Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Relatório Final

Transporte do Sistema SIC de MS-DOS para ambiente WINDOWS

por

Marco Rodrigo Costa
Valeska Gonçalves Russo

Projeto Orientado II
Área: Compiladores
Orientadora: Mariza Andrade S. Bigonha

Belo Horizonte
junho, 1996

Sinopse

Os principais objetivos de nosso trabalho são: (1) permitir a utilização do SIC em um novo ambiente, WINDOWS, e, (2) gerar o compilador na linguagem C.

O Sistema de Implementação de Compiladores (SIC) é uma ferramenta, implementada em ambiente MS-DOS, destinada a assistir a implementação de compiladores através de uma linguagem especial, SIC, baseada em PASCAL. O sistema recebe como entrada programas em SIC e gera compiladores na linguagem TURBO PASCAL.

Este relatório descreve como foi realizado o transporte do Sistema de Implementação de Compiladores, SIC, de MS-DOS para o ambiente WINDOWS, a geração do compilador na linguagem C e a sua nova interface. Além disto, relata a metodologia empregada no desenvolvimento de nosso trabalho.

Abstract

The two most important points of this project are: (1) to allow the system, SIC, to be used in a new environment, WINDOWS, and, (2) to generate the compiler in the C language.

The Compiler Implementation System (SIC) is a tool, implemented in MS-DOS, whose purpose is to assist the construction of compilers by means of a special language, SIC, based on PASCAL. The system receives SIC programs and generates compilers in the TURBO PASCAL language.

This report shows how the Compiler Implementation System, SIC, has been converted from MS-DOS to WINDOWS, how the generation of the compiler in the C language was made and its new interface. In addition, it describes the methodology used in the development of this project.

Sumário

1- Introdução

Os processos de compilação já estão bastante formalizados, e nem todos os problemas envolvidos no desenvolvimento de compiladores requerem, necessariamente, esforço humano para serem solucionados. Muitas pesquisas já foram realizadas no sentido de automatizar vários destes processos. Como resultado podem ser encontradas atualmente, na literatura, descrições de geradores automáticos de analisadores sintáticos: YACC [Johnson, 1975], SIC [Bigonha & Bigonha, 1988], LLama e Occs [Allen, 1990], LALRGen [Fisher & LeBlanc, 1991], entre outros; geradores de analisadores léxicos, como LEX [Lesk, 1975], e, em menor escala, geradores automáticos de geradores e otimizadores de código [Costa, 1990, Bigonha, 1995].

Do ponto de vista estrutural, todos os geradores de compiladores são similares. Eles consistem basicamente em dois grandes componentes, o gerador de analisador sintático e o gerador de gerador de código. O gerador de analisador sintático tem por objetivo produzir o algoritmo de análise sintática do compilador a partir da definição da sintaxe da linguagem. Ao gerador de gerador de código compete produzir o módulo de geração de código do compilador.

Nosso trabalho está relacionado com os geradores de analisadores sintáticos. Em particular estudamos o sistema SIC.

Programas escritos em SIC são traduzidos para TURBO PASCAL. Contudo seria interessante que ele pudesse gerar o compilador em outra linguagem, por exemplo, na linguagem C [Kernighan & Ritchie, 1988]. Isto proporcionaria uma maior flexibilidade ao sistema. Além da possibilidade de obter compiladores em diferentes linguagens, o fato de prover a ferramenta com uma interface distinta do MS-DOS, também contribuiria para uma maior utilização do sistema. Baseados nestes fatos nos propusemos a transportá-lo para o ambiente operacional WINDOWS e gerar o compilador na linguagem C.

A escolha de C padrão [Kernighan & Ritchie, 1988] como linguagem de trabalho foi devido a esta ser uma linguagem bastante flexível, expressiva, eficiente e amplamente utilizada, podendo o sistema ser usado com diferentes compiladores C.

A escolha de WINDOWS como novo ambiente se deve ao fato de que este está sendo cada vez mais usado por um grande número de pessoas e a interface neste ambiente é bastante amigável.

2- Sistemas Geradores de Compiladores

Um grande número de ferramentas foi projetado especificamente para auxiliar no processo de desenvolvimento de compiladores.

Nesta seção são descritos sucintamente alguns geradores automáticos de analisadores sintáticos, com o objetivo de ilustrar as diferenças e semelhanças entre eles e o SIC.

2.1- Yacc

Yacc [Johnson, 1975], *Yet Another Compiler-Compiler,* é uma ferramenta usada para produzir automaticamente analisadores sintáticos LR a partir da descrição gramatical da sintaxe de linguagens e está disponível tanto para plataformas UNIX quanto para PC-DOS.

