Compilador para a Linguagem *MLM*
Analisador léxico

Fábio Pereira e Henrique Pessoa

10 de Setembro de 2019

1 Introdução

Este trabalho tem como propósito a aplicação dos conceitos aprendidos ao decorrer da disciplina de Compiladores 2019-2. Para manter o conhecimento pratico em decorrência do modelo teórico apresentado pela disciplina, o trabalho foi divido em várias etapas.

Para esta primeira entrega, foi desenvolvido apenas o **analisador léxico** da linguagem **MLM** descrita na especificação do trabalho. Para isso, utilizamos a ferramenta de geração de análise léxica em C conhecida como *Flex*.

2 Análise Léxica

Para utilizar o *Gerador de Analisador Léxico* em questão, precisamos definir todas as expressões necessárias para reconhecer os padrões da linguagem e quais serão as saídas para cada padrão reconhecido pelo analisador para futura utilização na tabela de símbolos.

A estrutura geral do *Flex* é definida da seguinte forma:

```
definicoes /* opcional */
%%
regras
%%
rotinas de usuario /* opcional */
```

Para nossa implementação, utilizamos todas as três possibilidades de módulos do *Flex*, em *definições* foram agrupadas todas as expressões regulares que interpretam as sequências lidas; em *regras* agrupamos todas as formas de retorno para a criação dos **tokens** na tabela de símbolos; e em *rotinas de usuário* nessa primeira etapa apenas da analise léxica, deixamos a função **main()** e a função **yywrap()** e posteriormente para a analise semântica adicionaremos outras funcionalidades.

A Figura 1 a seguir lista todas as definições adotadas para o reconhecimento das expressões da linguagem **MLM**.

```
addop   "+"|"-"|"or"
relop   "="|"<"|"<="|">"|">="|"!="|"NOT"
mulop   "*"|"/"|"div"|"mod"|"and"

if          "if"
else        "else"
then        "then"
assign      ":="
begin       "begin"
while       "while"
until       "until"
end         "end"
do          "do"

ascii               [\40-\176]
char_constant       "'"{ascii}"'"
boolean_constant    "true"|"false"

digit       [0-9]
letter      [a-zA-Z]
identifier  {letter}({letter}|{digit})*
ident_lsit  {ident_list}","{identifier}|{identifier}

type                "integer"|"real"|"boolean"|"char"
unsigned_integer    {digit}+
sign                [+-]?
scale_factor        "E"{sign}{unsigned_integer}
unsigned_real       {unsigned_integer}("."{digit}*)?{scale_factor}?
integer_constant    {unsigned_integer}
real_constant       {unsigned_real}

%%
```

Figura 1: Definições do analisador léxico para a linguagem *MLM*.

Feito isso, transcrevemos quais eram as regras para a linguagem **MLM** da especificação da documentação para o nosso analisador léxico utilizando as seguintes expressões, Figura 2.

```
32  %%
33  {assign}              {printf("ASSIGN ");}
34  {addop}               {printf("ADDOP ");}
35  {relop}               {printf("RELOP ");}
36  {mulop}               {printf("MULOP ");}
37  {type}                {printf("TYPE ");}
38  {if}                  {printf("IF ");}
39  {else}                {printf("ELSE ");}
40  {then}                {printf("THEN ");}
41  {while}               {printf("WHILE ");}
42  {begin}               {printf("BEGIN ");}
43  {until}               {printf("UNTIL ");}
44  {end}                 {printf("END ");}
45  {do}                  {printf("DO ");}
46  {boolean_constant}    {printf("BOOLEAN_CONSTANT ");}
47  {integer_constant}    {printf("INTEGER_CONSTANT ");}
48  {real_constant}       {printf("REAL_CONSTANT ");}
49  {char_constant}       {printf("CHAR_CONSTANT ");}
50  {identifier}          {printf("IDENTIFIER ");}
51  %%
```

Figura 2: Regras do analisador léxico para a linguagem *MLM*.

Por fim, para a fase de implementação apenas do analisador léxico a rotina de usuário é responsável apenas por conter a função **main()** e o retorno padrão da linguagem. Posteriormente na fase de análise semântica adicionaremos, também, as formas de busca e geração da árvore de derivação para o código lido.

3 Execução

O fluxo de execução do *Flex* é dividido em duas etapas. Primeiro executamos o comando *flex* seguido do nosso arquivo que contém a definição para o gerador de analisador léxico e, posteriormente, executamos o comando *cc* seguido do novo arquivo gerado no formato *.yy.c*. A sequência de passos a ser executados foram agrupados no seguinte **Makefile**:

```
all:
    flex MLM.lex
    cc lex.yy.c -lfl
clean:
    rm a.out lex.yy.c
```

Este fluxo é melhor visualizado no diagrama abaixo, Figura 3.

