EXSEM.C:
Warning EXSEM.C 9: Call to function 'SALVA' with no prototype
Warning EXSEM.C 10: Call to function 'GEN' with no prototype
Warning EXSEM.C 19: Call to function 'GEN' with no prototype
Warning EXSEM.C 34: Call to function 'DECLARA' with no prototype
Warning EXSEM.C 42: Call to function 'DECLARA' with no prototype
Warning EXSEM.C 50: Call to function 'TAMANHO' with no prototype
Warning EXSEM.C 51: Call to function 'POPOFF' with no prototype
Warning EXSEM.C 53: Call to function 'GEN' with no prototype
Warning EXSEM.C 54: Call to function 'FBLOCO' with no prototype
Warning EXSEM.C 56: Call to function 'SALVA' with no prototype
Warning EXSEM.C 65: Call to function 'DECLARA' with no prototype
Warning EXSEM.C 73: Call to function 'DECLARA' with no prototype
Warning EXSEM.C 81: Call to function 'DECLARA' with no prototype
Warning EXSEM.C 85: Call to function 'GEN' with no prototype
Warning EXSEM.C 89: Call to function 'ABLOCO' with no prototype
...
```

Dependendo de onde está o "{" ou o "}" não balanceado pode ocorrer outros tipos de mensagens de erro, inclusive no arquivo de definições globais.

c) Outro problema que não gera mensagens e o erro surge na compilação do programa em C ocorre quando falta um símbolo % em uma das palavras-chaves das seções de declaração: %%CONSTANTS, %%TYPES, %%VARIABLES ou %%PROCEDURES. Como no caso anterior, o programa parece estar correto, mas na compilação há a geração de mensagens de erro.

- Exemplo:

```
...
%%CONSTANTS

    #define MAXMSG 20
    #define MAXERROR 11

%TYPES

/*  ERRO!!! FALTA UM % NA PALAVRA-CHAVE %%TYPES!!! */

    typedef char erros[MAXMSG+1];
    typedef struct { erros msg; } mensg;

%%VARIABLES

    FILE *out;
...
```

Ao compilar o arquivo EXGLB.H as seguintes mensagens são geradas:

```
Compiling EXGLB.H:
Error EXGLB.H 62: Declaration terminated incorrectly
Error EXGLB.H 65: Declaration missing ;
```

O usuário deve estar atento para estas possíveis fontes de erro, pois caso contrário pode ser difícil saber o que está errado.

3. REFERÊNCIAS BIBLIOGRÁFICAS

[AHO & ULLMAN, 1972]
AHO,A.V.,Ullman,J.D., The Theory of Parsing Translation and Compiling, Prentice-Hall,INC,1972.

[AHO & ULLMAN, 1977]
AHO, A.V., Johnson,S.C., and Ullman, J.D. Deterministic Parsing of Ambiguous Grammars. Comm. Assoc. Comp. Mach. vol. 18(8) 441-452 , August 1977.

[AHO, SETHI & ULLMAN, 1986]
AHO, A.V., Sethi R. S. & Ullman,J.D., Compilers, Principles, Techniques and Tools, Addison-Wesley Publishing Company Co. (1986).

[MBIGONHA, 1985]
BIGONHA, Mariza A. S., SIC : Sistema de Implementação de Compiladores, Tese de Mestrado, Departamento de Ciência da Computação (DCC) Icex-UFMG, junho/1985.

[BIGONHA & MBIGONHA, 1983]
BIGONHA, R.S. & BIGONHA, M.A.S., Um Método de Compactação de Tabelas LR(1), Anais do III Seminário sobre Software Básico para Micros, Rio de Janeiro, 1983.

[MBIGONHA & BIGONHA, 1985]
BIGONHA, Mariza A.S.& BIGONHA, Roberto S., Uma Experiência na Implementação de um Recuperador de Erro LR(1), Anais do V Simpósio sobre Desenvolvimento de Software Básico, Belo Horizonte, 1985.

[MBIGONHA & BIGONHA, 1988]
BIGONHA, Mariza A.S & 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.

[MBIGONHA & BIGONHA, 1989]
BIGONHA, Mariza A. S. & BIGONHA, Roberto S. SIC: Sistema de Implementação de Compiladores - Manual do Usuário versão 4.0 . Relatório Técnico RT/007-89, Departamento de Ciência da Computação, ICEx, UFMG.

[HOPCROFT & ULLMAN, 1979]
HOPCROFT, John E. & ULLMAN, Jeffrey D., Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Series in Computer Science, 1979.

[KERNIGHAN & RITCHIE, 1988]

KERNIGHAN, Brian W. & RITCHIE, Dennis M. The C Programming Language 2 ed. Prentice Hall, 1988.

[KNUTH, 1968]

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, 1973]

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

4. EXEMPLO