Um programa fonte Yacc possui três partes:

```
declarações
%%
regras de tradução
%%
rotinas de suporte
```

O usuário do Yacc especifica a estrutura de sua entrada junto com o código a ser ativado à medida que cada estrutura é reconhecida. A partir desta entrada, o Yacc gera uma função, o analisador sintático, para controlar o processo de entrada.

Uma facilidade existente no Yacc é a presença de um mecanismo de recuperação de erros, por meio de incorporação de produções de erro. Ele faz uso de *tokens error*, onde *error* é uma palavra reservada Yacc. Tais *tokens* são inseridos na gramática, possivelmente nos pontos mais propícios a erros. O algoritmo para recuperação de erro do Yacc baseia-se na técnica de re-sincronização da pilha sintática. A inserção de *tokens* de erro tem a desvantagem de complicar a gramática e na maioria das vezes provê uma análise que poderia ser obtida automaticamente da pilha sintática ou semântica.

Além do mecanismo de recuperação de erros, o Yacc possui artifícios para tratar gramáticas ambíguas e pode, até mesmo, ser utilizado conjuntamente com geradores de analisadores léxicos, como o Lex [Lesk, 1975] descrito anteriormente, projetados para produzir analisadores léxicos

2.2- SIC

O Sistema de Implementação de Compiladores (SIC), cujo projeto e implementação são apresentados em detalhes em [Bigonha, 1985 e Bigonha e Bigonha, 1988], foi desenvolvido pelo Grupo de Linguagens de Programação do Departamento de Ciência da Computação (DCC) da Universidade Federal de Minas Gerais (UFMG) e implementado em TURBO PASCAL para ambiente MS-DOS [Bigonha & Bigonha, 1989]. Seu principal objetivo é contribuir para a redução do esforço na implementação de linguagens.

O SIC possibilita a automação de fases utilizadas na construção de *front-ends* de compiladores, dando-se ênfase à geração de reconhecedores sintáticos eficientes e gerais. Este sistema oferece ao usuário uma linguagem especial, denominada SIC. Esta linguagem pode ser vista como uma linguagem PASCAL estendida, ou seja, ela contém todos os recursos da linguagem PASCAL além dos mecanismos considerados importantes na escrita de um compilador, como, por exemplo, meios de associar rotinas semânticas às produções de gramáticas, etc.

O SIC possui mecanismos para a especificação do tipo de processamento a ser usado no compilador do usuário. O compilador pode ser executado no modo *batch* ou interativo, pode ser de um único ou de vários passos e pode ter ou não recuperação de erros [Bigonha & Bigonha, 1989]. Dentro deste espírito, o SIC possui vários analisadores sintáticos. Então, dependendo das opções utilizadas, um dos analisadores sintáticos existentes será incluído no compilador do usuário. Cada analisador sintático é descrito em um arquivo separado.

O SIC possui um mecanismo de recuperação de erros mais elaborado que aquele descrito no Yacc. No SIC, a recuperação de erros é completamente transparente ao usuário. O método implementado não faz nenhuma restrição quanto ao uso de reduções *default* e o uso das mesmas provou não prejudicar a recuperação de erros.

2.3- LALRGen

LALRGen [Fisher & LeBlanc, 1991] é uma ferramenta utilizada para a tradução de linguagens, especificamente linguagens livre do contexto. LALRGen é

um gerador de tabelas. Ele aceita como entrada especificações de linguagens usando uma gramática LALR(1) e produz como resultado uma série de tabelas para serem usadas com o algoritmo do reconhecedor sintático. Ele provê também um mecanismo simples para resolução de conflitos para permitir o uso de gramáticas que não são LALR(1). LALR(1) é um subconjunto de ECP, um gerador de analisadores sintáticos LALR(1) escrito por Jon Mauney.

A entrada para LALRGen possui três seções principais: a) opções desejadas para a execução; b) os símbolos terminais da gramática, e; c) produções da gramática. A forma geral da entrada é:

<components>
* ecp
    <options>
* define
    <constant definitions>
* terminals
    <terminal especification>
* productions
    <productions especifications>
* end
    <comments>

Erros de sintaxe no arquivo de entrada para o LALRGen são tratados por uma rotina de recuperação de erros embutida no sistema. Caso haja erros na entrada, não haverá a geração da tabela.

2.4- Llama e Occs

LLama e Occs [Allen, 1990] são geradores de compiladores que traduzem gramáticas de atributos em um analisador sintático. Eles são essencialmente pré-processadores C que recebem como entrada um arquivo contendo código fonte em C, produções de gramática, diretivas de compilação e produzem como saída um programa em C para o analisador sintático.

Occs significa *the Other Compiler Compiler System*. Ambos LLama e Occs foram inspirados no Yacc. Occs constrói um reconhecedor sintático *bottom-up* LALR(1). Enquanto LLama constrói um reconhecedor sintático *top-down* LL(1).

