Trabalho 1

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

9 de abril de 2012

1 Objetivo

Nesse relatório descrevemos como o usar o analisador léxico FLEX. Nas seções seguintes relatamos as principais decisões de projeto que foram feitas e as justificativas para tais escolhas. Em seguida, uma breve explicação de como utilzar o analisador e por fim alguns exemplos.

2 Decisões de Projeto

2.1 Palavras Reservadas

Para palavras reservadas do sistema, foi criado uma *trie*(árvore de prefixos), por ser consideravelmente rápida para buscar por strings.

Cada palavra reservada é retornada com o próprio valor. Na tabela 1 estão listadas as palavras reservadas do sistema.

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

Tabela 1: Tabela de operadores

Optamos por utilizar variáveis globais para interagir com a função yylex(). Como a *yylex* deve retornar um inteiro, associamos um inteiro a cada token e criamos uma estrutura auxiliar que armazena, para cada índice, o nome do token (por exemplo, "nro_real", 2). Dessa forma, os tokens continuam com sentido semântico, bastando acessar a estrutura auxiliar. Os tokens para as palavras reservadas são armazenados diretamente na Trie; os demais tokens são dados por *defines*.

2.2 Tokens

Existem cinco tipos de tokens que foram usados: operadores, símbolos, identificadores, números e erros. Cada um desses tokens são retornados como:

$$< palavra\_reservada > \quad < token >$$

Além disso, para cada classe dos tokens foram escolhidos alguns padrões que são descritos nas tabelas 2 e 3. Os identificadores são retornados como *ident* e dígitos são descritos como *nro_inteiro* ou *nro_real*.

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 2: Tabela 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 3: Tabela de símbolos

2.3 Comentários

Para análise de comentários consideramos que eles podem ter mais de uma linha. Com isso, o analisador vai consumindo todos os valores que estiverem entre dos caracteres de comentário("{"e "}"), inclusive valores de tabulação, espaços vazios e quebra de linhas.

2.4 Erros

Os erros também são retornados com tokens. Os erros que são tratados na análise léxica podem ser vistos na tabela 4.

Nome do Erro	Token	Causa
Número mal formado	er_nmf	Dígitos e caracteres concatenados. Ex.: 12a34b5
Caractere inexistente	er_cin	Caracteres que não são dígitos, letras, operadores ou símbolos
Tamanho de variável muito grande	er_idg	Identificador com mais de 20 caracteres
Número muito grande	er_ifl	Número está fora dos valores máximos e mínimos

Tabela 4: Tabela de erros

3 Compilação

Em anexo está um arquivo *makefile* que pode ser executado para gerar o executável. Os comandos que estão no arquivo são:

```
flex lalg.l
gcc lex.yy.c trie.c -lfl -o t1
```

Em seguida pode-se utilizar o executável *t1* juntamente com o código fonte escrito em linguagem *lalg*.

```
t1 <arquivo_fonte>
```

A saída gerada será o próprio arquivo fonte com os respectivos tokens.

4 Exemplo

4.1 Exemplo 1

```
program lalg;
{entrada}
var a: integer;
begin
readd(a, @, 1);
1.000
1a222
1a.000
```

```
1.00a0
a1ksjdhfgkjsdhfgesuhrguhsfkushdfkguh = 22222222222222222222222222222222222;
x <= a;
?

end.
```

Saída:

```
program program
lalg ident
; sb_pv
var var
a ident
: sb_dp
integer integer
; sb_pv
begin begin
readd ident
( sb_po
a ident
, sb_vg
@ er_cin
, sb_vg
1 nro_inteiro
) sb_pc
; sb_pv
1.000 nro_real
1a222 er_nmf
1a.000 er_nmf
1.00a0 er_nmf
a1ksjdhfgkjsdhfgesuhrguhsfkushdfkguh er_idg
= op_eq
22222222222222222222222222222222222 er_ifl
; sb_pv
x ident
<= op_le
a ident
; sb_pv
? er_cin
end end
. sb_pf
```

4.2 Exemplo 2

```
program lalg;
```

```
{isto aqui eh
um comentario
de entrada}
var a: integer;
begin
readd(a, @, 1a.31);
end
```

Saida:

```
program program
lalg ident
; sb_pv
var var
a ident
: sb_dp
integer integer
; sb_pv
begin begin
readd ident
( sb_po
a ident
, sb_vg
@ er_cin
, sb_vg
1a.31 er_nmf
) sb_pc
; sb_pv
end end
```