Implementação de um Interpretador em Mini-Pascoal e Tradução para a Linguagem de Programação ”C++”

José Maria Clementino Junior [1], **André Luiz Lima**[1],

[1]Centro de Ciências Tecnológicas – Universidade Estadual do Norte do Paraná (UENP) Bandeirantes – – Brazil

(juninhoclementino45, andre.luiz.lima.1997)@gmail.com

Resumo. *Durante o período letivo de 2017 a terceira turma de Ciência da Computação. O Prof.Me. Wellington Dellamura, da Disciplina de Compiladores propôs aos alunos, a implementação de um interpretador para a linguagem ”Mini-Pasal”e um Tradutor para a linguagem ”C++”. A documentação da implementação serão apresentados no decorrer deste artigo.*

1. Introdução

O presente artigo vai demostrar como seu deu o funcionamento de um interpretador, que foi desenvolvido seguindo as técnicas de compiladores aprendidas durante o ano letivo, juntamente com o ANTLR *(ANother Tool for Language Recognition)*, o trabalho desenvolvido foi baseado na gramática do Mini-Pascal. Para auxiliar o desenvolvimento foi utilizada também o ambiente NetBeans devido a facilidade de integração com o *plugin* do ANTLR.

2. Pré-Requisitos Para Desenvolvimento e Execução

- NetBeans
- ANTLR 4 Plugin for NetBeans
- Java Virtual Machine
- Java Virtual Machine
- GNU Compiler Collection

Para desenvolvimento, é conhecimento básico de compiladores, lógica de programação.

3. Criação da Gramática em MiniPascal

Além das produções das gramáticas, no arquivo também possui o Fragment que é utilizado para especificar para a gramática que vai aceitar tantos os tokens maiúsculos quanto minúsculos. Então a nossa gramática criada vai ser Case Sensitive.

4. Códigos

```
92   fragment A:('a'|'A');
93   fragment B:('b'|'B');
94   fragment C:('c'|'C');
95   fragment D:('d'|'D');
96   fragment E:('e'|'E');
97   fragment F:('f'|'F');
98   fragment G:('g'|'G');
99   fragment H:('h'|'H');
100  fragment I:('i'|'I');
101  fragment J:('j'|'J');
102  fragment K:('k'|'K');
103  fragment L:('l'|'L');
104  fragment M:('m'|'M');
105  fragment N:('n'|'N');
106  fragment O:('o'|'O');
107  fragment P:('p'|'P');
108  fragment Q:('q'|'Q');
109  fragment R:('r'|'R');
110  fragment S:('s'|'S');
111  fragment T:('t'|'T');
112  fragment U:('u'|'U');
113  fragment V:('v'|'V');
114  fragment W:('w'|'W');
115  fragment X:('x'|'X');
116  fragment Y:('y'|'Y');
117  fragment Z:('z'|'Z');
```

Figura 1. Fragment 1

```
grammar Pascalzinho;

@header{
package basicintast.parser;
import basicintast.util.*;
}
program : p=PROGRAM STR EOL var? procedure*           #programStmtBegin
        | PROGRAM STR EOL start                       #programStmt
        ;
var     : VAR var2;
var2    : VARNAME varn* ':' type EOL  var2 #varNameFirst | start #startL ;

varn : ',' VARNAME #varName;

type    : INT
        | FLOAT
        | BOOLEAN
        | STRING
        | arraytype
        ;

arraytype   :   ARRAY '['n1=NUM'..'n2=NUM']' OF t=INT
            |   ARRAY '['n1=NUM'..'n2=NUM']' OF t=FLOAT
            |   ARRAY '['n1=NUM'..'n2=NUM']' OF t=STRING
            |   ARRAY '['n1=NUM'..'n2=NUM']' OF t=BOOLEAN
            ;
```