```
%%COMPILER PROGRAM exemplo;   %%BATCH


%%TOKENS

    "id"            = xid ;
    "cte"           = xcte ;
    "program "      = xprogram
    "end"           = xend
    "begin"         = xbegin
    "integer"       = xinteger
    "procedure"     = xprocedure
    "if"            = xif
    "then"          = xthen
    "else"          = xelse
    "while"         = xwhile
    "do"            = xdo
    ";"             = xpv
    ":"             = xdp
    ":="            = xatr
    "("             = xap
    ")"             = xfp
    "+"             = xmais
    "-"             = xmenos
    "eof"           = YYEOF
    "*"             = xvezes
    "/"             = xdiv
    "**"            = xpot
    "="             = xigual
    "error"         = xerror
    "boolean"       = xboolean


%%STACK 50  OF ATTRIBUTES


    exp       = (r,tipo:int)  { exp.r = 0; exp.tipo = INTEIRO; };
    "id"      = (valor :int)  { "id".valor = 1; }
    "cte"     = (valor :int)
    cond      = (quad  :int)
    n         = (quad  :int)
    marca     = (quad  :int)
    prochead  = (valor :int; inicio : int)
```

```
%%SCOPEMAP

    "begin" : "end" ;
    "("     : ")"   ;

%%NTMAP

    exp , dcl , cmd ;



(*=================================================================*)
(*                        MODULO DE TRATAMENTO DE ERRO             *)
(*=================================================================*)

%%CONSTANTS

        #define MAXMSG 20
        #define MAXERROR 11

%%TYPES

        typedef char erros[MAXMSG+1];
        typedef struct { erros msg; } mensg;

%%VARIABLES

        FILE *out;

%%PROCEDURES

    void ERRO(int n,int l,int p)
        {
                mensg message;

                fseek(out,sizeof(mensg)*n,SEEK_SET);
                fread(&message,sizeof(mensg),1,out);
                fprintf(YYSAIDA,"\n");
                fprintf(YYSAIDA,"%5c%s%s%d%s%d\n",'+',message.msg,"na linha", l,
                        "posicao", p);
        } /* ERRO */
```

```
/*=================================================================*/
/*                    MODULO TABELA DE SIMBOLOS                    */
/*=================================================================*/


%%CONSTANTS

        /* tipo tabela de simbolos */

        #define INTEIRO 1
        #define LOGICO 2

        /*  classe tabela de simbolos */

        #define NAO_DECL 0
        #define VARIAVEL 1
        #define PAR 3
        #define PROC 4
        #define NMAX 29
        #define MAX 1000
        #define ALPHA1 9
        #define NPC 10
        #define IDTAM 15        /* no. caracteres significativos ident */
                                /* a posicao 0 de ident nao contem nada.*/

%%TYPES

        typedef char alfa[IDTAM+1];
        typedef char alfa1[ALPHA1+1];

        typedef struct {    alfa nome;
                            int classe;
                            int tipotam;
                            int endoff;
                            int nivel;
                            int col;
                       } tscampos;

%%VARIABLES

        /* tabela de simbolos */

        int nivel,l;
        tscampos ts[MAX+1];
        int thash[91];
        int escopo[NMAX+1];
```

```
%%PROCEDURES

        int HASH(alfa simb)
        {
         int i,h;

         h = 0;
         i = 1;
         while ( (simb[i] != ' ') && (i < IDTAM))
         {
                 h = h + simb[i];
                 i = i + 1;
         }
         return h % 91;
        }  /*HASH*/


        int INSTALA(alfa simb)
        {
         int n,k;

         n = HASH(simb);
         k = thash[n];
         while (k >= escopo[nivel])
         {
                 if (strcmp(simb,ts[k].nome)==0)
                 {
                  ERRO(1,YYLINHA,YYPOS);
                  return k;
                 }
                 else k = ts[k].col;
         }
         if (l == MAX + 1)
         {
                 ERRO(2,YYLINHA,YYPOS);          /* estouro da TS */
                 return l;
         }
         strcpy(ts[l].nome,simb);
         ts[l].nivel = nivel;
         ts[l].classe = NAO_DECL;
         ts[l].col = thash[n];
         thash[n] = l;
         l++;
         return l-1;
        } /* INSTALA*/
```

