Trabalho Prático 3 - Analisador Léxico

Alunos:

- Lucas Caetano Lopes Rodrigues, matrícula: 2016006670
- Lucas Starling de Paula Salles, matrícula: 2016006697

Implementação do Analisador Léxico

Para implementação da etapa de análise léxica, utilizamos a ferramenta `flex` do `POSIX`. Essa ferramente recebe como entrada um arquivo `.lex` e retorna como saída um arquivo `lex.yy.c` contendo a implementação do analisador léxico na linguagem C.

Os *tokens* da gramática foram definidos através das expressões regulares mostradas a seguir. Discutimos, em seguida, os detalhes de escolhas para a criação dos *tokens* são discutidos em seguida.

```
// Arquivo Yylex.lex
%{
#include <stdio.h>
#include "grammar.h"
%}

%%

(if) return IF_TOKEN;
(then) return THEN_TOKEN;
(else) return ELSE_TOKEN;
(begin) return BEGIN_TOKEN;
(end) return END_TOKEN;
(do) return DO_TOKEN;
(while) return WHILE_TOKEN;
(until) return UNTIL_TOKEN;
(read) return READ_TOKEN;
(write) return WRITE_TOKEN;
(goto) return GOTO_TOKEN;
(NOT) return NOT_TOKEN;
(integer) return INTEGER_TYPE;
(real) return REAL_TYPE;
(boolean) return BOOL_TYPE;
(char) return CHAR_TYPE;
(program) return PROGRAM;

(\+|-|or) return ADDOP;
(\*|\/|div|mod|and) return MULOP;
(sin|cos|log|ord|chr|abs|sqrt|exp|eof|eoln) return FUNC;
(false|true) return BOOL_CONST;
[a-zA-Z]([a-zA-Z]|[0-9])* return IDENTIFIER;
[0-9][0-9]* return INTEGER_CONST;
[0-9][0-9]*(\.[0-9]+)?(E(\+|-)[0-9]+)? return REAL_CONST;
\'.\' return CHAR_CONST;
(<=|>=|<>|=|<|>) return RELOP;
```

```
(;|[ ];) return SEMICOLON;
: return COLON;
(,|[ ],) return COMMA;
(:=|[ ]:=) return ASSIGN;
[(] return OPEN_P;
[)] return CLOSE_P;
[ \t\n] ;
. return UNKNOWN;

%%

int yywrap(void) {
  return 0;
}
```

Inicialmente, define-se um *token* para cada palavra reservada da linguagem (`if`, `then`, `else`, `begin` e outros *statements*). Isto é feito para que, nas etapas seguintes do compilador, possamos identificar estruturas de condições e *loops* no código. Além disso, define-se um *token* para cada tipo de variável (`integer`, `real`, `boolean`, etc).

Ademais, definem-se as expressões regulares para todos os símbolos terminais da gramática, operações possíveis e funções embutidas na linguagem.

Todos os *tokens* definidos a partir das expressões regulares acima são mapeados para um valor inteiro, definidos no arquivo `grammar.h`.

Geração do Analisador Léxico e testes

Para gerar o programa que faz a análise léxica, basta executar o programa `flex` sobre o arquivo descrito acima:

```
$ flex Yylex.lex
```

O programa `flex` gera um arquivo na linguagem C, chamado `lex.yy.c`. Esse arquivo deve ser compilado para gerar o analisador léxico:

```
$ gcc -o lex-analyzer lex.yy.c -lfl
```

Realizando testes

Para a realização de testes, definimos o seguinte arquivo auxiliar `print.lex`:

```
%{
#include <stdio.h>
#include "grammar.h"
%}

%%

(if) printf("%d",IF_TOKEN);
(then) printf("%d",THEN_TOKEN);
(else) printf("%d",ELSE_TOKEN);
(begin) printf("%d",BEGIN_TOKEN);
(end) printf("%d",END_TOKEN);
(do) printf("%d",DO_TOKEN);
```

```
(while) printf("%d",WHILE_TOKEN);
(until) printf("%d",UNTIL_TOKEN);
(read) printf("%d",READ_TOKEN);
(write) printf("%d",WRITE_TOKEN);
(goto) printf("%d",GOTO_TOKEN);
(NOT) printf("%d",NOT_TOKEN);
(integer) printf("%d",INTEGER_TYPE);
(real) printf("%d",REAL_TYPE);
(boolean) printf("%d",BOOL_TYPE);
(char) printf("%d",CHAR_TYPE);
(program) printf("%d",PROGRAM);

(\+|-|or) printf("%d",ADDOP);
(\*|\/|div|mod|and) printf("%d",MULOP);
(sin|cos|log|ord|chr|abs|sqrt|exp|eof|eoln) printf("%d",FUNC);
(false|true) printf("%d",BOOL_CONST);
[a-zA-Z]([a-zA-Z]|[0-9])* printf("%d",IDENTIFIER);
[0-9][0-9]* printf("%d",INTEGER_CONST);
[0-9][0-9]*(\.[0-9]+)?(E(\+|-)[0-9]+)? printf("%d",REAL_CONST);
\'.\' printf("%d",CHAR_CONST);
(<=|>=|<>|=|<|>) printf("%d",RELOP);
(;|[ ];) printf("%d",SEMICOLON);
: printf("%d",COLON);
(,|[ ],) printf("%d",COMMA);
(:=|[ ]:=) printf("%d",ASSIGN);
[(] printf("%d",OPEN_P);
[)] printf("%d",CLOSE_P);
[ \t] ;
. printf("%d", UNKNOWN);

%%

int yywrap(void) {
  return 0;
}
```

Desta forma, ao contrário de retornar os *tokens* devidos, o programa `print.lex` faz com que os *tokens* reconhecidos sejam impressos na tela. Para testá-lo interativamente, basta gerar o arquivo `lex.yy.c` e o executável:

```
$ flex print.lex
$ gcc -o lex-tester lex.yy.c -lfl
$ ./lex-tester
var1
IDENTIFIER
12
INTEGER_CONST
10.23E+22
REAL_CONST
if a > 100 then b = 40 else b = -40
IF_TOKEN IDENTIFIER RELOP INTEGER_CONST THEN_TOKEN IDENTIFIER RELOP INTEGER_CONST
ELSE_TOKEN IDENTIFIER RELOP ADDOP INTEGER_CONST
sin ( 22 * pi )
FUNC OPEN_P INTEGER_CONST MULOP IDENTIFIER CLOSE_P
```