

T01 - Analisador Léxico

CIC0106 - Tradutores, Professora Cláudia Nalon, 2020.2

Filipe Alves do Nascimento - Matrícula 16/0070431

UnB - Universidade de Brasília, Brasília - DF, Brasil

1 Motivação

Como parte integrante do processo avaliativo da disciplina de Tradutores do departamento de Ciência da Computação da Universidade de Brasília, foi proposto pela docente a implementação de um compilador. Ao final das entregas, o mesmo deve ser capaz de produzir código intermediário para uma linguagem descrita. A linguagem proposta consiste de um *subset* da linguagem C acrescido de uma nova primitiva de dados para conjuntos e de operações que permitem trabalhar mais facilmente com esses conjuntos.

2 Desenvolvimento

A Análise Léxica é a primeira etapa de compilação. Ela é responsável por separar em tokens o código fonte e apontar erros léxicos. Como ferramenta auxiliar na geração do analisador léxico foi utilizada a linguagem FLEX para descrever as regras a serem consideradas na análise (mais informações sobre o FLEX podem ser encontradas no manual oficial da linguagem [3]). Por meio da descrição de Expressões Regulares, no FLEX, é possível realizar a associação de uma sequência de caracteres presente no código fonte com uma ação a ser executada.

O arquivo *analex.l* é o arquivo FLEX que contém a descrição do analisador léxico. Como orientado por Ullman *et. al* em [1], foram criadas declarações de expressões regulares: para cada *keyword* da linguagem; para cada operando, individualmente; um para representar todos os identificadores; para cada constante da linguagem; para cada símbolo de pontuação; para indentação; para comentários. As declarações podem ser encontradas no começo do arquivo *analex.l* enquanto as regras estão delimitadas entre os caracteres `%%`. São declaradas variáveis globais para controlar e exibir linha, coluna e contagem de erros léxicos encontrados durante a execução do analisador léxico.

2.1 Funções adicionais

- **main:** função principal do analisador léxico, é responsável por abrir o arquivo texto parâmetro da execução e por imprimir, ao final da análise léxica, a quantidade de erros encontrada durante a análise, se houverem [2].

```

1      int main(int argc, char *argv[]) {
2          yyin = fopen(argv[1], "r");
3          yylex();
4          fclose(yyin);
5          yylex_destroy();
6
7          if(error_count > 0){
8              printf("\nThe lexical analisys finished with %
9                  d errors found.\n", error_count);
10             }
11         return 0;
12     }

```

3 Arquivos Teste e Anexos

3.1 Arquivos Teste

Foram utilizados quatro arquivos de teste. Dois deles possuem os exemplos disponibilizados pela docente na descrição da linguagem e apresentam exemplos corretos de código, que não geram erros léxicos; estes arquivos estão nomeados por **ex\$R.txt** sendo \$ numerado de 1 a 2. Os outros dois arquivos teste foram compostos originalmente dos mesmos programas dos primeiros quatro arquivos citados (que eram corretos), porém foram introduzidos erros que podem ser captados pelo analisador léxico; estes arquivos estão nomeados por **ex\$W.txt** sendo \$ numerado de 1 a 2.

Os erros introduzidos nos arquivos de teste se apresentam como se segue abaixo (detalhados apenas erros léxicos).

- **ex1W.txt**: inserção de caracteres especiais em identificadores de variáveis parâmetros de funções;

```

1      set lsubsum(set s, int t@rget, int cur_su#n, set
2          ans) {
3          ...
4      }

```

- **ex2W.txt**: declaração de identificadores usando apenas caracteres especiais; utilização de aspas duplas para delimitar strings (apenas aspas simples são aceitas);

```

1      int main() {
2          ...
3          int %;
4
5          forall (% in s) {
6              ...
7              forall (val in possibleSums) add((% + val)
              in sumsWithX);

```

```

8      ...
9      if (13 in possibleSums) writeln("y"); else
10         writeln("n");
11     }

```

3.2 Anexos

Os anexos do primeiro trabalho consistem em:

- **lang_desc.txt**: anotações sobre a descrição da linguagem;
- **grammar.txt**: gramática inicial proposta para a linguagem descrita, apresentada no Apêndice A;
- **grammarBNF.txt**: gramática descrita em BNF;
- **analex.l**: arquivo FLEX com as especificações do analisador léxico;
- **lex.yy.c**: arquivo C contendo o analisador léxico, resultado da compilação do arquivo **analex.l**.

