PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

COMPILADORES

Nomes: Diego Jornada[1]
Isadora Kurtz[2]
Marina Barros[3]

[1] 112040787
[2] 13104445
[3] 11104974

Descrição do Trabalho

O trabalho tem como objetivo, a implementação dos analisadores léxico e sintático para a linguagem Minijava. Para isso, foi utilizado a gramática BNF for MiniJava que foi disponibilizada. Também foi implementado um sistema de tratamento de erros para entradas inválidas.

Léxico

Para o léxico foram utilizadas

-Palavras Reservadas

```
"$TRACE_ON"  { yyparser.setDebug(true); }
"$TRACE_OFF" { yyparser.setDebug(false); }

"public"             { return Parser.PUBLIC; }
"static"             { return Parser.STATIC; }
"void"               { return Parser.VOID; }
"class"              { return Parser.CLASS; }
"main"               { return Parser.MAIN; }
"String"             { return Parser.STRING; }
"extends"            { return Parser.EXTENDS; }
"return"             { return Parser.RETURN; }
"int"                { return Parser.INT; }
"boolean"            { return Parser.BOOLEAN; }
"if"                 { return Parser.IF; }
"else"               { return Parser.ELSE; }
"while"              { return Parser.WHILE; }
"length"             { return Parser.LENGTH; }
"System.out.println" { return Parser.PRINT; }
"true"               { return Parser.TOP; }
"false"              { return Parser.BOTTOM; }
"this"               { return Parser.THIS; }
"new"                { return Parser.NEW; }
"&&"                 { return Parser.AND; }
```

[1] 112040787
[2] 13104445
[3] 11104974

-Tokens

```
0 | [1-9][0-9]*      { return Parser.INTEGER; }
{ID}                 { return Parser.IDENT; }
{WHITE_SPACE}+               { }

"(" |
")" |
"{" |
"}" |
"[" |
"]" |
"," |
"." |
";" |
"!" |
"=" |
"<" |
"-" |
"*" |
"+" { return (int) yycharat(0); }
```

Programas Testes

```
class Factorial{
    public static void main(String[] a){
        System.out.println(new Fac().ComputeFac(10, 4));
    }
}
class Fac {

    public int ComputeFac(int num, int y){
        int x;
        int[] z;
        return 10;
    }

}
```

Para utilizar o programa é necessário usar um Make File e os programas rodam de maneira iterativa ou recebendo um arquivo como entrada.

[1] 112040787
[2] 13104445
[3] 11104974

Tabela de Símbolos

A tabela de símbolos foi criada de uma maneira que pudesse fazer uma abstração da classe TS_Entry. A principal função é `public void listar()` e `public TS_entry pesquisa(String umId)`

```
public void listar() {
  int cont = 0;
  System.out.println("\n\nListagem da tabela de simbolos:\n");
  for (TS_entry nodo : lista) {
      System.out.println(nodo);
  }
}
```

Nesse método é feito uma lista com todos os símbolos que são declarados no TS_Entry.

```
public TS_entry pesquisa(String umId) {
  for (TS_entry nodo : lista) {
      if (nodo.getId().equals(umId)) {
          return nodo;
        }
  }
```

É feito uma pesquisa através da lista que é construída no método listar(), e essas informações podem ser buscadas através do método **parser.y** nas declarações dos Tipos e Declarações .

Class_ID

```
public enum ClasseID {
  TipoBase, VarGlobal, NomeFuncao, NomeParam, VarLocal, TipoComplexo;
}
```

No método da Class_ID, são declaradas todos os tipos de parâmetros que o compilador pode interpretar.

[1] 112040787
[2] 13104445
[3] 11104974

Semântico

Na parte do semântico, cada uma das palavras tem a sua identificação. Seguindo o exemplo abaixo o CLASS IDENT, a primeira coisa que é feita é uma pesquisa através do $2, que é definido através da entrada, caso o compilador não encontre a palavra descrita ele cria uma instância e nova, caso já existe ele mostra a mensagem de erro do **yyerror**.

```
ClassDeclaration : CLASS IDENT
                 {
                    TS_entry nodo = ts.pesquisa($2);
                    if(nodo == null) {
                        ts.insert(new TS_entry($2, Tp_OBJECT, null, ClasseID.TipoComplexo, true));
                        ts.resolveType($2);
                        currEscopo = $2;
                    }
                    else
                        yyerror("(sem) -> Classe <" + $2 + "> ja declarada");
                 }
                 Extends '{' VarDeclarationMethodeDeclaration '}' ClassDeclaration
                 |
                 ;
```

Exemplos Teste

[1] 112040787
[2] 13104445
[3] 11104974

```
class Basic {
        public static void main(String[] args) {
                System.out.println(999);
        }
}
class X{
    int size ;
    X x;
    public int Start(int s){
        int[] number ;
        int y;
        int w;
        boolean h;
        y = y + number.length;
        System.out.println(x);
        if (!(h))
            size = 1 ;
        else
            number[0] = y;
        while(y < 1){
            w = 4;
        }
        return 0;
    }
}
```

Resultado

Abaixo segue o resultado da execução.

```
Listagem da tabela de simbolos:

Id: _erro_      Classe: TipoBase        Escopo:         Tipo: null      Resolvido: true
Id: int         Classe: TipoBase        Escopo:         Tipo: null      Resolvido: true
Id: boolean     Classe: TipoBase        Escopo:         Tipo: null      Resolvido: true
Id: array       Classe: TipoBase        Escopo:         Tipo: null      Resolvido: true
Id: X           Classe: TipoComplexo    Escopo: null    Tipo: object    Resolvido: true
Id: size        Classe: null    Escopo: X       Tipo: int       Resolvido: true
Id: x           Classe: null    Escopo: X       Tipo: X Resolvido: true
Id: Start       Classe: TipoComplexo    Escopo: X       Tipo: object    Resolvido: true
Id: number      Classe: null    Escopo: Start   Tipo: array     Resolvido: true (ne: -1, tBase: null)
Id: y           Classe: null    Escopo: Start   Tipo: int       Resolvido: true
Id: w           Classe: null    Escopo: Start   Tipo: int       Resolvido: true
Id: h           Classe: null    Escopo: Start   Tipo: boolean   Resolvido: true
Have a nice day
```

[1] 112040787
[2] 13104445
[3] 11104974