Occs, embora seja similar ao Yacc, não é um Yacc. Occs possui um mecanismo de recuperação de erros mais eficiente que Yacc, apesar de ainda deixar muito a desejar. A saída produzida é mais fácil de ser lida. Ele provê um ambiente de depuração embutido no sistema.

O arquivo de entrada para Llama e Occs é composto de três seções separadas por diretivas %%:

```
definitions
%%
rules
%%
code
```

A seção de definições é a primeira parte do arquivo de entrada e contém as diretivas para o compilador e código C para ser usado na seção de regras.

A seção de regras é delimitada por %% e contém as produções e as rotinas semânticas.

A seção de código compreende todo o texto após a segunda diretiva "%%". Esta seção é totalmente transcrita para a saída sem nenhuma modificação. Ela contém, dentre outras facilidades, procedimentos usados pelas rotinas semânticas.

3- Caracterização do Problema

3.1- Trabalho Proposto

Propomos transportar o Sistema de Implementação de Compiladores (SIC) para o ambiente operacional WINDOWS e gerar o compilador na linguagem C.

3.2- Plano de Trabalho

Para a realização do projeto proposto, o trabalho foi dividido em várias tarefas. O cronograma abaixo relaciona estas etapas cujo conteúdo e resultados são descritos na Seção 4.1.

Etapa	Descrição	Data
1	Estudo de técnicas de compilação	08/08/95 a 23/09/95
2	Estudo do sistema de interface homem-máquina	24/09/95 a 30/09/95
3	Estudo do SIC: uso do sistema e sua implementação	01/10/95 a 31/10/95
4	Tradução do compilador gerado pelo SIC em PASCAL para C, usando C padrão	03/11/95 a 30/04/96
5	Estudo do ambiente WINDOWS	02/05/96 a 31/05/96
6	Instalação do SIC versão C em ambiente WINDOWS	01/06/96 a 30/06/96
7	Geração de relatórios e manual para o usuário	15/05/96 a 30/06/96

Tabela 1: Descrição das Etapas

4- Desenvolvimento do Trabalho Proposto

4.1- Histórico do Desenvolvimento do Projeto

4.1.1- Etapa 1

A primeira etapa consistiu no estudo de técnicas utilizadas no desenvolvimento de compiladores como: a análise léxica, a análise sintática e a tradução dirigida por sintaxe. Esta etapa foi realizada fazendo, quando possível, o relacionamento das técnicas de compilação existentes com aquelas empregadas no SIC, dando-se ênfase à fase de análise sintática.

A análise sintática é uma fase da compilação em que a estrutura sintática de um programa é analisada [Aho et al, 1986, Kristensen, 1981]. Várias técnicas de análise sintática foram estudadas. Dentre elas foram vistas com mais cuidado: um dos métodos *top-down*, a técnica por descida recursiva, usada para a implementação da linguagem do SIC, e um dos métodos *bottom-up*, a técnica LALR, utilizada no compilador gerado pelo SIC.

Além da análise sintática, uma atenção especial foi dada à compactação de tabelas [Bigonha & Bigonha, 1983] e à recuperação de erros [Bigonha & Bigonha, 1985].

Esta etapa não foi muito complicada, pois a maior parte dos assuntos já tinham sido estudados no curso de Compiladores e portanto o que fizemos foi basicamente uma revisão.

4.1.2- Etapa 2

Entendidas as técnicas de compilação, o próximo passo dentro do cronograma consistiu no estudo de "WINDOW 2.0 ¾ Um Sistema Básico de Gerência de Interface" [Bigonha, 1990]. O pacote WINDOW é um gerenciador de interfaces em modo caractere e implementado em TURBO PASCAL versão 4.0, constituído por um conjunto de funções e procedimentos para construção de janelas e menus.

Nosso estudo, no decorrer desta etapa, se deu em dois níveis: nos preocupamos primeiramente com detalhes a nível de configuração e utilização do pacote, como por exemplo, como é feita a importação de identificadores, cuidados com o monitor de vídeo a ser utilizado, identificadores e tipos pré-definidos, condições de erro e entrada de dados. No segundo nível, nos concentramos nas

janelas; vimos sua construção e ativação, empilhamento e movimentação, dentre outras operações. Em um outro estágio, verificamos como se dá a construção de menus, sua exibição e seleção de itens. Para terminar, verificamos a seqüência de finalização, alguns exemplos de utilização dos procedimentos e executamos um programa DEMO para consolidar o aprendizado do WINDOW.

A importância deste estudo foi evidente, pois mostrou-nos como redefinir a interface do SIC de forma a isolá-la ao máximo do código principal do sistema.

