**Manual Do Programador**

**1. Gramática Utilizada:**

<programa>      ::=     <decl\_list>

<decl\_list>     ::=     <decl>

<decl>          ::=     <type\_spec>ID<decl\_1>

<decl\_1>        ::=     <var\_decl><decl>|

                        <fun\_decl><decl>

<var\_decl>      ::=     ";"             |

                        "["INT\_LIT"]";" |

                        "="<exp>";"     |

                        ","ID<var\_decl>

<fun\_decl>      ::=     "("<params>")"<fun\_decl\_1>

<type\_spec>     ::=     "void"  |

                        "float" |

                        "char"  |

                        "int"   |

                        "double"

<fun\_decl\_1>    ::=     ";"|

                        <com\_stmt>

<params>        ::=     "void"  |

                        <param> |

                        <vazio>

<param>         ::=     <type\_spec>ID<param\_1>

<param\_1>       ::=     "[""]"<param\_2>             |

                        ","<type\_spec>ID<param\_1>   |

                        <vazio>

<param\_2>       ::=     ","<type\_spec>ID<param\_1>|

                        <vazio>

<stmt>          ::=     <exp\_stmt>      |

                        <com\_stmt>      |

                        <if\_stmt>       |

                        <while\_stmt>    |

                        <return\_stmt>   |

                        <break\_stmt>    |

                        <for\_stmt>      |

                        <printf\_stmt>   |

                        <scanf\_stmt>    |

                        <do\_stmt>

<exp\_stmt>      ::=     ";"     |

                        <exp>";"

<com\_stmt>      ::=     "{"<content>"}"

<content>       ::=     <local\_decls><content>  |

                        <stmt><content>         |

                        <vazio>

<do\_stmt>       ::=     "do"<stmt>"while""("<exp>")"";"

<while\_stmt>    ::=     "while""("<exp>")"<stmt>

<local\_decls>   ::=     <type\_spec>ID<local\_decl\_1>|

                        <vazio>

<local\_decl\_1>  ::=     ";"<local\_decls>        |

                        "[""]";"<local\_decls>   |

                        "="<exp>";"<local\_decls>|

                        ","ID<local\_decl\_2>

<local\_decl\_2>  ::=     ";"|

                        ","ID<local\_decl\_1>

<if\_stmt>       ::=     "if""("<exp>")"<stmt><if\_stmt\_1>

<if\_stmt\_1>     ::=     <vazio>|

                        "else"<stmt>

<for\_stmt>      ::=     "for""("<for\_stmt\_1>";"<exp>";"<exp>")"<stmt>

<for\_stmt\_1>    ::=     <type\_spec>ID=<exp>|

                        <exp>

<return\_stmt>   ::=     "return"<return\_stmt\_1>

<return\_stmt\_1> ::=     ";"     |

                        <exp>";"

<break\_stmt>    ::=     "break"";"

<exp>           ::=     ID<exp\_1><exp\_3>|

                        "+"<exp><exp\_3> |

                        "-"<exp><exp\_3> |

                        "!"<exp><exp\_3> |

                        FLOAT\_LIT<exp\_3>|

                        CHAR\_LIT<exp\_3> |

                        INT\_LIT<exp\_3>

<exp\_3>     ::=         "||"<exp><exp\_3> |

                        "="<exp><exp\_3>  |

                        "=="<exp><exp\_3> |

                        "<="<exp><exp\_3> |

                        "<"<exp><exp\_3>  |

                        ">"<exp><exp\_3>  |

                        ">="<exp><exp\_3> |

                        "&&"<exp><exp\_3> |

                        "+"<exp><exp\_3>  |

                        "-"<exp><exp\_3>  |

                        "%"<exp><exp\_3>  |

                        "\*"<exp><exp\_3>  |

                        "/"<exp><exp\_3>  |

                        "+="<exp><exp\_3> |

                        "-="<exp><exp\_3> |

                        <vazio>

<exp\_1>         ::=     "["<exp>"]"<exp\_2>  |

                        ".""sizeof"         |

                        "("<args>")"        |

                        "++"                |

                        "--"                |

                        <vazio>

<exp\_2>         ::=     "="<exp>|

                        <vazio>

<arg\_list\_1>    ::=     ","<exp><arg\_list\_1>|

                        <vazio>

<args>          ::=     <exp><arg\_list\_1>|

                        <vazio>

**2. Conversão da Gramática em Código:**

package entities;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class SintaticSemanticAnalex {

    int pointer = 0;

    private String error = "";

    private Lexema readedLexema;

    private ArrayList<Lexema> lexema;

    private final List<String> TYPE\_SPEC\_ARRAY = Arrays.asList("void", "float", "char", "int", "double");

    public String getError() {

        return this.error;

    }

    public Lexema readLexema() {

        if (this.pointer >= 0 && lexema.size() > 0 && this.pointer < lexema.size() && this.error.equals("")) {

            this.readedLexema = lexema.get(this.pointer);

            this.pointer = this.pointer + 1;

            return this.readedLexema;

        }

        this.pointer = -1;

        return new Lexema("", "", -1);

    }

    public void printError(String text) {

        if (this.error.equals("") && this.readedLexema != null) {

            this.error = text + " na linha " + this.readedLexema.getLine();

