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 relatório 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 decisões 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	do	

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

O analisador sintático identifica corretamente qualquer erro sintático. A partir da identificação de um erro, regras de erro são aplicadas para prosseguir a compilação.

Os erros são recuperados pela ação default do *YACC*, que é assumir que um erro foi recuperado ao casar três tokens e por meio da chamada a *yyerrok*, contida na regra artificial `ok`. Foi utilizada, também, a função `yyless(0)`, contida na regra artificial `less`, que retorna a última cadeia lida, para sincronização. As mensagens de erro são obtidas com maior precisão no *YACC* ao utilizar o modo *verbose*. O modo *verbose*, ao realizar a chamada à função `yyerror`, default ao ocorrer um erro, passa como parâmetro uma cadeia de caracteres contendo o token obtido e a lista de tokens esperados.

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

- Erro na declaração de constantes
- Erro na declaração de variáveis
- Erro na declaração de procedimentos
- Declaração desordenada

- Falta de ponto-vírgula[1]
- Falta de identificador
- Símbolos não esperados
- Comentário 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.

Todos as regras de erros estão comentados nos códigos fonte.

4 Compilação

Para compilação do projeto em sistemas *UNIX*, execute o arquivo `Makefile` disponibilizado por meio do comando `make`. Para o sistema operacional *Windows*, execute a seguinte sequência de comandos:

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

Para execução do analisador sintático utilize os comandos:

```
t2.exe <nome_do_arquivo>
```

Observe que o analisador só termina quando encontra um *end-of-file*.

5 Exemplos

5.1 Teste 1

Entrada

```
1  program testecerto;
2  { declaracao de constantes }
3  const constante := 3;
4  const constante2 := 3.57;
5  { declaracao de variaveis }
6  var variavel : integer;
7  var variavel2, variavel4 : real;
8  { declaracao de procedimentos }
9  { procedimento sem parametros }
10 procedure procedimento;
11 var variavel3 : char;
12 begin
13         readln(variavel);
14         if variavel > 3 then
15                 begin
16                         variavel2 := constante2;
17                         variavel := constante;
18                 end
19         else
```

[1]Em grande parte dos casos

```
20                  while variavel < 3 do
21                          begin
22                                  variavel := variavel + (variavel + 2)/3;
23                                  variavel := constante2/3 +  variavel;
24                          end;
25          writeln (variavel3);
26  end;
27  { procedimento com parametros }
28  procedure procedimento2 (a,b: integer; c:real);
29  var variavel3: integer;
30  begin
31          repeat
32                  begin
33                          variavel3 := a+b;
34                          c := variavel3 -1;
35                  end;
36          until variavel3 > 10;
37  end;
38  { inicio do programa }
39  begin
40          procedimento;
41          procedimento2 (variavel; variavel; variavel2);
42          variavel := variavel * (variavel2 + 3*variavel) / 5;
43  end.
```

Saída

Não há saída por se tratar de um programa sintaticamente correto.

5.2 Teste 2

Entrada

```
1  { teste de erros basicos }
2  program SemPontoEVirgula { falta ';' }
3  { declaracao de constantes }
4  const constante := 3 { falta ';' }
5  var variavel2:char{ falta ';' }
6  var variavel:integer;
7  { declaracao de procedimento }
8  procedure 1erro (param1:integer) { identificador invalido }
9  var variavelLocal:char;
10 begin
11         readln(variavelLocal) { falta ';' }
12         writeln (variavelLocal);
13 end;
14 procedure SPtEVgNoFim;
15 begin
16         { nao faz nada }
17 end { falta ';' }
18 { inicio do programa }
19 begin
20         SPtEVgNoFim { falta ';' }
21         variavel := 3*7+12;
```

```
22          variavel := 3*; { expressao errada }
23  end.
24
25  resto { arquivo nao acabou }
```

Saída

```
1  Erro na linha 4: 'const' inesperado, esperava ';'
2  Erro na linha 5: 'var' inesperado, esperava ';'
3  Erro na linha 6: 'var' inesperado, esperava ';'
4  Erro na linha 8: 'Numero_mal_formado' inesperado [1erro], esperava
5                                          'identificador'
6  Erro na linha 12: 'writeln' inesperado, esperava ';'
7  Erro na linha 19: 'begin' inesperado, esperava ';'
8  Erro na linha 21: 'identificador' inesperado [variavel], esperava ';'
9  Erro na linha 22: ';' inesperado, esperava 'identificador' ou '(' ou
10                                         'numero_inteiro' ou 'numero_real'
11 Erro na linha 25: 'identificador' inesperado [resto], esperava
12                                         'Fim_de_arquivo'
```

5.3 Teste 3

Entrada

```
1  program prog;
2  { teste de comentario nao fechado
3  var x: integer;
4  begin
5          readln(x);
6  end;
```

Saída

```
1  Erro na linha 2: 'Comentario_nao_fechado'
```