Universidad de San Carlos de Guatemala
Centro Universitario de Occidente División de Ciencias de la Ingeniería
Compiladores 2
Ing. Jose Granados Guevara
Sección: A


# Manual Técnico

Carlos Raúl Alberto López Peláez 202031871

Quetzaltenango,10 de Septiembre del 2024

# Clase Abstracta: Instrucción

La clase Instruccion es una clase abstracta que sirve como base para todas las instrucciones del lenguaje.

**Atributos:**

- tipo: Representa el tipo de dato de la instrucción.
- linea: Indica la línea donde se encuentra la instrucción en el código fuente.
- columna: Indica la columna donde comienza la instrucción en el código fuente.

**Métodos:**

- interpretar(Arbol arbol, TablaSimbolos tabla): Método abstracto que debe ser implementado por las clases hijas para interpretar la instrucción.

# Clase: Arbol

La clase Arbol representa el árbol sintáctico abstracto (AST) del programa.

**Atributos:**

- instrucciones: Lista enlazada de instrucciones del programa.
- consola: String para almacenar la salida del programa.
- tablaGlobal: Tabla de símbolos global.
- errores: Lista de errores encontrados durante la interpretación.
- funciones: Lista de funciones definidas en el programa.

**Métodos principales:**

- getInstrucciones(), setInstrucciones(): Métodos para obtener y establecer la lista de instrucciones.
- getConsola(), setConsola(): Métodos para obtener y establecer la salida del programa.
- getTablaGlobal(), setTablaGlobal(): Métodos para obtener y establecer la tabla de símbolos global.
- getErrores(), setErrores(): Métodos para obtener y establecer la lista de errores.
- Print(String valor): Añade texto a la consola.
- addFunciones(Instruccion funcion): Agrega una función a la lista de funciones.
- getFuncion(String id): Busca y retorna una función por su identificador.

# Clase: Simbolo

La clase Simbolo representa un símbolo en la tabla de símbolos.

**Atributos:**

- tipo: Tipo de dato del símbolo.
- tipoDeclaracion: Tipo de declaración (por ejemplo, variable, constante).
- id: Identificador del símbolo.
- valor: Valor asociado al símbolo.
- linea y columna: Posición en el código fuente.
- mutabilidad: Indica si el símbolo puede ser modificado.
- entorno: Entorno en el que se declara el símbolo.

**Métodos:**

- Métodos get y set para acceder y modificar los atributos.

# Clase: TablaSimbolos

La clase TablaSimbolos implementa la tabla de símbolos del intérprete.

**Atributos:**

- tablasTotales: Lista de todas las tablas de símbolos.
- tablaAnt: Referencia a la tabla de símbolos del ámbito superior.
- tablaAct: HashMap que almacena los símbolos del ámbito actual.
- nombre: Nombre de la tabla de símbolos.
-

**Métodos principales:**

- getTablaAnt(), setTablaAnt(): Métodos para obtener y establecer la tabla de símbolos padre.
- getTablaAct(), setTablaAct(): Métodos para obtener y establecer la tabla de símbolos actual.
- getNombre(), setNombre(): Métodos para obtener y establecer el nombre de la tabla de símbolos.
- setVariable(Simbolo simbolo): Añade un símbolo a la tabla actual.
- getVariable(String ID): Busca un símbolo en la tabla actual y en las tablas superiores.
- setVariablesVector(Simbolo[] simbolo): Añade un array de símbolos a la tabla.
- getVariableVector(String ID, int index): Obtiene un símbolo de un array.

# Clase: Tipo

La clase Tipo encapsula el tipo de dato de un símbolo.

**Atributo:**

- tipo: Enumeración TipoDato que representa el tipo de dato.

**Métodos:**

- getTipo(), setTipo(): Métodos para obtener y establecer el tipo de dato.

# Enumeración: TipoDato

La enumeración TipoDato define los tipos de datos soportados por el intérprete:

- ENTERO
- DECIMAL
- BOOLEANO
- CARACTER
- CADENA
- VOID

Estas clases forman la estructura básica del intérprete, permitiendo la representación y manipulación de los

# Gramática

Gramática general de la estructura del lenguaje pascal

1. Inicio

```
ini ::=
     PROGRAM ID CIERRE declaraciones:a  BEGIN instrucciones:c END PUNTO
;
```

Esta regla define la estructura básica de un programa Pascal, que comienza con la palabra clave `PROGRAM`, seguida de un identificador, declaraciones, y un bloque principal de instrucciones entre `BEGIN` y `END`.

2. Declaraciones

```
declaraciones ::=
    declaracion:a {:if (a!=null){RESULT = a;} :}
   |declaracion:a declaraciones:b {: RESULT = a; if(b!=null ){ a.addAll(b); RESULT = a;} :}
;

declaracion ::=
    declaracionconst:a  {: RESULT = a;  System.out.println(Arrays.toString(a.toArray()));:}
   |declaraciontype:a    {: :}
   |declaracionvar:a     {: RESULT = a;  System.out.println(Arrays.toString(a.toArray()));:}
   |procedure:b          {: RESULT = b; :}
;
```

Define los diferentes tipos de declaraciones posibles: constantes, tipos, variables y procedimientos.