4..1.3- Etapa 3

A terceira etapa, a mais delicada delas, consistiu em um estudo do SIC para o entendimento de seu funcionamento e de sua implementação. Fez parte desta etapa além do estudo do código fonte do SIC, o estudo do manual do usuário e de outros materiais considerados importantes para a compreensão do sistema [Bigonha, 1985, Bigonha & Bigonha, 1989]. Esta fase foi realizada cuidadosamente, pois o código do sistema deveria estar claramente entendido para que pudesse ser posteriormente modificado.

Esta etapa, teve seu início em 07 de outubro. Analisamos o código fonte do SIC [Bigonha, 1985] e pudemos perceber a grande modularização do sistema. Para estabelecer uma ordem de estudo e facilitar a compreensão do sistema, os módulos analisados foram distribuídos em 4 fases. Relacionamos agora estas fases e apresentamos uma breve explicação de cada módulo incluso.

A primeira fase incluiu SICWINDW, que consiste de procedimentos e funções baseados no WINDOW e na biblioteca CRT do TURBO PASCAL. Em seguida, passamos para o SICBASIC e SICAMB. O primeiro contém procedimentos e funções para determinação do dia e hora em que o sistema está sendo executado e outras operações básicas realizadas no SIC. Já o segundo pacote, SICAMB, trata do arquivo de comunicação do SIC e qualquer outra variável que diz respeito ao ambiente no qual está sendo executado o SIC, por exemplo: os *drives* de entrada, de saída e de execução, o nome do arquivo fonte, dentre outros.

A segunda fase está relacionada mais diretamente com a geração do compilador. O principal módulo desta fase, o módulo SICPHSE1 é responsável pela implementação da linguagem SIC. O método de reconhecimento utilizado é o denominado *top-down* [Aho et al, 1986]. Neste módulo, a partir de um programa de entrada, são gerados vários arquivos que são posteriormente unidos para formar o compilador do usuário.

A terceira fase consistiu do módulo SICPHSE2, que provê os algoritmos para geração da tabela LALR [Aho et al, 1986, Bigonha & Bigonha, 1983, Kristensen, 1981] utilizada no compilador do usuário durante a fase da análise sintática.

A quarta fase consistiu do módulo SICPHSE3, responsável pela união dos vários arquivos, gerados nas fases SICPHSE1 e SICPHSE2, que formam o compilador do usuário.

Além da seqüência descrita, estudamos os módulos SICLISTA e SICGLOBL. O módulo SICLISTA é responsável pela impressão de informações úteis na depuração do compilador, como, por exemplo, listagem do conjunto de itens LR(0) gerados pelo SIC, tabelas de referências cruzadas, etc. O módulo SICGLOBL define as estruturas de dados globais utilizadas no sistema. É aqui que se pode conhecer como é a representação da tabela de símbolos, da coleção LR(0), da gramática, etc. [Aho et. al, 1986].

Durante o estudo do código fonte, foi dada uma maior atenção aos procedimentos e funções que deveriam ser modificados, tanto na geração do compilador em C, quanto na interface. Os pontos prováveis de alteração foram anotados para facilitar as etapas seguintes.

Em paralelo ao estudo do código fonte, analisamos também o manual do usuário do SIC [Bigonha & Bigonha, 1989]. Este pôde esclarecer algumas dúvidas que surgiram durante o estudo do sistema. A presença da especificação de um compilador em SIC no manual foi de grande ajuda na compreensão do sistema como um todo.

Para completar o estudo, executamos o SIC e analisamos os resultados gerados e os arquivos criados.

4.1.4- Etapa 4

A quarta etapa consistiu na tradução do compilador gerado pelo SIC em PASCAL [Jensen & Wirth, 1988, Borland, 1987/1988] para C, utilizando C padrão ANSI [Kernighan & Ritchie, 1988].

Nesta etapa, as tarefas foram divididas entre os dois orientandos, e esta divisão pode ser vista na Tabela 2.

Geração das declarações	Marco e Valeska
Geração das variáveis	Marco
Geração dos tipos	Valeska
Geração das constantes	Marco e Valeska
Geração dos procedimentos	Valeska
Geração dos menus	Marco

Algoritmo do analisador sintático	Marco e Valeska

Tabela 2: Itens a serem traduzidos de Pascal para C

O primeiro item traduzido foi o algoritmo do analisador sintático.

Durante a etapa 3 anotamos as partes do código que precisariam ser modificadas, tanto partes de interface quanto partes do compilador gerado, portanto ficou fácil a divisão de tarefas. Separamos vários procedimentos do módulo SICCOMP2.PAS, que contém a principal parte a ser modificada do SIC, e trabalhamos em separado. Após toda essa fase de geração do compilador ser modificada, unimos o fonte, compilamos e executamos o mesmo.