        }

    }

    public ArrayList<Lexema> filterDirectives(ArrayList<Lexema> lexema) {

        ArrayList<Lexema> newLexema = new ArrayList<Lexema>();

        int index = 0;

        for (Lexema item : lexema) {

            if (!item.getToken().equals(Token.TK\_DIRECTIV) &&

                    !item.getToken().equals(Token.TK\_CARD) &&

                    !item.getToken().equals(Token.TK\_CARDC)) {

                item.setIndex(index);

                newLexema.add(item);

            }

            index++;

        }

        return newLexema;

    }

    public SintaticSemanticAnalex(ArrayList<Lexema> lexema) {

        this.lexema = filterDirectives(lexema);

        program();

    }

    public void program() {

        decl\_list();

    }

    public void decl\_list() {

        this.decl();

    }

    public void decl() {

        if (this.var\_decl()) {

            // System.out.println("Declaraco de variavel valida");

            if (this.pointer < this.lexema.size()) {

                this.decl();

            }

        } else if (this.fun\_decl()) {

            // System.out.println("Declaracao de funcao valida");

            if (this.pointer < this.lexema.size()) {

                this.decl();

            }

        } else {

            // System.out.println("\*\*\*Fim de programa\*\*\*");

        }

    }

    public boolean var\_decl() {

        if (this.type\_spec()) {

            return true;

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean type\_spec() {

        Lexema item = readLexema();

        if (TYPE\_SPEC\_ARRAY.contains(item.getLexema())) {

            if (this.type\_spec\_id()) {

                return true;

            } else {

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            this.printError("[type\_spec] Esperava receber um tipo de dados int, float, char... ");

            return false;

        }

    }

    public boolean type\_spec\_id() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_ID) || item.getToken().equals(Token.TK\_VAR)) {

            item = readLexema();

            if (item.getToken().equals(Token.TK\_FDI)) {

                return true;

            } else if (item.getToken().equals(Token.TK\_APR)) {

                item = readLexema();

                if (item.getToken().equals(Token.TK\_FPR)) {

                    item = readLexema();

                    if (item.getToken().equals(Token.TK\_FDI)) {

                        return true;

                    } else {

                        this.pointer -= 1;

                        this.printError("[type\_spec\_id] Esperava receber ;");

                        return false;

                    }

                } else {

                    this.pointer -= 1;

                    this.printError("[type\_spec\_id] Esperava receber ]");

                    return false;

                }

            } else if (item.getToken().equals(Token.TK\_IG)) {

                if (this.exp()) {

                    item = readLexema();

                    if (item.getToken().equals(Token.TK\_FDI)) {

                        return true;

                    } else {

                        this.pointer -= 1;

                        this.printError("[type\_spec\_id-exp] Esperava receber ;");

                        return false;

                    }

                } else {

                    this.printError("[type\_spec\_id-exp] Expressao invalida");

                    return false;

                }

            } else {

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            this.printError("[type\_spec\_id] Esperava receber um identificador ou uma variavel ");

            return false;

        }

    }

    public boolean fun\_decl() {

        Lexema item = readLexema();

        if (TYPE\_SPEC\_ARRAY.contains(item.getLexema())) {

            return this.fun\_decl\_id();

        } else {

            this.printError("[fun\_decl] Esperava receber um tipo de dados ");

            pointer--;

            return false;

        }

    }

    public boolean fun\_decl\_id() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_ID) || item.getToken().equals(Token.TK\_VAR)) {

            return this.fun\_decl\_id\_open\_parent();

        } else {

            this.printError("[fun\_decl\_id] Esperava receber um ID ");

            this.pointer -= 1;

            return false;

        }

    }

    public boolean fun\_decl\_id\_open\_parent() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_AP)) {

            if (this.fun\_decl\_id\_open\_parent\_params()) {

                if (this.fun\_decl\_id\_close\_parent()) {

                    if (this.fun\_decl\_1()) {

                        return true;

                    } else {

                        return false;

                    }

                } else {

                    this.pointer -= 1;

                    return false;

                }

            } else {

                this.pointer -= 1;

                return false;

            }

        }

        this.printError("[fun\_decl\_id\_open\_parent] Esperava receber ( ");

        this.pointer -= 1;

        return false;

    }

    public boolean fun\_decl\_id\_close\_parent() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_FP)) {

            return true;

        } else {

            this.printError("[fun\_decl\_id\_close\_parent] Esperava receber ) ");

            this.pointer -= 1;

            return false;

        }

    }

    public boolean fun\_decl\_id\_open\_parent\_params() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_VOID)) {

            return true;

        } else {

            this.pointer -= 1;

        }

        if (this.param\_list\_2()) {

            return true;

        } else {

            this.printError("[fun\_decl\_id\_open\_parent\_params]  Espera receber um parametro ou void ");

            this.pointer -= 1;

            return false;

        }

    }

    public boolean fun\_decl\_1() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_FDI)) {

            return true;

        } else {

            this.pointer -= 1;

            return this.com\_stmt(false);

        }

    }

    public boolean com\_stmt(boolean isOptional) {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_ABC)) {

            if (this.local\_decls()) {

                if (this.stmt\_list()) {

                    item = readLexema();

                    if (item.getToken().equals(Token.TK\_FCH)) {

                        return true;

                    } else {

                        if (!isOptional)

                            this.printError("[com\_stmt] Esperava receber } ");