```
int PROCURAID(alfa simb)
{
  int n,k;

  n = HASH(simb);
  k = thash[n];
  while (k != 0)
  {
        if (strcmp(simb,ts[k].nome) == 0) return k;
        k = ts[k].col;
  }
  return INSTALA(simb);
} /* PROCURAID */

void ABLOCO(void)
{
  nivel++;
  if (nivel > NMAX)
        ERRO(3,YYLINHA,YYPOS);  /* estouro limite de niveis */
  else escopo[nivel] = l;
} /*ABLOCO*/

void FBLOCO(void)
{
  int s,b,k;

  s = l;
  b = escopo[nivel];
  while (s > b)                 /* desfaz hash */
  {
        s--;
        k = HASH(ts[s].nome);
        thash[k] = ts[s].col;
  }
  nivel--;
} /*FBLOCO*/

void TAMANHO(int proced,int addr)
{
  ts[proced].endoff = addr;
} /* TAMANHO */

void DECLARA(int k,int class,int tipot,int offset)
{
  ts[k].classe = class;
  ts[k].tipotam = tipot;
  ts[k].endoff = offset;
} /* DECLARA */
```

```
        int FOIDECLARADO(int k)
        {
         if (ts[k].classe != NAO_DECL)
                return 1;               /* FOIDECLARADO = true  */
         else return 0;                 /* FOIDECLARADO = false */
        } /*FOIDECLARADO*/


/*=================================================================*/
/*                          MODULO   OFFSET                        */
/*=================================================================*/


%%VARIABLES

        int poff[8];
        int toff;
        int offset;

%%PROCEDURES

        void POPOFF(int offst)
        {
                if (toff < 1)
                  ERRO(4,YYLINHA,YYPOS);
                else {
                        offst = poff[toff];
                        toff = toff-1;
                       }
        } /* POPOFF */

        void PUSHOFF(int offst)
        {
         if (toff > 7)
                 ERRO(5,YYLINHA,YYPOS);
         else {
                        toff = toff +1;
                        poff[toff] = offst;
                        offst = 1;
                      }
        } /* PUSHOFF */

        int TEMP(void)
        {
         temps = temps - 1;
         return temps + 1;
        } /* TEMP */
```

```
/*=================================================================*/
/*                    MODULO   DE ANALISE LEXICA                   */
/*=================================================================*/


%%CONSTANTS

        /**** GETCHAR ****/

        #define LINHAMAX 66
        #define SPACELINE 1

        /***** YYSCAN ******/

        #define ENDFILE EOF
        #define NEWLINE 13
        #define HTAB 9
        #define LINEFEED 10
        #define CTLZ 26
        #define BACKSPACE 8
        #define VT 11
        #define FF 12
        #define TLE 72

%%VARIABLES

        int errofatal;  /* Variavel para indicar um erro fatal */
        FILE *FONTE;

        /*YYSCAN*/

        int ii;
        int ctx;
        alfa key[12];
        int tipo_pal_res[12];

        /***** GETCHAR ******/

        char ch;
        int ll,cc;
        char linha[TLE + 1];
        int pagina;
        int linenum;
        int pagcomp;
        int marginf;
        int margem;
        int margsup1;
```

```
        int margsup2;
        int margsup;

%%PROCEDURES

        void MONTATABELAS(void)
        {
          /* tabela de palavras reservadas */
          strcpy(key[1]," begin         ");
          strcpy(key[2]," boolean       ");
          strcpy(key[3]," do            ");
          strcpy(key[4]," else          ");
          strcpy(key[5]," end           ");
          strcpy(key[6]," if            ");
          strcpy(key[7]," integer       ");
          strcpy(key[8]," procedure     ");
          strcpy(key[9]," program       ");
          strcpy(key[10]," then         ");
          strcpy(key[11]," while        ");

          tipo_pal_res[1] = xbegin;
          tipo_pal_res[2] = xboolean;
          tipo_pal_res[3] = xdo;
          tipo_pal_res[4] = xelse;
          tipo_pal_res[5] = xend;
          tipo_pal_res[6] = xif;
          tipo_pal_res[7] = xinteger;
          tipo_pal_res[8] = xprocedure;
          tipo_pal_res[9] = xprogram;
          tipo_pal_res[10] = xthen;
          tipo_pal_res[11] = xwhile;
        } /* MONTATABELAS */

        void INICIA(void)
        {
          ch = ' ';
          ll = 0;  cc = 0;
          pagina  = 0;
          linenum = 0;
          pagcomp = LINHAMAX;
          margem  = 4;
          margsup1 = 2;
          margsup2 = 2;
          margsup = margsup1 + margsup2 + 3;
          marginf = pagcomp - margem;
          ctx     = 1;
          l       = 2;
```