Nesse ponto do projeto, tínhamos tanto o analisador sintático quanto o compilador gerado pelo SIC já em C. Com o objetivo de rodar o programa e fazer alguns testes, traduzimos dois programas de entrada para o SIC de Pascal para C. Com tudo isso já na linguagem C, compilamos os módulos gerados e fizemos testes para algumas entradas. Para verificar se estavam corretos os resultados, executamos os mesmos testes para a versão Pascal, e não houve nenhuma diferença entre os resultados obtidos nos dois compiladores gerados.

4.1.5- Etapas 5, 6 e 7

Para as etapas 5, 6 e 7, apesar de as tarefas já estarem bem definidas para cada aluno, trabalhamos realmente juntos. Isto foi inevitável devido à grande dependência entre cada uma das partes apresentadas na Tabela 3.

Itens a serem tratados na interface	Sentido	Orientando
Nova interface para o SIC	Delphi ® SIC	Valeska
Mudança no código para interface no SIC	SIC ® Delphi	Marco

Tabela 3: Etapas da Interface

Como conseqüência, estas três etapas não seguiram rigidamente o cronograma apresentado na Tabela 1 (veja Seção 3). Para trabalhar nos itens mostrados na Tabela 3, desenvolvemos as etapas finais concomitantemente. Estudamos e elaboramos alguns aplicativos para nos familiarizar com a programação por eventos. A seguir, analisamos cada módulo do SIC identificando o que precisaria ser modificado.

Estudamos os recursos do Delphi à medida que desenvolvíamos a etapa, tanto consultando o *help on line* do Delphi como as notas de aula de Faiçal Farhat de Carvalho, baseadas em Curso de Programação: DELPHI.

4.2- Levantamento dos Problemas e Soluções Adotadas

Com o início da tradução do código SIC que gera o compilador em PASCAL para C, objeto da Etapa 4, os primeiros problemas começaram a aparecer. Estes problemas estão relacionados principalmente com a sintaxe das linguagens envolvidas e são enumerados a seguir:

1) Detectamos o problema da incompatibilidade de tipos existentes entre C e PASCAL. O compilador PASCAL gerado usa registros variantes, enumerações e *subranges*, etc., o que exige alguma alteração para o compilador gerado em C.

Solução: Para solucionar os problemas relacionados às estruturas de dados utilizadas na versão PASCAL que não possuem correspondências diretas em C, adotamos as seguintes convenções:

a) variáveis PASCAL do tipo *boolean* foram feitas da seguinte forma:

- definimos duas constantes, FALSE=0 e TRUE=1;
- uma variável PASCAL do tipo *boolean* foi declarada do tipo inteiro em C e obteve os valores FALSE ou TRUE, exclusivos.

b) A enumeração, não presente como tipo pré-definido em C, foi resolvida da seguinte forma:

- foram definidas constantes, tantas quantas foram necessárias, para se relacioná-las aos valores da enumeração que queríamos representar;
- uma variável do tipo enumeração em PASCAL, foi modificada para assumir o tipo inteiro em C.

O exemplo abaixo ilustra a correspondência adotada.

PASCAL:	C:
X: (a, b, c);	#define a 0 #define b 1 #define c 2 int X;
Utilização: X := a;	Utilização: X = a;

c) Para solucionar o subalcance do PASCAL, declaramos a variável inteira C e garantimos, no programa, que a mesma não assumia valores fora dos limites do *subrange* correspondente PASCAL.

d) Quanto aos arranjos definidos como A: array[*min*..*max*] of *TipoB*, a tradução foi feita para: *TipoB* A[*max*+1], sendo que, quando o valor de *min* era zero, a correspondência era biunívoca. Quando o valor de *min* era maior que zero, as posições de 0 a *min*-1 do arranjo não eram utilizadas, prevalecendo aquelas de *min* a *max*, como desejado, por questão de simplicidade. Desta forma, as referências aos índices do arranjo não precisaram ser alteradas. Optamos por esta convenção, pois a maioria dos arranjos utilizados no SIC ou começam de 0 (zero) ou de 1 (um), e assim demos mais importância à legibilidade do programa, já que o espaço não utilizado será mínimo.

2) Detectamos o problema do aninhamento de procedimentos permitido em PASCAL e não permitido em C [Kernigham & Ritchie, 1988]. O algoritmo de análise sintática incluído no compilador gerado foi traduzido de tal forma que este aninhamento desaparecesse.

Solução: Para resolver o problema do aninhamento de procedimentos não permitido em C, definimos todo procedimento antes de seu uso, obviamente de forma não aninhada.

