C7: Analisador Léxico

Marcelo G. M. M. C. de Oliveira, 12/0037301

Universidade de Brasília
http://unb.com.br

1 Motivação

Esse relatório tem como objetivo apresentar o gerador de analisadores lexicos FLEX e gerar uma base de conhecimento em assuntos relacionados a uma das etapas de uma Compilação ou Tradução. A descrição da linguágem a ser implementada foi um subconjunto da linguágem C com primitivas relacionadas ao tratamento de conjuntos.

1.1 Descrição das Novas Primitivas

Novas primitivas: 1. int 2. float 3. elem 4. set.
Operações envolvendo Conjuntos: 1. in (Pertinência), 2. is_set (Tipagem), 3. add (Inclusão de um elemento a um Conjunto), 4. remove (Remoção de um elemento em um Conjunto), 5. exists (Seleção), 6. forall (Iteração).
Constantes: Numeros de 0-9, EMPTY.
Operadores: 1. Aritméticos 2. Logicos 3. Relacionais 4. Atribuição 5. Escopo
Comentários e I/O: /**/ , // , read , write , writeln.

2 Descrição do Analisador Léxico

Para o desenvolvimento do Analisador Léxico foi utilizada a ferramenta [3] da seguinte forma:

- Geração do analisador e compilação:

```
flex clex.l
cc lex.yy.c -lfl
./a.out test_1.c
```

- TOKENS: Foram implementados seguindo o Padrão do Livro-texto [1] e da Linguágem C.

```
LETTER              [a-zA-Z]
DIGIT               [0-9]
UNDERSCORE          "_"
EOL                 \n
WS                  [ \t]+
INLINE_COMMENT      [/]{2}.*
TYPE                "int"|"float"|"elem"|"set"
```

```
INTEGER                 {DIGIT}+
FLOAT                   {DIGIT}+"."{DIGIT}+
EMPTY                   EMPTY
PLUS                    "+"
MINUS                   "-"
DIV                     "/"
MULT                    "*"
EQ                      "="
I_PLUS                  "++"
D_MINUS                 "--"
ARIT_OP                 {PLUS}|{MINUS}|{DIV}|{MULT}|{EQ}
                        |{I_PLUS}|{D_MINUS}
NOT                     "!"
OR                      "||"
AND                     "&&"
LOGIC_OP                {NOT}|{OR}|{AND}
EQ_TO                   "=="
NEQ_TO                  "!="
GT                      ">"
LT                      "<"
GTE                     ">="
LTE                     "<="
REL_OP                  {EQ_TO}|{NEQ_TO}|{GT}|{LT}|{GTE}
                        |{LTE}
KEYWORD                 "if"|"else"|"for"|"forall"|"return"
READ                    "read"
WRITE                   "write"|"writeln"
ID                      {LETTER}({LETTER}|{DIGIT}
                        |{UNDERSCORE})*
%x  STRING
```

3 Arquivos de Teste

Os testes são baseados em adicionar simbolos que não são reconhecidos pelo Analisador Léxico. Todos estão descritos no Anexo 6.

4 Próximas Etapas e Tabela de Símbolos

Na próxima etapa, vamos descrever o Analisador Semântico bem como a Tabela de Símbolos. A Tabela de Símbolos vai conter as informações sobre os identificadores, dando assim suporte para as múltiplas declarações de um identificador dentro do programa.

Referências

1. Aho, A., Lam, M., Sethi, R., Ullman, J.: Compilers: Principles, Techniques, and Tools. 2nd edn. Pearson. Boston (2007)

2. Kenneth C. Louden. 1997. Compiler Construction: Principles and Practice. PWS Publishing Co., USA.
3. Levine, J.: Flex and Bison. 2nd edn. O'Reilly, California (2009)
4. Flex Manual: Simple Example, `https://westes.github.io/flex/manual/Simple-Examples.html`. Fev 2021

5 Anexo 1: Gramática Livre-do-Contexto

Gramática adaptada do C-Minus descrita no livro [2] .

```
# Sintax of C7 on Backus-Naur form

<program> ::= <declaration-list>

<declaration-list> ::= <declaration> <declaration-list>
                     | <declaration>

<declaration> ::= <function-definition>
                | <var-declaration>

<var-declaration> ::= <type> <id> ";"

<function-definition> ::= <type> <id> "(" {<parameter-list>}? ")" <compound-statement>

<type> ::= int
         | float
         | set
         | elem

<parameter-list> ::= <parameter-declaration>
                   | <parameter-list> "," <parameter-declaration>

<parameter-declaration> ::= <type> <id>

<compound-statement> ::= "{" {<var-declaration>}* {<statement>}* "}"

<statement> ::= <expression-statement>
              | <compound-statement>
              | <if-statement>
              | <for-statement>
              | <return-statement>
              | <io-statement>
              | <set-statement>

<expression-statement> ::= <expression> ";"
                         | ";"

<if-statement> ::= if "(" <expression> ")" <statement>
                 | if "(" <expression> ")" <statement> "else" <statement>

<for-statement> ::= for "(" {<expression>}? ";" {<expression>}? ";" {<expression>}? ")" <statement>

<return-statement> ::= return {<expression>}? ";"

<io-statement> ::= read "(" <id> ")" ";"
                 | write "(" <id> ")" ";"
                 | writeln "(" <id> ")" ";"

<set-statement> ::= forall "(" <in-expression>  ")" <statement>
```

```
                | is_set "(" <id> ")" ";"


<expression> ::= <id> "=" <expression>
               | <basic-expression>
               | <set-expression>


<set-expression> ::=  "{" <element-list> "}"
                  | <set-operation>

<element-list> ::= <element-list> "," elem
                 | elem

<elem> ::= <id>
         | <integer-constant>
         | <float-constant>

<set-operation> ::= add "("<in-expression>")"
                  | remove"("<in-expression>")"
                  | exists "("<in-expression>")"

<in-expression> ::= <basic-expression> in <set-expression>
                  | <basic-expression> in <id>

<basic-expression> ::= <add-expression>
                     | <add-expression> <relop> <add-expression>

<relop> ::= "<="
          | "<"
          | ">"
          | ">="
          | "=="
          | "!="

<add-expression> ::= <term>
                   | <term> <addop> <term>
<addop> ::= "+"
          | "-"

<term> ::= <factor>
         | <term> <mulop> <factor>

<mulop> ::= "*"
          | "/"

<factor> ::= "(" <expression> ")"
           | <id>
           | <constant>

<constant> ::= <integer-constant>
             | <float-constant>
             | <empty-constant>
```

```
<id> ::= [a-zA-Z_][_a-z0-9A-Z]*

<integer-constant> ::= <digit> {<digit>}*

<float-constant> ::= <digit> "." {<digit>}+

<digit> ::= [0-9]

<empty-constant> ::= "EMPTY"
```

6 Anexo 2: Arquivos de Teste

6.1 teste_1.c

Teste que passa onde há uma declaração de um int e uma chamada de Função writeln:

```
int main(){
    int a_1;
    writeln("Hello World");
    return 0;
}
```

6.2 teste_2.c

Teste que passa aonde há uma declaração de uma variável de tipo set, ou seja, um conjunto:

```
int main(){
    set s1;

    writeln("Hello World");
    return 0;
}
```

6.3 erro_1.c

Teste que falha somente quando encontra os caracteres @, ! e $:

```

int main(){
    int 1_!a;
    set $x;
    writeln("Hello World"@);

    return 0;
}
```

6.4 erro_2.c

Teste que falha somente quando encontra o caractere # e ?:

```
int main(){
    set ?;
    writeln("Hello World"##);
    return 0;
}
```