```
  strcpy(ts[1].nome,"               ");
  ts[1].nivel   = 1;
  ts[1].classe  = NAO_DECL;
  ts[1].tipotam = INTEIRO;
  ts[1].endoff  = 0;
  for (ii = 1; ii <= 1000; ii++)
          ts[ii].col = 0;
  for (ii = 0; ii <= 90;ii++)
          thash[ii] = 0;
  MONTATABELAS();
  temps = -1;
  nivel = 0;
  ABLOCO();
} /* INICIA */

void WRITEFONTE(void)
{
  int i;

  if (linenum > marginf)
  {
        fprintf(YYSAIDA,"%4s"," ");
        for (i = 1; i <= 15; i++)
                fprintf(YYSAIDA,"----+");
        for (i = 1; i <= margem-1; i++)
                fprintf(YYSAIDA,"\n");
        linenum = 0;
  }
  if (linenum == 0)
  {             /* comeca nova pagina */
        pagina++;
        for (i = 1; i <= margsup1; i++)
                fprintf(YYSAIDA,"\n");
        fprintf(YYSAIDA,"%4s"," ");
        for (i = 1; i <= 15; i++)
                fprintf(YYSAIDA,"----+");
        fprintf(YYSAIDA,"\n");
        fprintf(YYSAIDA,"%s%8s%s%40s%s%3d\n","LINHA"," ","TEXTO", " ",
                "PAGINA ",pagina);
        for (i = 1; i <= margsup2; i++)
                fprintf(YYSAIDA,"\n");
        linenum = margsup + 3;
        fprintf(YYSAIDA,"%1d%3s",YYLINHA," ");
        for (i = 1; i <= ll-1; i ++)
                fprintf(YYSAIDA,"%c",linha[i]);
        linenum = linenum + SPACELINE;
        for (i = 1; i<= SPACELINE; i++)
```

```
                fprintf(YYSAIDA,"\n");
  }
  else {
                fprintf(YYSAIDA,"%1d%3s",YYLINHA," ");
                for (i = 1; i <= ll-1; i ++)
                        fprintf(YYSAIDA,"%c",linha[i]);
                linenum = linenum + SPACELINE;
                for (i = 1; i<= SPACELINE; i++)
                        fprintf(YYSAIDA,"\n");
               }
}  /* WRITEFONTE */

void GETCHAR(char *c)
{
  if (cc == ll)
  {
        if (feof(FONTE))
        {
          *c = ENDFILE;
          return;
        }
        ll = 1;
        cc = 0;
        fscanf(FONTE,"%c",&linha[ll]);
        while (linha[ll] != '\n')
        {
          ll++;
          fscanf(FONTE,"%c",&linha[ll]);
        }
        ll++;
        linha[ll] = NEWLINE;
        fscanf(FONTE,"\n");
        YYLINHA++;

        /* impressao do texto de entrada */
        WRITEFONTE();
  }
  cc++;
  *c = linha[cc];
} /*GETCHAR*/

void YYSCAN(void)
{
  int inum,i,j,k,kk;
  int tam;
  alfa ident;                   /* a posicao 0 de ident contem branco. */
  int PermiteLoop;              /* permite primeira entrada no loop while*/
```

```
  for (k = 0; k <= IDTAM; k++)
          ident[k] = ' ';
  ident[IDTAM] = '\0';
UM:while ((ch == ' ')  || (ch == HTAB) || (ch == NEWLINE) ||
                  (ch == CTLZ) || (ch == BACKSPACE) || (ch == VT) ||
                  (ch == FF) || (ch == LINEFEED))
                 GETCHAR(&ch);
  YYPOS = cc;
  if (ch == ENDFILE)
  {
        YYSIMB = YYEOF;
        goto DOIS;
  }
  else if (ch == '{')
                {
                  GETCHAR(&ch);
                  while (ch != '}')
                                GETCHAR(&ch);
                  GETCHAR(&ch);
                  goto UM;
                }
                else if (isalpha(ch))       /* palavra */
                        {
                                tam = 0;
                                PermiteLoop = 1;
                                while ((PermiteLoop) || (isalnum(ch)))
                                {
                                  PermiteLoop = 0;
                                  if (tam < IDTAM)
                                  {
                                  /* converte carater lido para minuscula */
                                        if (isupper(ch))
                                                ch = ch + 32;
                                        tam++;
                                        ident[tam] = ch;
                                  }
                                  GETCHAR(&ch);
                                }
                                /* procura por palavra chave */
                                i = 1;
                                j = NPC+1;
                                PermiteLoop = 1;
                                while ((PermiteLoop) || (i<=j))
                                {
                                  PermiteLoop = 0;
                                  k = (i+j) / 2;
                                  if (strcmp(ident,key[k]) <= 0)
```

