Trabalho 2

Bianca Madoka Shimizu Oe
Luís Fernando Dorelli de Abreu
Rafael Umino Nakanishi

4 de junho de 2012

1 Objetivo

Neste relatório descreveremos a utilização do analisador sintático para a LALG. Apresentaremos as pricipais decisões de projetos e as respectivas justificativas. A seguir uma breve explicação sobre o analisador utilizado e alguns exemplos que foram usados para fins de teste.

Esse relátório está organizado da seguinte forma. Na Seção 2 são discutidas as mudanças no analisador léxico implementadas para fornecer suporte ao analisador sintático. Na Seção 3 são apresentadas as decições de projeto e detalhes de implementação relacionadas ao analisador sintático produzido. Na Seção 4 encontram-se as instruções para compilação e execução do analisador sintático. Por fim, na Seção 5 são apresentados exemplos de entrada (programas LALG possivelmente incorretos) e suas respectivas saídas.

2 Mudanças Realizadas

Para dar suporte ao analisador sintático, foi adicionado ao analisador léxico da etapa anterior um contador de linhas, de forma que, quando um erro sintático for encontrado, seja possível mostrar a linha exata onde o erro ocorreu. A contagem de linhas é independente da presença de comentários: as linhas de comentário também são computadas.

Também em relação a entrega anterior, foi adicionada a análise léxica a detecção de comentários não fechados. Nesse caso, a contagem de linhas é parada, para que, quando o erro for detectado, seja mostrada a primeira linha do comentário, e não a última.

3 Decisões de Projeto

3.1 Implementação

Para implementação de um analisador sintático para a linguagem LALG, optamos pela utilização do YACC (*Yet Another Compiler Compiler*). Essa decisão foi influenciada pela simplicidade e elegância habilitadas pela ferramenta, especialmente no que diz respeito a descrição da gramática.

A implementação do analisador supracitado é dividida em três seções: declarações, regras e programas, separadas por dois símbolos de porcentagem (`%%`). A seguir são discutidos o conteúdo implementado em cada seção.

3.1.1 Declarações

Nessa seção são declarados todos os tokes utilizados pelo analisador sintático. Todos os tokens declarados são importados da fase anterior do projeto e podem ser vistas nos Quadros 1, 2 e 3.

Para declaração utilizamos o recurso do YACC `%token`, seguido dos nomes dos tokes e um nome semântico para os mesmos. Esse nome semântico é substituído nas mensagens de erros para uma apresentação mais amigável ao usuário.

Nome do operador	Token	Símbolo
Atribuição	op_at	:=
Igualdade	op_eq	=
Diferença	op_df	<>
Maior ou igual	op_ge	>=
Menor ou igual	op_le	<=
Maior	op_gr	>
Menor	op_ls	<
Soma	op_pl	+
Subtração	op_mi	-
Multiplicação	op_ml	*
Divisão	op_dv	/

Tabela 1: Quadro de operadores

Nome do símbolo	Token	Símbolo
Ponto e vírgula	sb_pv	;
Ponto final	sb_pf	.
Dois pontos	sb_dp	:
Vírgula	sb_vg	,
Abre parênteses	sb_po	(
Fecha parênteses	sb_pc	)

Tabela 2: Quadro de símbolos

program	begin	end	const
var	real	integer	char
procedure	if	else	readln
writeln	repeat	then	until
while	function		

Tabela 3: Quadro de palavras reservadas

3.1.2 Regras

As regras de produção utilizadas são as mesmas regras disponilizadas na LALG. Elas são declaradas utilizando a sintaxe do yacc da seguinte forma:

```
<Não terminal>  :    regra_de_derivação_1
                |    regra_de_derivação_2
                |    . . .
                |    regra_de_derivação_k
                ;
```

Observe que dentro das regras de derivações são colocados os tratamentos de erros. Tais tratamentos serão explicados na seção 3.2

3.1.3 Programa

Nessa seção são implementados o tratamento de erros, que será tratado mais adiante; e a função principal. Nessa ocorre a inicialização da Trie e inserção das palavras reservadas na estrutura gerada.

3.2 Tratamento de erros

Em linhas gerais, os erros que sãp tratados pelo analisador sintático são os seguintes:

- Falta de ponto-vírgula
- Falta de identificador
- Símbolos não esperados
- Comentaŕio não fechado
- Fim de programa inexperado

Os erros são recuperados no modo pânico, ou seja, tokens são descartados até que se encontre uma regra segura o suficiente para recomeçar a análise sintática.

4 Compilação

Para compilação do projeto basta executar o arquivo `Makefile` disponibilizado. Dentro do arquivo estão presentes os seguintes comandos:

```
yacc -d -v yacc.yacc --verbose
flex lalg.l
gcc lex.yy.c trie.c -lfl -o t2
```

Para execução do compilador basta utilizar os comandos:

```
t2 <nome_do_arquivo>
```

5 Exemplos