Um outro tipo de problema que nos deparamos diz respeito à mudança ou não da sintaxe do SIC em detrimento, também, das diferenças entre a sintaxe da linguagem C e a sintaxe da linguagem Pascal. Tais problemas, enumerados abaixo, foram devidamente estudados e tratados sem que houvesse a necessidade de alterar o SIC.

1) Rotinas semânticas, no SIC, são delimitadas por chaves: abre e fecha chaves são, também, delimitadores de bloco em C, portanto ocorre uma ambigüidade durante a análise léxica do SIC.

Solução: utilizamos um contador para guardar o número de abre ({) e fecha (}) chaves dentro do contexto das rotinas semânticas. O contador é inicializado com zero, e incrementado toda vez que encontra um "{" e decrementado quando encontra um "}". Quando tal contador volta a ser zero, assumimos que a rotina semântica em questão acabou.

2) Uso do símbolo de porcentagem (%): como o % não é usado em Pascal, ele é tratado na análise léxica do SIC de forma única. Quando o SIC encontra um %, age de forma como se não pudesse vir tal símbolo em um texto Pascal, mas o símbolo de porcentagem é utilizado em C.

Solução: quando é encontrado um %, o programa SIC interpreta-o como parte de um texto C.

Por questão de simplicidade, não mudamos a sintaxe do SIC em nenhuma destas situações apresentadas. Mas é bom que fique claro que podem ocorrer erros graves se cuidados não forem tomados pelo usuário do sistema na especificação de seu compilador. Os abre e fecha chaves devem ser balanceados ou as rotinas semânticas serão interpretadas de forma equivocada, dificultando encontrar a solução para um erro dessa natureza. Da mesma forma, se o usuário cometer um erro, por exemplo, de preceder a palavra reservada TYPES do SIC apenas por um %, TYPES parecerá para o SIC como um texto C, o que não é verdade. A conseqüência é também um erro grave e de difícil detecção.

Durante a mudança de plataforma de DOS para WINDOWS nos deparamos também com algumas dificuldades as quais enumeramos abaixo:

1) Delphi não é uma linguagem procedural: o SIC foi originalmente implementado em uma linguagem procedural (Pascal), portanto o mesmo deveria ser adaptado para executar em um ambiente orientado a eventos (Delphi).

Solução: não foi adotada nenhuma medida específica. Tivemos de nos acostumar com o "novo" paradigma cercando toda e qualquer ação do usuário.

2) Tamanho da pilha destinada ao SIC: o Delphi 1.0 só reserva 65.520 *bytes* para variáveis globais. À medida que eram inseridos novos formulários e novas variáveis a pilha crescia, gerando erros na compilação do SIC.

Solução: sacrificamos o segmento do Heap em detrimento do segmento de Dados e diminuímos as dimensões de variáveis globais.

Este problema é mais sério, tendo em vista que existem linguagens, como ADA e C++, cujas gramáticas possuem maiores proporções. Portanto, para que o SIC seja capaz de gerar compiladores para estas linguagens sem que haja geração de erros devido a estouros de tabelas, nossa próxima etapa é transferir todo o código para executar na versão 2.0 do Delphi.

4.3- Tradução dos Analisadores Sintáticos

Os primeiros módulos traduzidos para a linguagem C foram os analisadores sintáticos usados no compilador do usuário gerado pelo SIC. Nosso trabalho foi desenvolvido envolvendo as traduções de dois analisadores sintáticos: um sem e o outro com recuperador de erros, ambos em modo *batch* e de um único passo. Em sua implementação, colocamos as estruturas globais, definidas em outros módulos do SIC, como locais ao analisador para que o mesmo compilasse. A tradução destas estruturas foi feita mais tarde, no local correto, à medida que fomos avançando na Etapa 4.

4.4- Geração do Compilador em C

Durante a tradução do compilador gerado houve alguns detalhes simples de se resolver mas que merecem ser documentados. Alguns outros foram omitidos nesta documentação. Por exemplo inserção de novas variáveis globais, mas pelo menos um comentário no código fonte acompanha a modificação e/ou inserção feita.

A primeira atitude tomada foi eliminar a parte de tratamento de *labels* do SIC. *Labels* não são declarados em C, apenas usados, e são encarados de forma semelhante a variáveis.

Uma outra modificação foi alterar o procedimento de geração de comentários em Pascal para gerar comentários em C. Onde vinha {comentário} passou a vir /*comentário */.

Os *tokens* usados no SIC na versão Pascal são delimitados por aspas duplas (“ ”). Como o compilador C interpreta o que vem entre aspas duplas como *string*, ocorria um erro durante a compilação do código gerado. Para solucionar isso, foram feitas alterações nas rotinas QUICKLEX e SEMLEX do SIC.