                        this.pointer -= 1;

                        return false;

                    }

                } else {

                    this.pointer -= 1;

                    return false;

                }

            } else {

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean if\_stmt() {

        Lexema item = readLexema();

        if (item.getLexema().equals("if")) {

            item = readLexema();

            if (item.getToken().equals(Token.TK\_AP)) {

                if (this.exp()) {

                    item = readLexema();

                    if (item.getToken().equals(Token.TK\_FP)) {

                        if (this.stmt()) {

                            if (this.if\_stmt\_1()) {

                                return true;

                            } else {

                                this.printError("[if\_stmt] Expressão inválida");

                                return false;

                            }

                        } else {

                            this.printError("[if\_stmt] Expressão inválida");

                            return false;

                        }

                    } else {

                        this.printError("[if\_stmt] Esperava receber )");

                        this.pointer -= 1;

                        return false;

                    }

                } else {

                    this.printError("[if\_stmt] Expresão invalida");

                    return false;

                }

            } else {

                this.printError("[if\_stmt] Esperava receber (");

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean if\_stmt\_1() {

        Lexema item = readLexema();

        if (item.getLexema().equals("else")) {

            if (this.stmt()) {

                return true;

            } else {

                return false;

            }

        } else {

            this.pointer -= 1;

            return true;

        }

    }

    public boolean while\_stmt() {

        Lexema item = readLexema();

        if (item.getLexema().equals("while")) {

            item = readLexema();

            if (item.getToken().equals(Token.TK\_AP)) {

                if (this.exp()) {

                    item = readLexema();

                    if (item.getToken().equals(Token.TK\_FP)) {

                        if (this.stmt()) {

                            return true;

                        } else {

                            this.printError("[while\_stmt-stmt] Expressão invalida");

                            return false;

                        }

                    } else {

                        this.printError("[while\_stmt] Esperava receber ) ");

                        return false;

                    }

                } else {

                    this.printError("[while\_stmt-exp] Expressão invalida");

                    return false;

                }

            } else {

                this.printError("[while\_stmt] Esperava receber (");

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean stmt() {

        if (this.exp\_stmt()) {

            return true;

        } else if (this.com\_stmt(true)) {

            return true;

        } else if (this.if\_stmt()) {

            return true;

        } else if (this.while\_stmt()) {

            return true;

        } else if (this.return\_stmt()) {

            return true;

        } else if (this.break\_stmt()) {

            return true;

        } else if (this.for\_stmt()) {

            return true;

        } else {

            return false;

        }

    }

    public boolean for\_stmt() {

        Lexema item = this.readLexema();

        if (!item.getLexema().equals("for")) {

            this.pointer -= 1;

            return false;

        }

        item = this.readLexema();

        if (!item.getToken().equals(Token.TK\_AP)) {

            this.pointer -= 1;

            this.printError("[for\_stmt] Esperava receber ( ");

            return false;

        }

        if (!this.for\_stmt\_1()) {

            this.printError("[exp] Expresão inválida ");

            return false;

        }

        item = this.readLexema();

        if (!item.getToken().equals(Token.TK\_FDI)) {

            this.pointer -= 1;

            this.printError("[for\_stmt] Esperava receber ; ");

            return false;

        }

        if (!this.exp()) {

            this.printError("[for\_stmt-exp] Expressão invalida");

            return false;

        }

        item = this.readLexema();

        if (!item.getToken().equals(Token.TK\_FDI)) {

            this.pointer -= 1;

            this.printError("[for\_stmt] Esperava receber ; ");

            return false;

        }

        if (!this.exp()) {

            this.printError("[for\_stmt-exp] Expressão invalida");

            return false;

        }

        item = this.readLexema();

        if (!item.getToken().equals(Token.TK\_FP)) {

            this.pointer -= 1;

            this.printError("[for\_stmt] Esperava receber ) ");

            return false;

        }

        if (!this.stmt()) {

            this.printError("[for\_stmt-stmt] Expressão invalida");

            return false;

        }

        return true;

    }

    public boolean for\_stmt\_1() {

        Lexema item = this.readLexema();

        if (TYPE\_SPEC\_ARRAY.contains(item.getLexema())) {

            item = this.readLexema();

            if (!item.getToken().equals(Token.TK\_ID)) {

                this.pointer -= 1;

                this.printError("Esperava receber um ID ");

                return false;

            }

            item = this.readLexema();

            if (!item.getToken().equals(Token.TK\_IG)) {

                this.pointer -= 1;

                this.printError("Esperava receber um = ");

                return false;

            }

            if (this.exp()) {

                return true;

            } else {

                return false;

            }

        } else {

            this.pointer -= 1;

        }

        if (this.exp()) {

            return true;

        } else {

            return false;

        }

    }

    public boolean break\_stmt() {

        Lexema item = this.readLexema();

        if (item.getToken().equals("break")) {

            item = this.readLexema();

            if (item.getLexema().equals(";")) {

                return true;

            } else {

                this.pointer -= 1;

                this.printError("[break\_stmt] Esperava receber ;");

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean return\_stmt() {

        Lexema item = this.readLexema();

        if (item.getLexema().equals("return")) {

            if (this.return\_stmt\_1()) {

                return true;

            } else {

                this.printError("[return\_stmt\_1] Expressão invalida");

