Relatório Analisador Léxico X+++

Equipe

14104255 - Bruno Aurélio Rôzza de Moura Campos
14101370 - Fabiano Pereira de Oliveira
14101383 - Laís Ferrigo Perazzolo
14101398 - Thary Correia

Papeís no Desenvolvimento

Houve 3 encontros com **todos** os membros participando do desenvolvimento da primeira parte do trabalho.

Especificação léxica da linguagem utilizada como arquivo de entrada da ferramenta javacc

Foi utilizado a referência do livro "Como Construir um Compilador ", no Capítulo 3, Delamaro (2004) para obter a especificação léxica da linguagem. Em seguida, foi implementado as seguintes extensões:

• Operadores lógicos AND, OR, XOR e NOT;
• Novos tipos de variáveis e literais: BYTE, SHORT, LONG e FLOAT, além dos já existentes;
• Qualificadores de identificadores: FINAL, PUBLIC, PRIVATE e PROTECTED, como usado em Java.

```
/*************************************************************************
Arquivo a ser processado pelo programa JavaCC.
Contem:
       - descrição do analisador léxico para a linguagem X+++

Autor: Bruno Campos, Fabiano Oliveira, Lais Perazzolo, Thary Correa
*************************************************************************/

options {
  STATIC = false;
}

PARSER_BEGIN(langX)
package parser;

import java.io.*;


public class langX {
final static String Version = "X++ Compiler - Version 1.0 - 2004";
boolean Menosshort = false; // sa?da resumida = falso


// Define o método "main" da classe langX.
  public  static void main(String args[]) throws ParseException
  {
```

```
    String filename = ""; // nome do arquivo a ser analisado
    langX parser;     // analisador l?xico/sint?tico
    int i;
    boolean ms = false;

    System.out.println(Version);
    // lê os parâmetros passados para o compilador
    for (i = 0; i < args.length - 1; i++)
    {
    if ( args[i].toLowerCase().equals("-short") )
      ms = true;
    else
        {
           System.out.println("Usage is: java langX [-short] inputfile");
           System.exit(0);
        }
    }

    if (args[i].equals("-"))
    {       // lê da entrada padrão
        System.out.println("Reading from standard input . . .");
        parser = new langX(System.in);
     }
     else
    {       // lê do arquivo
        filename = args[args.length-1];
        System.out.println("Reading from file " + filename + " . . .");
        try {
            parser = new langX(new java.io.FileInputStream(filename));
        }
    catch (java.io.FileNotFoundException e) {
               System.out.println("File " + filename + " not found.");
        return;
        }
    }
    parser.Menosshort = ms;
    parser.program();   // chama o método que faz a an?lise
    if ( parser.token_source.foundLexError() != 0 ) // verifica se houve
erro léxico
        System.out.println(parser.token_source.foundLexError() + " Lexical
Errors found");
    else
        System.out.println("Program successfully analyzed.");
}  // main

static public String im(int x)
{
int k;
String s;
   s = tokenImage[x];
   k = s.lastIndexOf("\"");
   try {s = s.substring(1,k);}
   catch (StringIndexOutOfBoundsException e)
   {}
```

```
   return s;
}

}  // langX

PARSER_END(langX)

TOKEN_MGR_DECLS :
{
int countLexError = 0;

public int foundLexError()
{
   return countLexError;
}

}


/* Espacos a serem desprezados no inicio de cada token */

SKIP :
{
  " "
| "\t"
| "\n"
| "\r"
| "\f"
}

SKIP :
{
  "/*" : multilinecomment
}

SKIP :
{
  "//" : singlelinecomment
}

<multilinecomment> SKIP:
{
   "*/" : DEFAULT
|  <~[]>
}

<singlelinecomment> SKIP:
{
   <["\n","\r"]> : DEFAULT
|   <~[]>
}


/* Palavras reservadas */
```

```
TOKEN :
{
  < BREAK: "break" >
| < CLASS: "class" >