```
                    j = k-1;
                if (strcmp(ident,key[k]) >= 0)
                    i = k+1;
            }
            if (i-1>j)
              YYSIMB = tipo_pal_res[k];
            else {
                        YYSIMB = xid;
                        if (ctx == 1)
                          "id".valor = INSTALA(ident);
                        else "id".valor = PROCURAID(ident);
                    }
        }
        else if (isdigit(ch))    /* numero */
             {
                inum = 0;
                YYSIMB = xcte;
                PermiteLoop = 1;
                while ((PermiteLoop) || (isdigit(ch)))
                {
                 PermiteLoop = 0;
                 kk = ch - '0';
                 if ((inum > 3276) || ((inum == 3276) && (kk >
                        7)))
                         ERRO(6,YYLINHA,YYPOS);
                 else {
                            inum = inum * 10 + kk;
                            GETCHAR(&ch);
                       }
                }
                "cte".valor = inum;
             }
             else switch (ch)
                  {
                        case ':':{      GETCHAR(&ch);
                                        if (ch == '=')
                                        {
                                        YYSIMB = xatr;
                                        GETCHAR(&ch);
                                        }
                                        else YYSIMB = xdp;
                                        break;
                                 }
                        case ';': {     GETCHAR(&ch);
                                        YYSIMB = xpv;
                                        break;
                                  }
```

```
                                        case '(': {     GETCHAR(&ch);
                                                        YYSIMB = xap;
                                                        break;
                                                  }
                                        case ')': {     GETCHAR(&ch);
                                                        YYSIMB = xfp;
                                                        break;
                                                  }
                                        case '+': {     GETCHAR(&ch);
                                                        YYSIMB = xmais;
                                                        break;
                                                   }
                                        case '-': {     GETCHAR(&ch);
                                                        YYSIMB = xmenos;
                                                        break;
                                                  }
                                        case '*': {     GETCHAR(&ch);
                                                        if (ch == '*')
                                                        {
                                                        YYSIMB = xpot;
                                                        GETCHAR(&ch);
                                                        }
                                                        else YYSIMB = xvezes;
                                                        break;
                                                  }
                                        case '/': {     GETCHAR(&ch);
                                                        YYSIMB = xdiv;
                                                        break;
                                                  }
                                        case '=': {     GETCHAR(&ch);
                                                        YYSIMB = xigual;
                                                        break;
                                                   }
                                        default : {     fprintf(YYSAIDA,
                                                        "CHAR=%d%c", ch,ch);
                                                        YYSIMB = xerror;  /* 40 */
                                                        GETCHAR(&ch);
                                                        fprintf(YYSAIDA,
                                                        "char=%1d%1c",ch,ch);
                                                        break;
                                                  }
                                    } /* switch */
DOIS:;} /*YYSCAN*/
```

```
/*=================================================================*/
/*                        MODULO CODIGO INTERMEDIARIO
        */
/*=================================================================*/


%%CONSTANTS

        /* operadores para opquad */

        #define CPROCEND   1
        #define CPROCBEGIN 2
        #define CATRIB     3
        #define CPARAM     4
        #define CCALL      5
        #define CIGUAL     6
        #define CGOTO      7
        #define CMAIS      8
        #define COU       13
        #define CVEZES     9
        #define CE        14
        #define CMENOS     10
        #define CINVERTE   15
        #define CNEGA      16
        #define CDIV       11
        #define CPOT       12
        #define CDSVF      17
        #define CPROGEND   18
        #define CPROGBEGIN 19

%%TYPES

        /* gen */

        typedef struct {
                          int opquad,opn1quad,opn2quad,opn3quad;
                        } tquad;

%%VARIABLES

        /* gen */
        int op;
        tquad quad[501];
        FILE *CODE;
        int proxq;
        alfa opnome[20];
```