4 Instruções de compilação e execução

Para compilar o arquivo flex execute no terminal:

```
$ flex analex.l
```

isto irá gerar o programa C com o Analisador Léxico no arquivo *lex.yy.c*; então compile este arquivo usando:

```
$ gcc lex.yy.c -o analex -lf
```

para executar o analisador léxico você deve fornecer para o programa um arquivo *.txt* como input com o código escrito na linguagem especificada:

```
$ ./analex examples/ex1R.txt
```

References

1. Aho, A., Lam, M., Sethi, R., Ullman, J.: Compilers: Principles, Techniques, & Tools. 2nd edn. Pearson, Boston (2007).
2. Flex Manual: Simple Example, <https://westes.github.io/flex/manual/Simple-Examples.html#Simple-Examples>. Last accessed 17 Feb 2021.
3. Flex Manual, <https://westes.github.io/flex/manual/index.html#Top>. Last accessed 17 Feb 2021.
4. Flex (Fast Lexical Analyzer Generator), <https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/>. Last accessed 17 Feb 2021.
5. Writing a simple Compiler on my own - Lexical Analysis using Flex, <https://steemit.com/programming/@drifter1/writing-a-simple-compiler-on-my-own-lexical-analysis-using-flex>. Last accessed 17 Feb 2021.
6. Input and Output Files, http://web.mit.edu/gnu/doc/html/flex_2.html. Last accessed 17 Feb 2021.
7. An Overview of flex, with Examples, http://web.mit.edu/gnu/doc/html/flex_1.html#SEC1. Last accessed 17 Feb 2021.
8. Flex Regular Expressions, <https://people.cs.aau.dk/~marius/sw/flex/Flex-Regular-Expressions.html>. Last accessed 17 Feb 2021.

A Gramática

Melhor visualização nos arquivos anexos.

```

program → declarationList
declarationList → declarationListdeclaration | declaration
declaration → varDeclaration | funcDeclaration
varDeclaration → variable;
variable → typeSpecifierID
funcDeclaration → typeSpecifierID(parameters)compoundStmt
typeSpecifier → int | float | elem | set
parameters → parameterList | ε
parameterList → parameterList, parameter | parameter
parameter → variable
compoundStmt → localDeclarationsstatementList
localDeclarations → localDeclarationsvarDeclaration | ε
statementList → statementListstatement | ε
statement → compoundStmt | conditionalStmt | iterationStmt | expressionStmt | returnStmt
conditionalStmt → if(expression)statement | if(expression)statementelsestatement
iterationStmt → for(expression.Stmtexpression.Stmtexpression)statement
expressionStmt → expression; | ;
returnStmt → return; | returnexpressionStmt
expression → ID = expression | simpleExpression | setExpression | ioExpression
simpleExpression → logopUnarelationalExp | relationalExplogopBinrelationalExp | relationalExp
relationalExp → arithmAddExp | relOp | arithmAddExp | arithmMulExp
arithmAddExp → arithmAddExp | relOp | arithmMulExp | arithmMulExp
arithmMulExp → arithmMulExp | relOp | factor | factor
factor → (expression) | funcCall | ID | INT | FLOAT | EMPTY
logopUna → !
logopBin → && | ||
relOp → < | <= | > | >= | == | !=
ariOpAdd → + | -
ariOpMul → * | /
setExpression → setOpIn | setOpIsSet | setOpAdd | setOpRemove | setOpExists | setOpForall
setOpIn → (expressioninexpression)
setOpIsSet → isset(ID)
setOpAdd → addsetOpIn
setOpRemove → removesetOpIn
setOpExists → exists(IDinID)
setOpForall → forallsetOpInstatement
ioExpression → ioOpRead | ioOpWrite
ioOpRead → read(ID)
ioOpWrite → write(word) | writeln(word)
word → CHAR | STRING
funcCall → ID(arguments)
arguments → argsList | ε
argsList → argsList, expression | expression

```

$DIGIT \rightarrow 0 \mid \dots \mid 9$
 $INT \rightarrow DIGIT +$
 $FLOAT \rightarrow DIGIT + .DIGIT +$
 $LETTER_ \rightarrow a \mid \dots \mid z \mid A \mid \dots \mid Z \mid -$
 $ID \rightarrow LETTER_ LETTER_ * DIGIT *$
 $CHAR \rightarrow ' DIGIT \mid LETTER_ '$
 $STRING \rightarrow ' DIGIT * \mid LETTER_ *'$
 $EMPTY \rightarrow EMPTY$

KEYWORDS : if else for return read write writeln in is_set add remove exists forall
SPECIALSYMBOLS : + - */!| < <= > >== != == (), ; / * *//
COMMENTS : / * ... * // / ...