| < CONSTRUCTOR: "constructor" >
| < ELSE: "else" >
| < EXTENDS: "extends" >
| < FOR: "for" >
| < IF: "if" >
| < INT: "int" >
| < NEW: "new" >
| < PRINT: "print" >
| < READ: "read" >
| < RETURN: "return" >
| < STRING: "string" >
| < SUPER: "super" >
| < BYTE: "byte" >
| < SHORT: "short" >
| < LONG: "long" >
| < FLOAT: "float" >
| < FINAL: "final" >
| < PUBLIC: "public" >
| < PRIVATE: "private" >
| < PROTECTED: "protected" >
}

/* constantes */

TOKEN :
{
  < int_constant:(  // numeros decimais, octais, hexadecimais ou binarios
                 (["0"-"9"] (["0"-"9"])* ) |
                 (["0"-"7"] (["0"-"7"])* ["o", "O"] ) |
                 (["0"-"9"] (["0"-"7","A"-"F","a"-"f"])* ["h", "H"] ) |
                 (["0"-"1"] (["0"-"1"])* ["b", "B"])
                ) >
|
   < long_constant:(  // numeros long
                    (["0"-"9"] (["0"-"9"])*"F")
                    ) >

|
   < short_constant:(  // numeros short
                     (["0"-"9"] (["0"-"9"])* )
                    ) >

|
   < float_constant:(  // numeros com ponto flutuante
                     (["0"-"9"])+ "." (["0"-"9"])*
                    ) >
|
   < string_constant:    // constante string como "abcd bcda"
      "\""( ~["\"","\n","\r"])* "\"" >
```

```
|
  < null_constant: "null" > // constante null
}

/* Identificadores */

TOKEN :
{
  < IDENT: <LETTER> (<LETTER>|<DIGIT>)* >
|
  < #LETTER:["A"-"Z","a"-"z"] >
|
  < #DIGIT:["0"-"9"] >
}

/* Simbolos especiais */

TOKEN :
{
  < LPAREN: "(" >
| < RPAREN: ")" >
| < LBRACE: "{" >
| < RBRACE: "}" >
| < LBRACKET: "[" >
| < RBRACKET: "]" >
| < SEMICOLON: ";" >
| < COMMA: "," >
| < DOT: "." >
}

/* Operadores */

TOKEN :
{
  < ASSIGN: "=" >
| < GT: ">" >
| < LT: "<" >
| < EQ: "==" >
| < LE: "<=" >
| < GE: ">=" >
| < NEQ: "!=" >
| < PLUS: "+" >
| < MINUS: "-" >
| < STAR: "*" >
| < SLASH: "/" >
| < REM: "%" >
| < AND: "&&" >
| < OR: "||" >
| < XOR: "^" >
| < NOT: "!" >
}

/* Trata os erros lexicos */
SPECIAL_TOKEN :
```

```
{
<INVALID_LEXICAL:
(~ ["a"-"z", "A"-"Z",
  "0"-"9",
  "\"",
  "(",
  ")",
  "[",
  "]",
  "{",
  "}",
  ";",
  ",",
  ".",
  "=",
  ">",
  "<",
  "!",
  "+",
  "-",
  "*",
  "/",
  "%",
  " ",
  "\t",
  "\n",
  "\r",
  "\f"
])+>
   {
    System.err.println("Line " + input_stream.getEndLine() +
                           " - Invalid string found: " + image);
    countLexError++;
   }
|
   <INVALID_CONST:
   "\"" (~ ["\n","\r","\""])* ["\n","\r"]>
   {
    System.err.println("Line " + input_stream.getEndLine() +
                           " - String constant has a \\n: " + image);
    countLexError++;
   }

}



JAVACODE void program()
{
Token t;
    do
    {
    t = getNextToken();
    Token st = t;
```

```
         while ( st.specialToken != null)
         st = st.specialToken;
         do {
            if ( Menosshort )
           System.out.println(st.image + "  " +
                                im(st.kind) + " " +
                                st.kind);
            else
               System.out.println("Line: " + st.beginLine +
                                " Column: " + st.beginColumn +
                                " " + st.image +
                                "  " + im(st.kind) + "  "+t.kind);
           st = st.next;
         } while (st != t.next);
    } while (t.kind != langXConstants.EOF);
}
```

Comandos utilizados

```
sudo apt install javacc
```

- Generate parser

```
javacc langX++.jj
```

- Generate .class

```
javac parser/langX.java
```

- Testes

```
 java parser.langX -short testes_e_logs/teste_lexico.x
 java parser.langX -short testes_e_logs/teste_com_erro_lexico.x
```

Notas

- Todo o trabalho foi versionado usando a ferramenta git
- Encoding dos arquivos: US-ASCII