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean return\_stmt\_1() {

        Lexema item = this.readLexema();

        if (item.getLexema().equals(";")) {

            return true;

        } else {

            this.pointer -= 1;

        }

        if (this.exp()) {

            item = this.readLexema();

            if (item.getLexema().equals(";")) {

                return true;

            } else {

                this.pointer -= 1;

                return false;

            }

        } else {

            return false;

        }

    }

    public boolean exp\_stmt() {

        Lexema item = this.readLexema();

        if (item.getToken().equals(Token.TK\_FDI)) {

            return true;

        } else {

            this.pointer -= 1;

        }

        if (this.exp()) {

            item = this.readLexema();

            if (item.getToken().equals(Token.TK\_FDI)) {

                return true;

            } else {

                this.pointer -= 1;

                return false;

            }

        } else {

            return false;

        }

    }

    public boolean stmt\_list() {

        if (this.stmt()) {

            return this.stmt\_list();

        } else {

            return true;

        }

    }

    public boolean exp() {

        Lexema item = this.readLexema();

        if (item.getToken().equals(Token.TK\_ID)) {

            if (this.exp\_1()) {

                if (this.exp\_3()) {

                    return true;

                } else {

                    this.printError("[exp-exp\_3] Expressão invalida");

                    return false;

                }

            } else {

                this.printError("[exp-exp\_1] Expressão invalida");

                return false;

            }

        } else if (item.getLexema().equals("+") || item.getLexema().equals("-") || item.getLexema().equals("!")) {

            if (this.exp()) {

                if (this.exp\_3()) {

                    return true;

                } else {

                    this.printError("[exp-exp-exp\_3] Expressão invalida");

                    return false;

                }

            } else {

                this.printError("[exp-exp] Expressão invalida");

                return false;

            }

        } else if (item.getToken().equals(Token.TK\_NF) || item.getToken().equals(Token.TK\_NI)

                || item.getToken().equals(Token.TK\_CH)) {

            if (this.exp\_3()) {

                return true;

            } else {

                this.printError("[exp-exp\_3] Expressão invalida");

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean exp\_3() {

        Lexema item = this.readLexema();

        if (item.getToken().equals(Token.TK\_OR) || item.getToken().equals(Token.TK\_IG)

                || item.getToken().equals(Token.TK\_IGIG) || item.getToken().equals(Token.TK\_MENRI)

                || item.getToken().equals(Token.TK\_MENR) || item.getToken().equals(Token.TK\_MAIOR)

                || item.getToken().equals(Token.TK\_MAIOI) || item.getToken().equals(Token.TK\_AND)

                || item.getToken().equals(Token.TK\_M) || item.getToken().equals(Token.TK\_MEN)

                || item.getToken().equals(Token.TK\_PERCENT) || item.getToken().equals(Token.TK\_AST)

                || item.getToken().equals(Token.TK\_DIV)

                || item.getToken().equals(Token.TK\_MI) || item.getToken().equals(Token.TK\_MEMI)) {

            if (this.exp()) {

                if (this.exp\_3()) {

                    return true;

                } else {

                    this.printError("[exp\_3] Expressão inválida");

                    return false;

                }

            } else {

                this.printError("[exp\_3] Expressão inválida");

                return false;

            }

        } else {

            this.pointer -= 1;

            return true;

        }

    }

    public boolean exp\_1() {

        Lexema item = this.readLexema();

        if (item.getToken().equals(Token.TK\_APR)) {

            if (this.exp()) {

                item = this.readLexema();

                if (item.getToken().equals(Token.TK\_FPR)) {

                    if (this.exp\_2()) {

                        return true;

                    } else {

                        this.printError("[exp\_1-exp-exp\_2] Expressão invalida");

                        return false;

                    }

                } else {

                    this.printError("[exp\_1-exp-exp\_2] Esperava receber ]");

                    this.pointer -= 1;

                    return false;

                }

            } else {

                this.printError("[exp\_1-exp] Esperava receber ]");

                return false;

            }

        } else if (item.getToken().equals(Token.TK\_PONT)) {

            item = this.readLexema();

            if (item.getLexema().equals("sizeof")) {

                return true;

            } else {

                this.printError("[exp\_1] Esperava receber sizeof");

                this.pointer -= 1;

                return true;

            }

        } else if (item.getToken().equals(Token.TK\_AP)) {

            if (this.args()) {

                item = this.readLexema();

                if (item.getToken().equals(Token.TK\_FP)) {

                    return true;

                } else {

                    this.pointer -= 1;

                    this.printError("[exp\_1-args] Esperava receber ]");

                    return false;

                }

            } else {

                this.printError("[exp\_1-args] Expressão inválida");

                return false;

            }

        } else if (item.getToken().equals(Token.TK\_MA)) {

            return true;

        } else if (item.getToken().equals(Token.TK\_MEM)) {

            return true;

        }

        this.pointer -= 1;

        return true;

    }

    public boolean exp\_2() {

        Lexema item = this.readLexema();

        if (item.getToken().equals(Token.TK\_IG)) {

            if (this.exp()) {

                return true;

            } else {

                this.printError("[exp\_2] Expressão invalida!");

                return false;

            }

        } else {

            this.pointer -= 1;

            return true;

        }

    }

    public boolean args() {

        if (this.arg\_list()) {

            return true;

        } else {

            return true;