Figura 2. Gramatica Parte 1

```
1  // Generated from /home/andre/Area de Trabalho/Trabalho/InterpreIraduto/grammar
2
3  package basicintast.parser;
4  import basicintast.util.*;
5
6  import org.antlr.v4.runtime.tree.AbstractParseTreeVisitor;
7
8  /**
9   * This class provides an empty implementation of {@link PascalzinhoVisitor},
10  * which can be extended to create a visitor which only needs to handle a subse
11  * of the available methods.
12  *
13  * @param <T> The return type of the visit operation. Use {@link Void} for
14  * operations with no return type.
15  */
16 public class PascalzinhoBaseVisitor<T> extends AbstractParseTreeVisitor<T> impl
17         /**
18          * {@inheritDoc}
19          *
20          * <p>The default implementation returns the result of calling
21          * {@link #visitChildren} on {@code ctx}.</p>
22          */
23         @Override public T visitProgramStmtBegin(PascalzinhoParser.ProgramStmtB
24         /**
25          * {@inheritDoc}
26          *
27          * <p>The default implementation returns the result of calling
28          * {@link #visitChildren} on {@code ctx}.</p>
29          */
30         @Override public T visitProgramStmt(PascalzinhoParser.ProgramStmtContex
31         /**
32          * {@inheritDoc}
33          *
34          * <p>The default implementation returns the result of calling
35          * {@link #visitChildren} on {@code ctx}.</p>
36          */
```

Figura 3. Visitor

```
program "test";
var
    n1, n2, n3, n4, media, exame: float;
    i: integer;
    nome: string;
    aprovado: boolean;
    a: array [0..10] of integer;

begin
for i:=0 to 10 do
begin
    a[i] := i;
    write a[i];
    write " ";
end;
write "Digite o nome do aluno: ";
readln nome;
writeln "";

i:=0;
while(i<10) do
begin
    write i;
    i:= i+1;
    write " ";
end;

writeln "";
writeln "Calculadora da média!!";

write "Nota 1: ";
readln n1;
write "Nota 2: ";
readln n2;
write "Nota 3: ";
readln n3;
write "Nota 4: ";
```

Figura 4. Código em Pascal parte 1

```
write "Nota 4: ";
readln n4;

media := (n1+n2+n3+n4)/4;
writeln media;

if(media >= 7) then
    writeln "";
    aprovado := true;
else
    aprovado := false;
    if(media>=4) then
        write "Nota Exame: ";
        readln exame;
        media := (media + exame)/2;
        if(media>=5) then
            aprovado := true;
        end;
    end;
end;
write "Nota Final: ";
writeln media;

if (aprovado) then
    write nome;
    writeln "Aprovado";
else
    write nome;
    writeln "Reprovado";
end;


end.
```

Figura 5. Código em Pascal parte 2

```
#!/bin/bash
echo Compilou
java -jar Pascalzinho.jar input.basic
```

Figura 6. Rodando projeto do interpretador do MiniPascal,

```
1  #!/bin/bash
2  echo Gerando arquivo
3  java -jar Pascalzinho.jar $1 -o $2
4  echo Compilando...
5  g++ $2 -o $3
6  echo Compilado amigo ;)
```

Figura 7. Traduzindo e compilando para o g++

```cpp
#include<iostream>
#include<string>

using namespace std;

float n1, n2, n3, n4, media, exame;
int i;
string nome;
bool aprovado;
int  a[10-0];

int main(){
i = 0;
for(i=0;i<10;i++){
a[i] = i;
cout <<a[i];
cout <<" ";
}
cout <<"Digite o nome do aluno: ";
cin >> nome;
cout <<"" << endl;
i = 0;
while(i<10){
cout <<i;
i = i+1;
cout <<" ";
}
cout <<"" << endl;
cout <<"Calculadora da média!!" << endl;
cout <<"Nota 1: ";
cin >> n1;
cout <<"Nota 2: ";
cin >> n2;
cout <<"Nota 3: ";
cin >> n3;
cout <<"Nota 4: ";
cin >> n4;
media = (n1+n2+n3+n4)/4;
```

Figura 8. Codigo Traduzido para C++,Parte 1

```
39  cout <<media << endl;
40  if(media>=7){
41  cout <<"" << endl;
42  aprovado = true;
43  }else{
44  aprovado = false;
45  if(media>=4){
46  cout <<"Nota Exame: ";
47  cin >> exame;
48  media = (media+exame)/2;
49  if(media>=5){
50  aprovado = true;
51  }
52  }
53  }
54  cout <<"Nota Final: ";
55  cout <<media << endl;
56  if(aprovado){
57  cout <<nome;
58  cout <<"Aprovado" << endl;
59  }else{
60  cout <<nome;
61  cout <<"Reprovado" << endl;
62  }
63  return 0;
64  }
```

Figura 9. Codigo Traduzido para C++,Parte 2