PROJETO PRÁTICO – Parte 1

ENTREGA: 21/10/2022 ÀS 23H59

Sumário

1 Introdução

1.1 Grupos e Ambiente de Programação

- O trabalho deve ser feito em grupo de, no máximo, 3 integrantes e deve se manter até o final do semestre.
- O trabalho deve ser implementado, **preferencialmente**, na linguagem Java.
  - O IDE *default* do projeto é o Replit Teams, especificamente na equipe da classe `ecoi26-2022-2`.
  - Ingresse no projeto disponível no link do Replit Teams – equipe de classe `ecoi26-2022-2`: `https://replit.com/team/ecoi26-2022-2/uC-Scanner-Parser`.
  - O uso de outra linguagem deve ser discutida com o professor, pois o mesmo deve ter acesso ao código-fonte do trabalho, podendo compilá-lo e executá-lo. Para submeter o projeto em outra linguagem, use o link no SIGAA
- Configuração do ambiente de execução do trabalho
  - O projeto do trabalho deve estar organizado nos pacotes (*packages*):
    - `lexer` – pacote contendo os arquivos do Analisador Léxico (`scanner`);
    - `parser` – pacote contendo os arquivos do Analisador Sintático (`parser`);
    - `symbols` – pacote contendo os arquivos relacionados à Tabela de Símbolos;
    - `main` – pacote contendo o programa principal que irá invocar o *parser* da linguagem.

1.2 Gramática da linguagem de programação μC

Um programa na linguagem-fonte μC consiste em um bloco com declarações e comandos opcionais. A gramática da linguagem μC está descrita a seguir. O símbolo não-terminal *program* é o símbolo inicial da gramática. A gramática possui recursividade à esquerda.3

$$
\begin{array}{rcl}
\textit{program} & \Rightarrow & \textit{block} \\
\textit{block} & \Rightarrow & \textbf{\{}\ \textit{decls}\ \textit{stmts}\ \textbf{\}} \\
\textit{decls} & \Rightarrow & \textit{decls}\ \textit{decl} \mid \lambda \\
\textit{decl} & \Rightarrow & \textit{type}\ \textbf{id ;} \\
\textit{type} & \Rightarrow & \textit{type}\ \textbf{[num]} \mid \textit{basic} \\
\textit{stmts} & \Rightarrow & \textit{stmts}\ \textit{stmt} \mid \lambda \\
\textit{stmt} & \Rightarrow & \textit{loc} = \textit{bool}\ \textbf{;} \\
 & \mid & \textbf{if (}\ \textit{bool}\ \textbf{)}\ \textit{stmt} \\
 & \mid & \textbf{if (}\ \textit{bool}\ \textbf{)}\ \textit{stmt}\ \textbf{else}\ \textit{stmt} \\
 & \mid & \textbf{do}\ \textit{stmt}\ \textbf{while (}\ \textit{bool}\ \textbf{);} \\
 & \mid & \textbf{while (}\ \textit{bool}\ \textbf{)}\ \textit{stmt} \\
 & \mid & \textbf{break ;} \\
 & \mid & \textbf{print (}\ \textit{bool}\ \textbf{);} \\
 & \mid & \textbf{read (}\ \textit{loc}\ \textbf{);} \\
 & \mid & \textit{block} \\
\textit{loc} & \Rightarrow & \textit{loc}\ \textbf{[}\ \textit{bool}\ \textbf{]} \mid \textbf{id} \\
\textit{bool} & \Rightarrow & \textit{bool} \mid\mid \textit{join} \mid \textit{join} \\
\textit{join} & \Rightarrow & \textit{join}\ \&\&\ \textit{equality} \mid \textit{equality} \\
\textit{equality} & \Rightarrow & \textit{equality} == \textit{rel} \mid \textit{equality}\ \textbf{!}= \textit{rel} \mid \textit{rel} \\
\textit{rel} & \Rightarrow & \textit{expr} < \textit{expr} \mid \textit{expr} <= \textit{expr} \\
 & \mid & \textit{expr} > \textit{expr} \mid \textit{expr} >= \textit{expr} \mid \textit{expr} \\
\textit{expr} & \Rightarrow & \textit{expr} + \textit{term} \mid \textit{expr} - \textit{term} \mid \textit{term} \\
\textit{term} & \Rightarrow & \textit{term} * \textit{unary} \mid \textit{term} / \textit{unary} \mid \textit{unary} \\
\textit{unary} & \Rightarrow & \textbf{!}\textit{unary} \mid -\textit{unary} \mid \textit{factor} \\
\textit{factor} & \Rightarrow & \textbf{(}\ \textit{bool}\ \textbf{)} \mid \textit{loc} \mid \textbf{num} \mid \textbf{real} \mid \textbf{false} \mid \textbf{true}
\end{array}
$$

Exemplo de um texto na linguagem-fonte μC

```
{
  int a;
  int b;
  read(a);
  read(b);
  if(a > b)
    print(a);
  else {
    print(b);
  }
}
```

Perceba que as variáveis `a` e `b` são lexemas para o *token* **id**.