        }

    }

    public boolean arg\_list() {

        if (this.exp()) {

            if (this.arg\_list\_1()) {

                return true;

            } else {

                this.printError("[arg\_list-arg\_list\_1] Expressão invalida!");

                return false;

            }

        } else {

            return false;

        }

    }

    public boolean arg\_list\_1() {

        Lexema item = this.readLexema();

        if (item.getToken().equals(Token.TK\_VIR)) {

            if (this.exp()) {

                if (this.arg\_list\_1()) {

                    return true;

                } else {

                    this.printError("[arg\_list\_1] Argumento Inválida!");

                    return false;

                }

            } else {

                this.printError("[arg\_list\_1-exp] Expressão Inválida!");

                return false;

            }

        } else {

            this.pointer -= 1;

            return true;

        }

    }

    public boolean local\_decls() {

        Lexema item = readLexema();

        if (TYPE\_SPEC\_ARRAY.contains(item.getLexema())) {

            item = readLexema();

            if (item.getToken().equals(Token.TK\_ID)) {

                if (this.local\_decl\_1()) {

                    return true;

                } else {

                    this.pointer -= 1;

                    return false;

                }

            } else {

                this.printError("[local\_decls] Esperava receber um ID");

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            return true;

        }

    }

    public boolean local\_decl\_1() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_FDI)) {

            return this.local\_decls();

        } else if (item.getToken().equals(Token.TK\_APR)) {

            item = readLexema();

            if (item.getToken().equals(Token.TK\_FPR)) {

                item = readLexema();

                if (item.getToken().equals(Token.TK\_FDI)) {

                    return this.local\_decls();

                } else {

                    this.printError("Esperava receber ;");

                    this.pointer -= 1;

                    return false;

                }

            } else {

                this.printError("Esperava receber ]");

                this.pointer -= 1;

                return false;

            }

        } else if (item.getToken().equals(Token.TK\_IG)) {

            if (this.exp()) {

                item = readLexema();

                if (item.getToken().equals(Token.TK\_FDI)) {

                    return this.local\_decls();

                } else {

                    this.pointer -= 1;

                    this.printError("[local\_decl\_1-exp] Esperava receber ;");

                    return false;

                }

            } else {

                this.printError("[local\_decl\_1-exp] Expressao invalida");

                return false;

            }

        } else {

            this.pointer -= 1;

            this.printError("[local\_decl\_1] Esperava receber ; ou [ ");

            return false;

        }

    }

    public boolean param\_list\_2() {

        if (this.param()) {

            return this.param\_list\_1();

        } else {

            return true;

        }

    }

    public boolean param() {

        Lexema item = readLexema();

        if (TYPE\_SPEC\_ARRAY.contains(item.getLexema())) {

            if (param\_type\_spec\_id()) {

                return true;

            } else {

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean param\_type\_spec\_id() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_ID)) {

            if (this.param\_type\_spec\_id\_param\_1()) {

                return true;

            } else {

                this.pointer -= 1;

                return false;

            }

        } else {

            this.pointer -= 1;

            return false;

        }

    }

    public boolean param\_type\_spec\_id\_param\_1() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_APR)) {

            return this.param\_type\_spec\_id\_param\_1\_close\_parent();

        } else {

            this.pointer -= 1;

            return true;

        }

    }

    public boolean param\_type\_spec\_id\_param\_1\_close\_parent() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_FPR)) {

            return true;

        } else {

            this.printError("[param\_type\_spec\_id\_param\_1\_close\_parent] Esperava receber ] ");

            this.pointer -= 1;

            return false;

        }

    }

    public boolean param\_list\_1() {

        Lexema item = readLexema();

        if (item.getToken().equals(Token.TK\_VIR)) {

            if (this.param()) {

                return this.param\_list\_1();

            } else {

                if (this.error.equals("")) {

                    this.printError("[param\_list\_1] Esperava receber um parametro ");

                }

                return false;

            }

        } else {

            this.pointer -= 1;

            return true;

        }

    }

}

**3. Semântico**

package entities;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class SemanticReader {

    private ArrayList<Semantic> semanticTable;

    private ArrayList<String> erros;

    private final List<String> TYPE\_SPEC\_ARRAY = Arrays.asList("void", "float", "char", "int", "double");

    public SemanticReader(ArrayList<Semantic> semanticTable) {

        this.semanticTable = semanticTable;

        this.erros = new ArrayList<>();

        this.checkDeclarationOfUsedVars();

        this.checkDoubleVarDeclaration();

        this.checkVarsValues();

        this.checkMainDeclaration();

        this.checkBooleanExpression();

        // this.checkFunctionParams();

        // this.checkPrintf();

        // this.checkSanf();

    }

    public ArrayList<String> getErros() {

        return this.erros;

    }

    boolean checkIdDeclaration(int startIndex, int endIndex, Lexema value) {

        for (int i = 0; i < endIndex; i++) {

            Semantic item = semanticTable.get(i);

            // filter var and function declaration

            if (item.getType().equals("var\_decl") ||

                    item.getType().equals("fun\_decl") ||

                    item.getType().equals("var\_decl\_equal") ||

                    item.getType().equals("fun\_decl\_prototype")) {

                ArrayList<Lexema> lexemas = item.getSignature();