Em vários momentos optamos por não alterar a sintaxe do SIC, fosse por simplicidade fosse por manter coerência com o projeto original do sistema. Tal postura nossa quanto a isso fica evidente na parte de declaração de atributos. O usuário entrava com o símbolo terminal ou não-terminal seguido pelos seus atributos entre parênteses. Dentro dos parênteses os atributos eram seguidos pelo tipo o qual pertenciam. Se a entrada do usuário for destinada à versão C do SIC, ele deve agir da mesma forma, exceto que o identificador do tipo deve preceder o atributo, e não suceder como na versão Pascal. Neste ponto houve a única mudança quanto à sintaxe do SIC: não existe mais só um identificador para o tipo do atributo, ou seja, a produção da forma **...®...identificador** se torna **...®...identificador**$^{+}$.

Adotamos esse procedimento para que um tipo como *unsigned int*, por exemplo, pudesse ser usado.

A parte que, talvez, mereça maiores esclarecimentos é quanto à estruturação dos módulos finais gerados pelo SIC, especificamente, pelo SICPHSE3. Em Pascal, eram gerados quatro módulos com o mesmo nome do arquivo de entrada para o SIC e com extensões DCL, PRO, SEM e PRS. A partir daí, o usuário editava um arquivo com extensão PAS incluindo os quatro arquivos finais gerados. Para a versão do compilador C, também são criados quatro arquivos finais, três com extensão C, arquivos fonte, e o de declarações com extensão H, arquivo de cabeçalho. Foi criado um mecanismo para montar os nomes dos arquivos finais extraindo-se os três primeiros caracteres do nome do arquivo de entrada, ou menos que três, caso o nome tenha menos que três caracteres. Essa extração, digamos EXE para o programa/arquivo EXEMPLO.SIC, é concatenada com outros padrões de *string*, resultando em EXEGLB.H, EXEPRO.C, EXESEM.C e EXEPRS.C.

O módulo EXEGLB.H contém todas as declarações utilizadas nos outros arquivos finais. A única alteração foi a inclusão de dois novos procedimentos no SICPHSE3, o IncluiExtern e IncluiPrototipos. O primeiro se encarrega de por a palavra *extern* antes de todas as variáveis declaradas. Já o segundo, lê o arquivo de procedimentos, monta os protótipos das funções e os escreve no arquivo EXEGLB.H.

Com os *includes* para o arquivo .H seriam inseridas todas as variáveis precedidas pela palavra *extern*, o que não é desejado, já que as variáveis devem ser definidas em algum lugar para serem utilizadas. Para solucionar isso, definimos *extern* como branco para o módulo EXEPRS.C, de maneira que os outros módulos vêem suas variáveis definidas nesse módulo.

Quanto às variáveis inicializadas, utilizamos o mesmo artifício para interpretação do *extern* como branco, já que variáveis exportáveis só podem ser inicializadas na sua definição. Ou seja, no módulo de sua definição, a variável é inicializada, nos outros módulos são interpretados apenas seus protótipos. Além dos pontos citados acima, é bom colocar que foram incluídas todas as bibliotecas fornecidas pelo padrão ANSI, de forma que o usuário não precisará incluir biblioteca, a não ser que surja alguma não disponível pela biblioteca padrão e que ele queira usar. Também não é necessária a criação de um arquivo no fim da execução do SIC, como na versão Pascal.

EXEPRO.C e EXEPRS.C permanecem bem semelhante ao que eram em Pascal, não merecendo maiores comentários.

A nível de implementação, foi evitado o fato das variáveis do analisador sintático poderem se confundir com alguma outra declarada pelo usuário: precedemos todas as variáveis do *parser* por ZZ. Apesar de não ser muito comum declarações de variáveis iniciadas com ZZ, no manual do usuário existe uma alerta

ao projetista caso ele tenha a intenção de nomear qualquer variável sua iniciando com ZZ.

4.5- Interface Homem-Máquina

O código do SIC foi bastante otimizado e simplificado no que diz respeito à interface, já que tratamento de *menus*, janelas, dentre outros, é feito de maneira bem simples e direta pelo Delphi.

Incluímos formulários, as novas janelas do SIC, com componentes pré-criados pelo Delphi, botões, *menus*, etc. A parte de programação foi muito resumida, restringindo-se a algumas manipulações dos componentes que não pudessem ser feitas por meio de suas propriedades.

A interface foi desenvolvida de forma totalmente voltada para o usuário e objetivando minimização de erros. Botões e janelas são desabilitados conforme a necessidade. Dados da execução e do compilador gerado são mostrados ao projetista de forma clara.Tentativas de se listar uma gramática, por exemplo, antes de haver algum compilador gerado não são permitidas.