```
        int temps;
%%PROCEDURES

        void GEN(int operador,int opn1,int opn2,int opn3)
        {
         quad[proxq].opquad   = operador;
         quad[proxq].opn1quad = opn1;
         quad[proxq].opn2quad = opn2;
         quad[proxq].opn3quad = opn3;
         proxq++;
        } /* GEN */

        void CONSERTA(int cond,int addr)
        {
         quad[cond].opn2quad = addr;
        } /* CONSERTA */

        void SALVA(int i,int k)
        {
         int j;
         for (j = i; j <= k; j++)
                 fwrite(&quad[j],sizeof(tquad),1,CODE);
        } /* SALVA */

        void MONTAOPNOME(void)
        {
         strcpy(opnome[1 ]," PROCEND      ");
         strcpy(opnome[2 ]," PRCBEGIN     ");
         strcpy(opnome[3 ]," :=           ");
         strcpy(opnome[4 ]," PARAM        ");
         strcpy(opnome[5 ]," CALL         ");
         strcpy(opnome[6 ]," =            ");
         strcpy(opnome[7 ]," GOTO         ");
         strcpy(opnome[8 ]," +            ");
         strcpy(opnome[9 ]," *            ");
         strcpy(opnome[10]," -            ");
         strcpy(opnome[11]," /            ");
         strcpy(opnome[12]," **           ");
         strcpy(opnome[13]," OU           ");
         strcpy(opnome[14]," E            ");
         strcpy(opnome[15]," -            ");
         strcpy(opnome[16]," NEGA         ");
         strcpy(opnome[17]," DSVF         ");
         strcpy(opnome[18]," PROGEND      ");
         strcpy(opnome[19]," PRGBEGIN     ");
        } /* MONTAOPNOME */
```

```
void IMPRIMECODIGO(void)
{
 int i,j;
 tquad quad;

 MONTAOPNOME();
 fprintf(YYSAIDA,"%4s%s\n"," ",
              "==========CODIGO GERADO==========");
 fprintf(YYSAIDA,"\n");
 fprintf(YYSAIDA,"%6s%s%2s%s%s%s"," ","opquad"," ","opn1quad ",
              "opn2quad ","opn3quad","\n");
 j = 1;
 while (!feof(CODE))
 {
        fread(&quad,sizeof(tquad),1,CODE);
        fprintf(YYSAIDA,"\n");
        if (!feof(CODE))
        {
        fprintf(YYSAIDA,"%4d%2s%s",j," ",opnome[quad.opquad]);
        switch(quad.opquad)
        {
              case CPROCEND    : { break; }

              case CPROCBEGIN  : {    fprintf(YYSAIDA,"%4s%d"," ",
                                      quad.opn1quad);
                                       break;
                                 }

              case CATRIB      : { fprintf(YYSAIDA,"%4s%d%6s%d"," ",
                                   quad.opn1quad," ",quad.opn2quad);
                                   break;
                                 }

              case CPARAM      : {fprintf(YYSAIDA,"%4s%d"," ",quad.opn1quad);
                                   break;
                                   }

              case CCALL       : { fprintf(YYSAIDA,"%4s%d"," ",quad.opn1quad);
                                   break;
                                 }

              case CGOTO       : { fprintf(YYSAIDA,"%12s%d"," ",quad.opn2quad);
                                    break;
                                 }
```

```
case CMAIS      : { fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                        quad.opn1quad," ",quad.opn2quad," ",
                        quad.opn3quad);
                    break;
                  }

case COU        : {   fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                        quad.opn1quad," ", quad.opn2quad," ",
                        quad.opn3quad);
                    break;
                  }

case CIGUAL     : {fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                        quad.opn1quad," ",quad.opn2quad," ",
                        quad.opn3quad);
                    break;
                  }

case CDIV       : {   fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                        quad.opn1quad," ",quad.opn2quad," ",
                        quad.opn3quad);
                    break;
                  }

case CVEZES     : {fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                    quad.opn1quad," ",quad.opn2quad," ",
                    quad.opn3quad);
                    break;
                  }

case CE         : {     fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                        quad.opn1quad," ",
                    quad.opn2quad," ",quad.opn3quad);
                    break;
                  }

case CPOT       : {   fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                    quad.opn1quad," ",
                    quad.opn2quad," ",quad.opn3quad);
                    break;
                  }

case CMENOS     : {fprintf(YYSAIDA,"%4s%d%6s%d%8s%d"," ",
                        quad.opn1quad," ",
                    quad.opn2quad," ",quad.opn3quad);
                    break;
                   }
```