                // check top scope

                if (i >= startIndex || lexemas.get(0).getScope() == 0) {

                    // check if ID exist in position 0 or 1 of declaration

                    if (lexemas.get(0).getToken().equals(Token.TK\_ID) &&

                            lexemas.get(0).getLexema().equals(value.getLexema()) ||

                            lexemas.get(1).getToken().equals(Token.TK\_ID) &&

                                    lexemas.get(1).getLexema().equals(value.getLexema())) {

                        return true;

                    }

                }

            }

        }

        return false;

    }

    Semantic getVarFunDec(int startIndex, int endIndex, Lexema id) {

        for (int i = 0; i < endIndex; i++) {

            Semantic expression = semanticTable.get(i);

            // verify scope

            if (i >= startIndex || expression.getSignature().get(0).getScope() == 0)

                // filter var and function declaration

                if (expression.getType().equals("var\_decl") ||

                        expression.getType().equals("fun\_decl") ||

                        expression.getType().equals("var\_decl\_equal") ||

                        expression.getType().equals("var\_decl\_array") ||

                        expression.getType().equals("fun\_decl\_prototype")) {

                    ArrayList<Lexema> signature = expression.getSignature();

                    // check if ID exist in position 0 or 1 of declaration

                    if (signature.get(0).getToken().equals(Token.TK\_ID) &&

                            signature.get(0).getLexema().equals(id.getLexema())) {

                        return expression;

                    } else if (signature.get(1).getToken().equals(Token.TK\_ID) &&

                            signature.get(1).getLexema().equals(id.getLexema())) {

                        return expression;

                    }

                }

        }

        return null;

    }

    // a) Verificar se a variável usada foi declarada;

    public void checkDeclarationOfUsedVars() {

        int i = 0;

        int topScope = 0;

        for (Semantic expression : this.semanticTable) {

            if (expression.getType().equals("fun\_decl"))

                topScope = i;

            if (expression.getType().equals("exp\_stmt")) {

                int j = 0;

                for (Lexema item : expression.getSignature()) {

                    if (item.getToken().equals(Token.TK\_ID)) {

                        if (!checkIdDeclaration(topScope, i, item)) {

                            Lexema nextLexema = expression.getSignature().get(j + 1);

                            String IDName = "";

                            if (nextLexema.getToken().equals(Token.TK\_AP)) {

                                IDName = "Funcao";

                            } else {

                                IDName = "Variavel";

                            }

                            this.erros.add(

                                    IDName + " " + item.getLexema() + " nao foi declarada na linha: " + item.getLine());

                        }

                    }

                    j++;

                }

            }

            i++;

        }

    }

    // c) Uma variável não deve ser declarada duas vezes no mesmo escopo;

    public void checkDoubleVarDeclaration() {

        ArrayList<Semantic> vars = this.getVarDecls();

        // filter var declaration

        for (Semantic expression : vars) {

            int count = 0;

            Lexema lexemaID = getVarID(expression);

            int scope = expression.getSignature().get(0).getScope();

            for (Semantic expression1 : vars) {

                Lexema lexemaID1 = getVarID(expression1);

                int scope1 = expression1.getSignature().get(0).getScope();

                if (scope1 == scope && lexemaID.getLexema().equals(lexemaID1.getLexema())) {

                    if (count > 0) {

                        String errorMessage = "Declaracao da variavel " + lexemaID.getLexema() + " duplicada na linha: "

                                + lexemaID1.getLine();

                        if (!erros.contains(errorMessage))

                            this.erros.add(errorMessage);

                    }

                    count++;

                }

            }

        }

    }

    public Lexema getVarID(Semantic expression) {

        return expression.getSignature().get(expression.getSignature().get(0).getToken().equals(Token.TK\_ID) ? 0 : 1);

    }

    public ArrayList<Semantic> getVarDecls() {

        ArrayList<Semantic> vars = new ArrayList<>();

        // filter var declaration

        for (Semantic expression : this.semanticTable) {

            if (expression.getType().equals("var\_decl") ||

                    expression.getType().equals("var\_decl\_array") ||

                    expression.getType().equals("var\_decl\_equal")

            ) {

                vars.add(expression);

            }

        }

        return vars;

    }

    public ArrayList<Semantic> getVarEqual() {

        ArrayList<Semantic> vars = new ArrayList<>();

        // filter var declaration

        for (Semantic expression : this.semanticTable) {

            if (expression.getType().equals("var\_decl\_equal")) {

                vars.add(expression);

            }

        }

        return vars;

    }

    public Lexema getVarDataType(Semantic expression) {

        if (TYPE\_SPEC\_ARRAY.contains(expression.getSignature().get(0).getLexema()))

            return expression.getSignature().get(0);

        for (int i = expression.getIndex(); i > 0; i--) {

            if (TYPE\_SPEC\_ARRAY.contains(this.semanticTable.get(i).getSignature().get(0).getLexema()))

                return this.semanticTable.get(i).getSignature().get(0);

        }

        return null;

    }

    // b) Compatibilidade de tipos, ou seja, uma variável do tipo inteiro não deve

    // receber por exemplo um valor do tipo string;

    public void checkVarsValues() {

        int startIndex = 0;

        for (Semantic expression : this.semanticTable) {

            if (expression.getType().equals("fun\_decl"))

                startIndex = expression.getIndex();