Além das opções disponíveis através da interface DOS, incluímos algumas outras na versão WINDOWS, como por exemplo: a) permitir que o usuário veja os arquivos gerados sem sair do programa; b) dispor de janelas para configuração de cor e impressora; etc. Todos os detalhes referentes à interface estão descritos em [Mbigonha et all, 1996].

5- Conclusão

Concluímos que a criação de uma nova versão para o Sistema de Implementação de Compiladores, SIC, é vital importância para a expansão de sua utilização.

A opção de geração de um compilador na linguagem C pode aumentar o número de usuários do sistema, já que muitas pessoas preferem a programação em C por esta ser uma linguagem bastante poderosa e eficiente.

Além disso, o desenvolvimento de uma interface amigável em um ambiente WINDOWS que está tendo cada vez mais aceitação, constitui um fator importantíssimo no aprimoramento do sistema.

A interface desenvolvida é bastante simples e não faz uso de nenhum artifício mais complicado da linguagem Object Pascal. Porém, tentamos estabelecer para o usuário um meio agradável e de fácil utilização para que o uso do SIC na construção de compiladores seja cada vez maior.

Salientamos ainda, como trabalhos futuros: a implementação de todo o sistema em C, que faria com que o mesmo pudesse ser utilizado também no ambiente UNIX; a criação de um *Help on-line*, que é de suma importância para os usuários do sistema; o acoplamento de analisadores léxicos para que o projeto do compilador seja mais modularizado; tradução de outros analisadores sintáticos; etc.

6- Bibliografia

[Aho et al, 1986]

AHO, Alfred V.; SETHI, R.; ULLMAN, J. D. Compilers: Principles, Techniques, and Tools. [s.l.], Addison Wesley, 1986.

[Allen, 1990]

Allen I. Holub. Compiler Design in C. Prentice-Hall International Editions, 1990.

[Bigonha & 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, Sociedade Brasileira de Computação, Belo Horizonte, Minas Gerais, 158-171, 1985.

[Bigonha & Bigonha, 1983]

BIGONHA, Roberto S. & BIGONHA, Mariza A. S. Um Método de Compactação de Tabelas LR(1), Anais do III Seminário sobre Desenvolvimento de Software Básico, Sociedade Brasileira de Computação, Rio de Janeiro, RJ, 141-157, 1983.

[Bigonha & 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.

[Bigonha, 1985]

BIGONHA, Mariza A. S. SIC: Sistema de Implementação de Compiladores, Código Fonte, 1985.

[Bigonha & Bigonha, 1988]

BIGONHA, Mariza A. S. & BIGONHA, Roberto S. SIC: Uma Ferramenta Para Implementação de Linguagens, Anais do XXI Congresso Nacional de Informática, SUCESU, Rio de Janeiro, RJ, 426-431, 1988.

[Bigonha, 1990]

BIGONHA, Roberto S. Window 2.0: Um Sistema Básico de Gerência de Interface; RT 011/90, DCC, UFMG, Belo Horizonte, 1990.

[Bigonha, 1995]

BIGONHA, Mariza A. S. Sistema Gerador de Geradores de Código para Arquiteturas Superescalares. Anais do VII SBAC-PAD, Canela- RS, julho 1995.

[Borland, 1987/1988]

BORLAND. TURBO PASCAL. Borland International, 1987/88.

[Costa, 1990]

Costa, P.S.S. Um Gerador Automático de Geradores de Código. Tese de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, 1990.

[Fisher & LeBlanc, 1991]

Fisher, Charles N. & LeBlanc, Richard Y. Crafting a Compiler with C. The Benjamim Cummings, 1991.

[Jensen & Wirth, 1988]

JENSEN, Kathleen & WIRTH, Nicklaus. PASCAL ISO: Manual do usuário e relatório. trad. Regina Szwarcfitor. Rio de Janeiro, Campus, 1988.

[Johnson, 1975]

JOHNSON, S. C.; Yacc - yet another compiler compiler; Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, 1975.

[Kernighan & Ritchie, 1988]

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

[Kristensen, 1981]

KRISTENSEN, B. B. & MADSEN, O. L. Methods for computing LALR(k) lookahead. TOPLAS 3:1, 60-82.

[Lesk, 1975]

LESK, M. E.; Lex - a lexical analyzer generator. Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, 1975.

[Mbigonha et all, 1996]

BIGONHA, Mariza A. S.; COSTA, Marco R.; RUSSO, Valeska G. Transporte do Sistema SIC de MD-DOS para ambiente WINDOWS, Manual do Usuário, Versão 1.0 C, Relatório Técnico RT/020-96.

Belo Horizonte, 21 de junho de 1996

Mariza Andrade da Silva Bigonha

Marco Rodrigo Costa

Valeska Gonçalves Russo