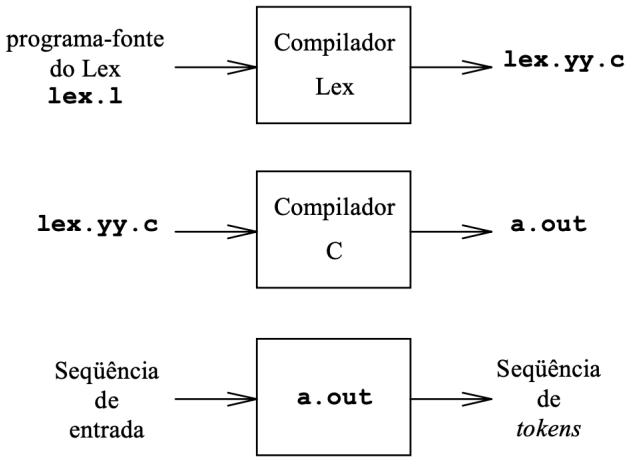

Figura 3: Fluxo de execução de um programa em *lex*.

Após a execução do *Makefile* para iniciar a analise da entrada basta utilizar o comando *./a.out* seguido de uma sequência de comandos ou um arquivo contendo o código desejado, como no exemplo abaixo.

```
./a.out < codigo.mlm
```

4 Resultados

A avaliação dos resultados do analisador léxico desenvolvido foi feita a partir de alguns exemplos de código fonte, na linguagem **MLM**, escritos para esse propósito. Alguns desses códigos serão exibidos abaixo, assim como o resultado gerado.

4.1 Exemplo 1

Nesse exemplo foi criado um código simples com único intuito demonstrativo. Nele pode-se reconhecer exemplos de quase todos tipos de constantes e operadores de relação, assim como as estruturas de condição e *loop*.

```
begin
        b : boolean
        x : real

        b := true
        x := 0.5E-1

        if x * 3 > 1 then x := x * 3

        while b := true do
                x := x - 1
                if x < 2 then b := false
        end
end

BEGIN
        IDENTIFIER  : TYPE
        IDENTIFIER  : TYPE

        IDENTIFIER  ASSIGN  BOOLEAN_CONSTANT
        IDENTIFIER  ASSIGN  REAL_CONSTANT

        IF  IDENTIFIER  MULOP  INTEGER_CONSTANT  RELOP  INTEGER_CONSTANT  THEN  IDENTIFIER  ASSIGN  IDENTIFIER  MULOP  INTEGER_CONSTANT

        WHILE  IDENTIFIER  ASSIGN  BOOLEAN_CONSTANT  DO
                IDENTIFIER  ASSIGN  IDENTIFIER  ADDOP  INTEGER_CONSTANT
                IF  IDENTIFIER  RELOP  INTEGER_CONSTANT  THEN  IDENTIFIER  ASSIGN  BOOLEAN_CONSTANT
        END
END
```

Figura 4: Primeiro exemplo de código **MLM** e saída do analisador léxico.

4.2 Exemplo 2

Como segundo exemplo, foi usado um código sem muita indentação ou separação, inclusive sem espaços entre os *tokens*. Dessa forma, mostra-se que as definições usadas para o analisador foram bem definidas.

```
begin
        c1,c2,c3:char
        c1:='a'
        c2:=c1
        c3:='z'
end

BEGIN
        IDENTIFIER ,IDENTIFIER ,IDENTIFIER :TYPE
        IDENTIFIER ASSIGN CHAR_CONSTANT
        IDENTIFIER ASSIGN IDENTIFIER
        IDENTIFIER ASSIGN CHAR_CONSTANT
END
```

Figura 5: Segundo exemplo de código **MLM** e saída do analisador léxico.

5 Conclusão

Houveram algumas dificuldades para decifrar as interfaces e formatações do programa *Flex* usado, entretanto, ao acostumar com seu funcionamento, o restante do trabalho foi desenvolvido sem muitos problemas. Com ele, foi possível entender bem como funciona um analisador léxico, assim como o uso seu uso de expressões regulares, cumprindo assim como um bom exercício prático para essa primeira parte da disciplina.

Referências

Compiladores - JFlex. Fabio Mascarenhas – 2018.1. Disponível em: ⟨https://dcc.ufrj.br/~fabiom/comp/04JFlex.pdf⟩. Acesso em 05 de setembro de 2019.

Lex - A Lexical Analyzer Generator. M. E. Lesk and E. Schmidt. Disponível em: ⟨http://dinosaur.compilertools.net/lex/⟩

Lex and Yacc: A Brisk Tutorial. Saumya K. Debray. Department of Computer Science The University of Arizona Tucson, AZ 85721. Disponível em: ⟨https://www2.cs.arizona.edu/~debray/Teaching/CSc453/DOCS/tutorial-large.pdf⟩

Compilador para a linguagem MLM. Bigonha, Mariza A. S. Disponível em: ⟨https://homepages.dcc.ufmg.br/~mariza/Cursos/CompiladoresI/Comp2019-2/Pagina/Dia-a-Dia/comp2019-2.html⟩