            // var\_decl\_equal

            if (expression.getType().equals("var\_decl\_equal")) {

                Lexema varId = this.getVarID(expression);

                Lexema varType = this.getVarDataType(expression);

                Boolean canValidate = false;

                Boolean hasIgual = false;

                for (Lexema item : expression.getSignature()) {

                    // int

                    if (varType.getLexema().equals("int")) {

                        // vaild token

                        if (canValidate) {

                            if (item.getToken().equals(Token.TK\_NI) ||

                                    item.getToken().equals(Token.TK\_VIR) ||

                                    item.getToken().equals(Token.TK\_FDI) ||

                                    item.getToken().equals(Token.TK\_M) ||

                                    item.getToken().equals(Token.TK\_MEN) ||

                                    item.getToken().equals(Token.TK\_AST)

                            ) {

                                canValidate = true;

                            } else if (item.getToken().equals(Token.TK\_AP) || item.getToken().equals(Token.TK\_APR)) {

                                canValidate = false;

                            } else if (item.getToken().equals(Token.TK\_DIV)) {

                                this.erros.add("Operador " + item.getLexema()

                                        + " Nao pode ser usado na linha: "

                                        + item.getLine());

                                canValidate = false;

                            } else if (item.getToken().equals(Token.TK\_ID)) {

                                Semantic idExpression = getVarFunDec(startIndex, expression.getIndex(), item);

                                if (idExpression == null) {

                                    this.erros.add("Variavel/Funcao " + item.getLexema() + " Nao declarado na linha: "

                                            + item.getLine());

                                    this.erros.add(

                                            "Tipo de dados da variavel/funcao " + varId.getLexema()

                                                    + " nao identificado na linha: "

                                                    + item.getLine());

                                    canValidate = false;

                                } else {

                                    Lexema idToken = getVarID(idExpression);

                                    Lexema idType = getVarDataType(idExpression);

                                    if (!idType.getLexema().equals("int")) {

                                        this.erros.add(

                                                "Tipo de dados da funcao/variavel " + idToken.getLexema()

                                                        + " invalido na linha: "

                                                        + item.getLine());

                                        canValidate = false;

                                    }

                                }

                            } else {

                                this.erros.add(

                                        "Valor invalido para variavel " + varId.getLexema() + " na linha: "

                                                + item.getLine());

                                canValidate = false;

                            }

                        } else if (item.getToken().equals(Token.TK\_FP) || item.getToken().equals(Token.TK\_FPR)) {

                            canValidate = true;

                        } else if (item.getToken().equals(Token.TK\_ID) && hasIgual && !canValidate) {

                            Semantic idExpression = getVarFunDec(startIndex, expression.getIndex(), item);

                            if (idExpression == null) {

                                this.erros.add(

                                        "Variavel/funcao " + item.getLexema() + " nao declarada na linha: "

                                                + item.getLine());

                            }

                        }

                    }

                    // float, double

                    if (varType.getLexema().equals("float") || varType.getLexema().equals("double")) {

                        // vaild token

                        if (canValidate) {

                            if (item.getToken().equals(Token.TK\_NF) ||

                                    item.getToken().equals(Token.TK\_NI) ||

                                    item.getToken().equals(Token.TK\_VIR) ||

                                    item.getToken().equals(Token.TK\_FDI) ||

                                    item.getToken().equals(Token.TK\_M) ||

                                    item.getToken().equals(Token.TK\_MEN) ||

                                    item.getToken().equals(Token.TK\_AST) ||

                                    item.getToken().equals(Token.TK\_DIV)

                            ) {

                                canValidate = true;

                            } else if (item.getToken().equals(Token.TK\_AP) || item.getToken().equals(Token.TK\_APR)) {

                                canValidate = false;

                            } else if (item.getToken().equals(Token.TK\_ID)) {

                                Semantic idExpression = getVarFunDec(startIndex, expression.getIndex(), item);

                                if (idExpression == null) {

                                    this.erros.add("Variavel/Funcao " + item.getLexema() + " Nao declarado na linha: "

                                            + item.getLine());

                                    this.erros.add(

                                            "Tipo de dados da variavel/funcao " + varId.getLexema()

                                                    + " nao identificado na linha: "

                                                    + item.getLine());

                                    canValidate = false;

                                } else {

                                    Lexema idToken = getVarID(idExpression);

                                    Lexema idType = getVarDataType(idExpression);

                                    if (!idType.getLexema().equals("float") &&

                                            !idType.getLexema().equals("double") &&

                                            !idType.getLexema().equals("int")) {

                                        this.erros.add(

                                                "Tipo de dados da funcao/variavel " + idToken.getLexema()

                                                        + " invalido na linha: "

                                                        + item.getLine());

                                        canValidate = false;

                                    }

                                }

                            } else {

                                this.erros.add(

                                        "Valor invalido para variavel " + varId.getLexema() + " na linha: "

                                                + item.getLine());

                                canValidate = false;

                            }

                        } else if (item.getToken().equals(Token.TK\_FP) || item.getToken().equals(Token.TK\_FPR)) {

                            canValidate = true;

                        } else if (item.getToken().equals(Token.TK\_ID) && hasIgual && !canValidate) {

                            Semantic idExpression = getVarFunDec(startIndex, expression.getIndex(), item);

                            if (idExpression == null) {

                                this.erros.add(

                                        "Variavel/funcao " + item.getLexema() + " nao declarada na linha: "