2 Analisador Léxico

O analisador léxico – *scanner* é a etapa na qual textos na linguagem-fonte devem ser extraídos e traduzidos como *tokens*. Um *token* é a informação mais básica de uma linguagem. Palavras-reservadas – *keywords* como **if** possuem um lexema "`if`" e uma constante definida como `IF`, conforme o código ilustrado a seguir. A classe `Tag.java` ilustra as constantes existentes em um analisador léxico da gramática da linguagem μC.

Arquivo `Tag.java`

```
1 package lexer;
2 public class Tag {
3  public final static int AND = 256, BASIC = 257, BREAK = 258, DO = 259,
4      ELSE = 260, EQ = 261, FALSE = 262, GE = 263, ID = 264, IF = 265,
5      INDEX = 266, LE = 267, MINUS = 268, NE = 269, NUM = 270, OR = 271,
6      PRINT = 272, READ = 273, REAL = 274, TEMP = 275, TRUE = 276,
7      WHILE = 277;
8 }
```

Na Tabela 1 podem ser observados os respectivos lexemas e *tokens* que devem ser retornados pelo analisador léxico.

Lexema	**Constante relacionada**
&&	AND
`int`, `bool`, `float`, `char`	BASIC
==	EQ
>=	GE
[	INDEX
<=	LE
−	MINUS
!=	NE
Números inteiros	NUM
Números reais	REAL
||	OR

Tabela 1: Tabela de relação entre lexemas da linguagem μC e seus respectivos *tokens*.

2.1 Representações de *Tokens*, Palavras e Números

A implementação do analisador léxico deve respeitar a hierarquia de classes da Figura 1.

A classe `Token` está descrita no código a seguir.

Arquivo `Token.java`

```
1 package lexer;
2 public class Token {
3   private final int tag;
4   public Token(int id) { this.tag = id; }
5   public int getTag() { return tag; }
6   @Override public String toString() { return "" + (char) tag; }
7 }
```

As classes que aparecem na Figura 1 estão implementadas no pacote `lexer` e no pacote `symbols`.

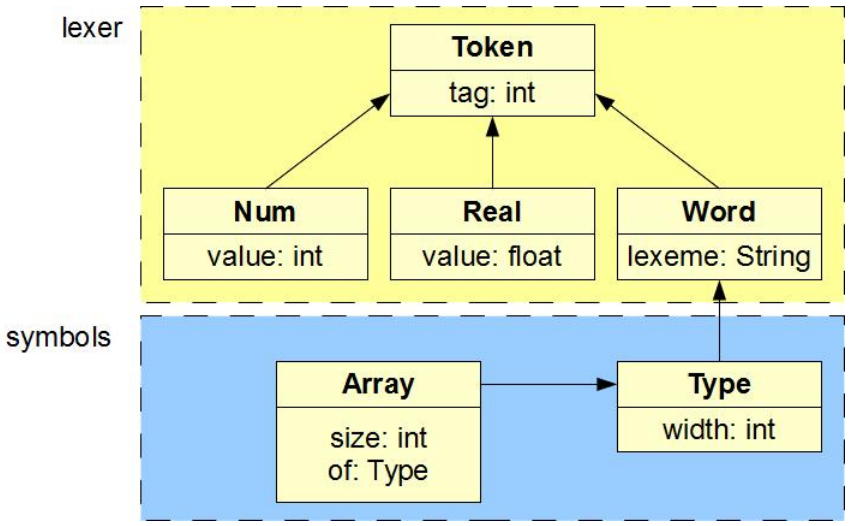

Figura 1: Classes básicas do Analisador Léxico.

2.2 Classe `Lexer`

A classe `Lexer` é responsável em conter o analisador léxico e demais informações necessárias tais como: linha do arquivo sendo processado, caracter atual do arquivo e a tabela de símbolos. O código a seguir ilustra uma parte do analisador léxico a ser implementado. **É necessário que se implemente o tratamento aos demais caracteres permitidos na linguagem μC, identificadores e números**.