```
                    case CINVERTE   : {      fprintf(YYSAIDA,"%4s%d%6s%d"," ",
                                             quad.opn1quad," ",quad.opn2quad);
                                             break;
                                      }

                    case CNEGA      : { fprintf(YYSAIDA,"%4s%d%6s%d"," ",
                                        quad.opn1quad," ",quad.opn2quad);
                                        break;
                                      }

                    case CDSVF      : { fprintf(YYSAIDA,"%4s%d%6s%d"," ",
                                        quad.opn1quad," ",quad.opn2quad);
                                        break;
                                                }
                    case CPROGEND   : { break; }

                    case CPROGBEGIN  : { break; }

            } /* switch */
            j++;
            } /* if */
        } /* while */

        /* imprime tabela de simbolos */
        fprintf(YYSAIDA,"\n");
        fprintf(YYSAIDA,"%4s%s\n"," ",
                    "==========TABELA DE SIMBOLOS==========");
        fprintf(YYSAIDA,"%s%4s%s%4s%s%4s%s%4s%s\n","nome"," ","classe"," ",
                     "tipotam"," ","endoff"," ","nivel");
        for (i = 1; i <= l-1; i++)
              fprintf(YYSAIDA,"%6s%4d%8d%12d%12d\n",ts[i].nome,ts[i].classe,
                       ts[i].tipotam,ts[i].endoff,ts[i].nivel);
    } /* IMPRIMECODIGO */

/*=================================================================*/
/*                    MODULO ROTINAS SEMANTICAS                    */
/*=================================================================*/


%%VARIABLES

    int temporario;
    int classe;
```

```
%%GRAMMAR  program  AND  SEMANTICS


program  = proghead dcls cmdc
                {
                        SALVA(1,proxq-1);
                        GEN(CPROGEND,0,0,0);
                };

proghead = "program"
                {
                        toff    = 0;
                        offset  = 0;
                        proxq   = 1;
                        GEN(CPROGBEGIN,0,0,0);
                };

dcls  = dcl
                {    }
          |  dcls ";" dcl
                {    } ;

dcl  = "id" ":" "integer"
           {
                DECLARA("id".valor,VARIAVEL,INTEIRO,offset);
                offset++;
           };

dcl  = "id" ":" "boolean"
           {
                DECLARA("id".valor,VARIAVEL,LOGICO,offset);
                offset++;
           };

dcl  = prochead "(" par ")" dcls cmdc
           {
                TAMANHO(prochead.valor,offset);
                POPOFF(offset);
                ctx = 1;
                GEN(CPROCEND,0,0,0);
                FBLOCO();
                SALVA(prochead.inicio,proxq-1);
                proxq = prochead.inicio;
           };
```

```
par  = "id" ":" "integer"
          {
               DECLARA("id".valor,PAR,INTEIRO,offset);
               offset++;
          };

par  = "id" ":" "boolean"
          {
               DECLARA("id".valor,PAR,LOGICO,offset);
               offset++;
          };

prochead  = "id" ":" "procedure"
                    {
                     DECLARA("id".valor,PROC,0,proxq);
                     prochead.inicio = proxq;
                     GEN(CPROCBEGIN,"id".valor,0,0);
                     prochead.valor = "id".valor;
                     ABLOCO();
                     PUSHOFF(offset);
                    };

cmdc  = c "begin" cmds "end"
               {    };

c  =
               { ctx = 2; /* CONTEXTO DE COMANDOS  */ };

cmds  = cmd
               {    }
         | cmds ";" cmd
               {    };

cmd  = "id" ":=" exp
          {
               classe = ts["id".valor].classe;
               if ((FOIDECLARADO("id".valor)) && ((classe==VARIAVEL) ||
                    ( classe == PAR)))
                    GEN(CATRIB,"id".valor,exp.r,0);
               else ERRO(7,"id".YYLINHA,"id".YYPOS);
          };

cmd  = "id" "(" exp ")"
          {
               if ((FOIDECLARADO("id".valor)) && (ts["id".valor].classe == PROC))
               {
                    GEN(CCALL,"id".valor,0,0);
```

