A linguagem *Pyscal* (Corrigida com ações semânticas):

Programa → Classe **EOF** 1
Classe → "class" ID {TS.setTipo(ID.lexval, vazio)} ":" ListaFuncao Main "end" "." 2
DeclaraID → TipoPrimitivo ID { TS.setTipo(ID.lexval, TipoPrimitivo.t) } ";" 3
ListaFuncao → ListaFuncao' 4
ListaFuncao' → Funcao ListaFuncao' 5 | ε 6
Funcao → "def" TipoPrimitivo ID {TS.setTipo(ID.lexval, TipoPrimitivo.t)} "(" ListaArg ")" ":"
RegexDeclaraId ListaCmd Retorno {se Retorno.t != TipoPrimitivo.t: **sinalizar erro para tipo de retorno incompatível**} "end" ";" 7
RegexDeclaraId → DeclaraID RegexDeclaraId 8 | ε 9
ListaArg → Arg ListaArg' 10
ListaArg' → "," ListaArg 11 | ε 12
Arg → TipoPrimitivo ID {TS.setTipo(ID.lexval, TipoPrimitivo.t)} 13
Retorno → "return" Expressao ";" {Retorno.t = Expressao.t} 14 | ε {Retorno.t = vazio} 15
Main → "defstatic" "void" "main" "(" "String" "[" "]" ID {TS.setTipo(ID.lexval, string)} ")" ":"
RegexDeclaraId ListaCmd "end" ";" 16
TipoPrimitivo → "bool" {TipoPrimitivo.t = logico} 17 | "integer" {TipoPrimitivo.t = numerico} 18
| "String" {TipoPrimitivo.t = string} 19 | "double" {TipoPrimitivo.t = numerico} 20
| "void" {TipoPrimitivo.t = vazio} 21
ListaCmd → ListaCmd' 22
ListaCmd' → Cmd ListaCmd' 23 | ε 24
Cmd → CmdIF 25 | CmdWhile 26 | CmdWrite 28
| ID {se TS.getTipo(ID.lexval) == null:
sinalizar erro para ID não declarado}
CmdAtribFunc {se CmdAtribFunc.t != vazio &&
TS.getTipo(ID.lexval) != CmdAtribFunc.t:
sinalizar erro de atribuição incompatível} 27
CmdAtribFunc → CmdAtribui {CmdAtribFunc.t = CmdAtribui.t} 29
| CmdFuncao {CmdAtribFunc.t = vazio} 30
CmdIF → "if" "(" Expressao ")" {se Expressao.t != logico: **sinalizar erro**}
":" ListaCmd CmdIF' 31
CmdIF' → "end" ";" 32 | "else" ":" ListaCmd "end" ";" 33
CmdWhile → "while" "(" Expressao ")" {se Expressao.t != logico: **sinalizar erro**}
":" ListaCmd "end" ";" 34
CmdWrite → "write" "(" Expressao ")" ";" {se Expressao.t != string: **sinalizar erro**} 35
CmdAtribui → "=" Expressao ";" {CmdAtribui.t = Expressao.t} 36
CmdFuncao → "(" RegexExp ")" ";" 37
RegexExp → Expressao RegexExp' 38 | ε 39
RegexExp' → , Expressao RegexExp' 40 | ε 41
Expressao → Exp1 Exp' {se Exp'.t == vazio: Exp.t = Exp1.t;
senao se Exp'.t == Exp1.t && Exp'.t == logico: Exp.t = logico;
senao: Exp.t = **tipo_erro**} 42
Exp' → "or" Exp1 Exp' 43 | "and" Exp1 Exp' 44
// ação semantica vale para as regras 43 e 44
{se Exp'Filho.t == vazio && Exp1.t == logico: Exp'.t = logico
senao se Exp'Filho.t == Exp1.t && Exp1.t == logico: Exp'.t = logico;
senao: Exp'.t = **tipo_erro**}

| ε {Exp'.t = vazio} 45

Exp1 → Exp2 Exp1'
{se Exp1'.t == vazio: Exp1.t = Exp2.t;
senao se Exp1'.t == Exp2.t && Exp1'.t == numerico: Exp1.t = logico;
senao: Exp1.t = **tipo_erro**} 46

Exp1' → "<" Exp2 Exp1' 47 | "<=" Exp2 Exp1' 48 | ">" Exp2 Exp1' 49
| ">=" Exp2 Exp1' 50 | "==" Exp2 Exp1' 51 | "!=" Exp2 Exp1' 52
// ação semantica vale para as regras 47, 48,..., 52
{se Exp1'Filho.t == vazio && Exp2.t == numerico: Exp1'.t = numerico;
senao se Exp1'Filho.t == Exp2.t && Exp2.t == numerico: Exp1'.t = numerico;
senao Exp1'.t = **tipo_erro** }
| ε {Exp1'.t = vazio} 53

Exp2 → Exp3 Exp2'
{se Exp2'.t == vazio: Exp2.t = Exp3.t;
senao se Exp2'.t == Exp3.t && Exp2'.t == numerico: Exp2.t = numerico;
senao: Exp2.t = **tipo_erro**} 54

Exp2' → "+" Exp3 Exp2' 55 | "-" Exp3 Exp2' 56
// ação semantica vale para as regras 55 e 56
{se Exp2'Filho.t == vazio && Exp3.t == numerico: Exp2'.t = numerico;
senao se Exp2'Filho.t == Exp3.t && Exp3.t == numerico: Exp2'.t = numerico;
senao Exp3'.t = **tipo_erro**}
| ε {Exp2'.t = vazio} 57

Exp3 → Exp4 Exp3'
{se Exp3'.t == vazio: Exp3.t = Exp4.t;
senao se Exp3'.t == Exp4.t && Exp3'.t == numerico: Exp3.t = numerico;
senao: Exp3.t = **tipo_erro**} 58

Exp3' → "*" Exp4 Exp3' 59 | "/" Exp4 Exp3' 60
// ação semantica vale para as regras 59 e 60
{se Exp3'Filho.t == vazio && Exp4.t == numerico: Exp3'.t = numerico
senao se Exp3'Filho.t == Exp4.t && Exp4.t == numerico: Exp3'.t = numerico;
senao Exp3'.t = **tipo_erro** }
| ε {Exp3'.t = vazio} 61

Exp4 → ID Exp4' {Exp4.t = TS.getTipo(ID.lexval);
se Exp4.t == null:
Exp4.t = **tipo_erro**; **sinalizar erro para ID não declarado**} 62
| ConstInteger {Exp4.t = numerico} 63 | ConstDouble {Exp4.t = numerico} 64
| ConstString {Exp4.t = string} 65
| "true" {Exp4.t = logico} 66 | "false" {Exp4.t = logico} 67
| OpUnario Exp4
{se Exp4Filho.t == OpUnario.t && OpUnario.t == numerico: Exp4.t = numerico;
senão se Exp4Filho.t == OpUnario.t && OpUnario.t == logico: Exp4.t = logico;
senão Exp4.t = **tipo_erro**} 68
| "(" Expressao")" {Exp4.t = Expressao.t} 69

Exp4' → "(" RegexExp ")" 70 | ε 71

OpUnario → "-" {OpUnario.t = numerico} 72 | "!" {OpUnario.t = logico} 73