Arquivo `Lexer.java`

```
1  package lexer;
2
3  import java.io.FileReader;
4  import java.io.IOException;
5  import java.util.Hashtable;
6
7  import symbols.Type;
8
9  public class Lexer {
10   private Hashtable<String, Word> words; // the hash table of words
11   private char peek = ' '; // the actual character of input file
12   private String buffer;
13   private int posBuffer = -1;
14   public static int line = 1; // the actual line of input file
15
16   public Lexer(String file) throws Exception {
17     loadFile(file);
18     words = new Hashtable<String, Word>();
19     /* keywords */
20     keyWords(new Word("if", Tag.IF));
21     keyWords(new Word("else", Tag.ELSE));
22     keyWords(new Word("while", Tag.WHILE));
23     keyWords(new Word("do", Tag.DO));
24     keyWords(new Word("break", Tag.BREAK));
25     keyWords(new Word("print", Tag.PRINT));
26     keyWords(new Word("read", Tag.READ));
```

```
27      /* boolean constants */
28      keyWords(Word.True);
29      keyWords(Word.False);
30      /* basic data types */
31      keyWords(Type.Bool);
32      keyWords(Type.Char);
33      keyWords(Type.Int);
34      keyWords(Type.Float);
35    }
36    private void loadFile(String file) throws Exception {
37      int ch = ' ';
38      FileReader fileInput = new FileReader(file);
39      StringBuffer sbuf = new StringBuffer();
40      ch = fileInput.read();
41      while (ch != -1) {
42        sbuf.append((char) ch);
43        ch = fileInput.read();
44      }
45      buffer = sbuf.toString();
46    }
47    public void readChar() {
48      ++posBuffer;
49      peek = buffer.charAt(posBuffer);
50    }
51    public void retract() {
52      --posBuffer;
53    }
54    private boolean readChar(final char ch) {
55      readChar();
56      if (peek != ch)
57        return false;
58      peek = ' ';
59      return true;
60    }
61    private void keyWords(Word word) {
62      words.put(word.getLexeme(), word);
63    }
64    public Hashtable<String, Word> getWords() {
65      return words;
66    }
67    public Token scanner() throws IOException {
68      int state = 0;
69      boolean finished = false;
70      int intValue = 0;
71      StringBuffer lexeme = new StringBuffer();
72      while (!finished) {
73        readChar();
74        switch (state) {
75        case 0:
76          switch (peek) {
77          case ' ':
```

```
78          case '\t':
79          case '\r':
80            break;
81          case '\n':
82            line++;
83            break;
84          }
85          break;
86          // estado para tratar numeros
87        case 1:
88          break;
89          // estado para tratar identificadores
90        case 2:
91          break;
92        }
93      }
94      Token token = new Token(peek);
95      peek = ' ';
96      return token;
97    }
98  }
```

3 Analisador Sintático

No código a seguir está ilustrado uma parte do analisador sintático da linguagem μC. Implemente as demais funções utilizando a estratégia de Análise Sintática Descendente Recursiva (ASD-R) Preditiva do material de aula.

Arquivo `Parser.java`

```
1  package parser;
2
3  import java.io.IOException;
4  import lexer.*;
5  import symbols.*;
6
7  public class Parser {
8    private Lexer lexer;
9    private Token lookahead;
10
11   public Parser(Lexer lexer) {
12     this.lexer = lexer;
13   }
14   private void match() throws IOException {
15     lookahead = lexer.scanner();
16   }
17   private void error(String errorMessage) {
18     throw new Error("" + Lexer.line + ": " + errorMessage);
19   }
20   private void match(Token t) throws IOException {
21     if (lookahead.getTag() == t.getTag())
```

```
22          match();
23        else
24          error("syntax␣error,␣" + t.toString() + "␣was␣expected");
25      }
26  }
```

4 Avaliação do trabalho

A avaliação do trabalho, cuja nota máxima será 10 (dez) pontos, terá os seguintes critérios:

1. Geração de arquivos de teste do trabalho;
2. Documentação do trabalho como: funções, variáveis, constantes, etc. Pode-se utilizar o Java-Doc ou similar para gerar a documentação;
3. Entrega do trabalho no prazo, com o analisador léxico e o analisador sintático funcionando para a linguagem μC;
4. A cada dia de atraso será descontado um (01) ponto da nota final do trabalho.

Bom Trabalho
Prof. *Walter*.