```
                    GEN(CPARAM,exp.r,0,0);
                }
                else ERRO(8,"id".YYLINHA,"id".YYPOS);
        };

cmd  = "if" cond "then" cmd  "else" n cmd
        {
                CONSERTA(cond.quad,n.quad+1);
                CONSERTA(n.quad,proxq);
        };

cmd  = "while" marca cond "do"  cmd
        {
                GEN(CGOTO,marca.quad,0,0);
                CONSERTA(cond.quad,proxq);
        };

cmd  = cmdc
        {    };

cond  = exp
                {
                 if (exp.tipo == LOGICO)
                        GEN(CDSVF,exp.r,0,0);
                 else ERRO(10,exp.YYLINHA,exp.YYPOS);
                 cond.quad = proxq;
                };

n  =
                {
                 n.quad = proxq;
                 GEN(CGOTO,0,0,0);
                };

marca  =
                { marca.quad = proxq; };

exp  = exp "+" exp
        {
                if (exp[2].tipo != exp[3].tipo)
                        ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
                if (exp[2].tipo == INTEIRO)
                        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) || (exp[2].tipo != INTEIRO))
                        ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
                temporario = TEMP();
                GEN(CIGUAL,temporario,exp[2].r,exp[3].r);
                exp.r = temporario;
                exp.tipo = LOGICO;
        };

exp  = exp "/" exp
        {
                if ((exp[2].tipo != exp[3].tipo) || (exp[2].tipo != INTEIRO))
                        ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
                temporario = TEMP();
                GEN(CDIV,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)
                        ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
                if (exp[2].tipo == INTEIRO)
                        op = CVEZES;
                else op = CE;
                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) || (exp[2].tipo != INTEIRO))
                         ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
                temporario = TEMP();
                GEN(CPOT,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) || (exp[2].tipo != INTEIRO))
                      ERRO(9,exp[2].YYLINHA,exp[2].YYPOS);
                temporario = TEMP();
                GEN(CMENOS,temporario,exp[2].r,exp[3].r);
                exp.r = temporario;
                exp.tipo = exp[2].tipo;
          };

exp  = "-" exp   %%PREC "*"
          {
                temporario = TEMP();
                if (exp[2].tipo == INTEIRO)
                      op = CINVERTE;
                else op = CNEGA;
                GEN(op,temporario,exp[2].r,0);
                exp.r = temporario;
                exp.tipo = exp[2].tipo;
          }

         | "(" exp ")"
          {
                exp.r = exp[2].r;
                exp.tipo = exp[2].tipo;
          };

exp  = "id"
          {
                classe = ts["id".valor].classe ;
                exp.r  = "id".valor;
                if (FOIDECLARADO("id".valor) && ((classe == VARIAVEL) ||
                      (classe == PAR)))
                      exp.tipo = ts["id".valor].tipotam;
                else {
                            ERRO(7,"id".YYLINHA,"id".YYPOS) ;
                            exp.tipo = INTEIRO;
                       }
          };

exp  = "cte"
          {
                exp.r ="cte".valor;
                exp.tipo = INTEIRO;
          };
```

```
%%CONFLICTS

        %%RIGHT   "=" ;
        %%LEFT    "+","-" ;
        %%LEFT    "*","/"
        %%RIGHT   "**"

(*=================================================================*)
(*                          MODULO PRINCIPAL                       *)
(*=================================================================*)


%%PROGRAM

        if ((FONTE=fopen("entrada.008","r"))==NULL)
        {
                printf("ARQUIVO FONTE NAO PODE SER ABERTO\n");
                exit(1);
        }
        else if ((YYSAIDA = fopen("saida.008","w"))==NULL)
                 {
                        printf("ARQUIVO DE SAIDA NAO PODE SER ABERTO\n");
                        exit(1);
                 }
                 else if ((CODE =fopen("EX.COD","wb"))==NULL)
                          {
                                printf("ARQUIVO PARA ESCRITA DAS QUADRUPLAS
                                        NAO PODE SER ABERTO","\n");
                                exit(1);
                          }
                          else { INICIA();
                                        YYPARSER();
                                        fclose(CODE);
                                        if ((CODE=fopen("EX.COD","rb"))==NULL)
                                        {
                                                printf("ARQUIVO PARA LEITURA DAS
                                                QUADRUPLAS NAO PODE SER
                                                ABERTO","\n");
                                                exit(1);
                                        }
                                        else {IMPRIMECODIGO();
                                                switch(YYRESULTADO)
                                                {
                                                case YYSINTAXEOK   : { fprintf(YYSAIDA,
                                                "PARSE COMPLETED -- SINTAXE OK!\n");
                                                break;
                                                }
```

```
                                        case YYERROFATAL   : { fprintf(YYSAIDA,
                                        "ERRO FATAL: PROGRAMA
                                        CANCELADO.\n");
                                        break;
                                    }
                                        case YYERROSINTAXE : { fprintf(YYSAIDA,
                                        "ERRO DE SINTAXE.\n");
                                        break;
                                    }
                            }
                            fclose(YYSAIDA);
                            fclose(CODE);
                        }
                }
%%END
```