                                                + item.getLine());

                            }

                        }

                    }

                    // char

                    if (varType.getLexema().equals("char")) {

                        // vaild token

                        if (canValidate) {

                            if (item.getToken().equals(Token.TK\_CH)) {

                                canValidate = false;

                            } else if (item.getToken().equals(Token.TK\_ID)) {

                                canValidate = false;

                                Semantic idExpression = getVarFunDec(startIndex, expression.getIndex(), item);

                                if (idExpression == null) {

                                    this.erros.add("Variavel/Funcao " + item.getLexema() + " Nao declarado na linha: "

                                            + item.getLine());

                                    this.erros.add(

                                            "Tipo de dados da variavel/funcao " + varId.getLexema()

                                                    + " nao identificado na linha: "

                                                    + item.getLine());

                                } else {

                                    Lexema idToken = getVarID(idExpression);

                                    Lexema idType = getVarDataType(idExpression);

                                    if (!idType.getLexema().equals("char")) {

                                        this.erros.add(

                                                "Tipo de dados da funcao/variavel " + idToken.getLexema()

                                                        + " invalido na linha: "

                                                        + item.getLine());

                                    }

                                }

                            } else {

                                this.erros.add(

                                        "Valor invalido para variavel " + varId.getLexema() + " na linha: "

                                                + item.getLine());

                                canValidate = false;

                            }

                        } else if (item.getToken().equals(Token.TK\_VIR) ||

                                item.getToken().equals(Token.TK\_FDI)) {

                            canValidate = false;

                        } else if (hasIgual && !canValidate) {

                            this.erros.add(

                                    "Valor invalido para variavel " + varId.getLexema() + " na linha: "

                                            + item.getLine());

                            hasIgual = false;

                        }

                    }

                    // others

                    if (item.getToken().equals(Token.TK\_IG)) {

                        canValidate = true;

                        hasIgual = true;

                    }

                }

            }

        }

    }

    // g) Fazer também a verificação do identificador principal (main) se foi bem

    // declarada;

    public void checkMainDeclaration() {

        boolean hasMain = false;

        for (Semantic expression : this.semanticTable) {

            // var\_decl\_equalfun\_decl

            if (expression.getType().equals("fun\_decl")) {

                Lexema varId = this.getVarID(expression);

                Lexema varType = this.getVarDataType(expression);

                if (varId.getLexema().equals("main")) {

                    hasMain = true;

                    // data type

                    if (!varType.getLexema().equals("int")) {

                        this.erros.add(

                                "O tipo de dados da funcao main precisa ser inteiro! na linha: "

                                        + varType.getLine());

                    }

                }

            }

        }

        if (!hasMain)

            this.erros.add("funcao main nao foi declarada!");

    }

    public void checkBooleanExpression() {

        for (Semantic expression : this.semanticTable) {

            if (expression.getType().equals("if\_stmt") ||

                    expression.getType().equals("while\_stmt") ||

                    expression.getType().equals("do\_while\_stmt")) {

                boolean isBoolean = false;

                for (Lexema item : expression.getSignature()) {

                    if (Arrays.asList("==", ">=", "<=", "&&", "||", "!=", ">", "<").contains(item.getLexema()))

                        isBoolean = true;

                }

                if (!isBoolean && expression.getSignature().size() != 4) {

                    this.erros

                            .add("Expressao boleana faltando na linha: " + expression.getSignature().get(0).getLine());

                }

            }

            if (expression.getType().equals("for\_stmt")) {

                boolean isBoolean = false;

                boolean canValidate = false;

                for (Lexema item : expression.getSignature()) {

                    if (item.getLexema().equals(";"))

                        canValidate = !canValidate;

                    if (canValidate

                            && Arrays.asList("==", ">=", "<=", "&&", "||", "!=", ">", "<").contains(item.getLexema()))

                        isBoolean = true;

                }

                if (!isBoolean) {

                    this.erros

                            .add("Expressao boleana faltando na linha: " + expression.getSignature().get(0).getLine());

                }

            }

        }

    }

    public void checkFunctionParams() {

        int startIndex = 0;

        for (Semantic expression : this.semanticTable) {

            if (expression.getType().equals("fun\_decl"))

                startIndex = expression.getIndex();

            if (!expression.getType().equals("exp\_stmt"))

                continue;

            for (int i = 0; i < expression.getSignature().size(); i++) {

                if (expression.getSignature().get(i).getToken().equals(Token.TK\_ID)

                        && expression.getSignature().get(i + 1).getToken().equals(Token.TK\_AP)) {

                    ArrayList<Lexema> listParams = new ArrayList<>();

                    ArrayList<Lexema> listParamsTypes = new ArrayList<>();

                    Semantic sintaxiFun = getVarFunDec(startIndex, i, expression.getSignature().get(i));

                    int j;

                    for (j = i; j < expression.getSignature().size()

                            && expression.getSignature().get(j).getLexema().equals(")"); j++) {

                        if (!expression.getSignature().get(j).getLexema().equals("(")

                                && !expression.getSignature().get(j).equals(")")

                                && !expression.getSignature().get(j).equals(","))

                            listParams.add(expression.getSignature().get(j));

                    }

                    i = j;

                }

            }

        }

    }

    public void validateList() {

    }

}