- Procedure

```
procedure ::=
    PROCEDURE listaprocedure:a {: RESULT = a; :}
;

listaprocedure ::=
   declaracionprocedure:a CIERRE listaprocedure:b {: if(b!= null){a.addAll(b); } RESULT =a;:}
   |/* empty */
;

declaracionprocedure::=
    ID:b {:System.out.println(b);:} CIERRE varprocedure:a BEGIN instruccionesincierre:c END
   |ID:b PARA params:c PARC CIERRE varprocedure:a BEGIN instruccionesincierre:d END  {:if (a!
;

varprocedure::=
       declaracionvar:a    {: RESULT = a; :}
      | /* empty */
;
```

- Const

```
;

declaracionconst ::=
    CONST {:tipoD = "const";:}listadeclaraciontipo:a
;

listadeclaraciontipo ::=
    declaraciontipo:a CIERRE listadeclaraciontipo:b
    | /* empty */
;

declaraciontipo ::=
    ID:a listaids:b DOBP tipo:c IGUAL expresion:d {: R
   |ID:a listaids:b IGUAL expresion:c  {: RESULT =new
;
```

- Type

```
declaraciontype ::=
    TYPE {:tipoD = "type"; System.out.println("Type");:}asignartipos:a  {::}
;

asignartipos ::=
    listadeclaraciontype:a CIERRE asignartipos:b {: :}
   | /* empty */
;

listadeclaraciontype::=
     ID:a COMA {:System.out.println(" ID , listadeclaracion");:}listadeclaraciontype:b {: :}
    |ID:a {:System.out.println(" ID 2 " + a);:}  IGUAL  tipo:b  {::}

;
```

- Var

```
declaracionvar ::=
    VAR {:tipoD = "var";:}listadeclaracionvar:a  {: RESULT = a; :}
;

listadeclaracionvar ::=
      variables:a  CIERRE listadeclaracionvar:b      {:  if(b!= null){a.addAll(b); } RES
     | /* empty */
;

variables ::=
    ID:a listaids:b DOBP tipo:c IGUAL expresion:d  {: RESULT = new LinkedList<>(Arrays.a
   |ID:a listaids:b IGUAL expresion:c  {: RESULT = new LinkedList<>(Arrays.asList(  new
   |ID:a listaids:b DOBP tipo:c {: RESULT =new LinkedList<>(Arrays.asList( new Declaraci
   |ID:a listaids:b DOBP ARRAY CORA ENTERO:c RANGO ENTERO:d CORC OF tipo:e CORC CIERRE {
;
```

```
listaids ::=
    COMA ID:a listaids:b
  | /* empty */
;
```

3. Instrucciones

```
instrucciones ::=
   instrucciones:a instruccion:b CIERRE {: RESULT = a; RESULT.add(b); :}
 | instruccion:a  CIERRE{: RESULT = new LinkedList<>(); RESULT.add(a); :}
 | error CIERRE

;
```

Define una lista de instrucciones. Puede ser una instrucción seguida de más instrucciones, una sola instrucción, o manejar un error sintáctico.

4. Expresiones

```
expresion ::= RESTA expresion:a                {:{
            | expresion:a POTENCIA expresion:b
            | expresion:a MOD expresion:b    {:
            | expresion:a SUMA expresion:b    {:
            | expresion:a RESTA expresion:b    {
            | expresion:a MULTI expresion:b    {
            | expresion:a DIV expresion:b    {:
            | expresion:a IGUALIGUAL expresion:
            | expresion:a DIFIGUAL expresion:b
            | expresion:a MENORQ expresion:b
            | expresion:a MENORIGQ expresion:b
            | expresion:a MAYORQ expresion:b
            | expresion:a MAYORIGQ expresion:b
            | expresion:a OR expresion:b    {:
            | expresion:a AND expresion:b    {:
            | NOT expresion:a    {:    RESULT = n
            | PARA expresion:a PARC  {:    RESUL
            | ENTERO:a       {:    RESULT = new N
            | DECIMAL:a      {:    RESULT = new N
            | STRING_LITERAL:a
            | CHAR_LITERAL:a
            | booleanos:a       {
            | ID:a           {:
;
```

5. Estructuras de Control

```
cfor ::=
    FOR ID:a DOBP IGUAL expresion:b TO expresion:c DO BEGIN instruccionesincierre:d END
;

swhile ::=
    WHILE expresion:a DO BEGIN instruccionesincierre:b  END  {: RESULT = new WHILE(a,b,
;

srepeat ::=
    REPEAT instruccionesincierre:b UNTIL expresion:a     {: RESULT = new REPEAT(a,b, ale
;

scase ::=
    CASE expresion:a OF casos:b casodefault:c END
   |CASE expresion:a OF casos:b END  {: RESULT = r
   |CASE expresion:a OF casodefault:b  END {: RESU
;

casos ::=
       casos:a caso:b {: RESULT = a; RESULT.add(b)
    | caso:a  {: RESULT = new LinkedList<>(); RESU
;


caso ::=
    listacasos:a DOBP subcasos:b {:   for (Instruc
;

listacasos ::=
    expresion:a COMA  listacasos:b {: if (b != nul
   |expresion:a {: RESULT = new LinkedList<>(Array
;

subcasos::=
      instruccion:b CIERRE {: RESULT = new LinkedL
    | BEGIN instruccionesincierre:b END CIERRE {:
;

casodefault ::=
    ELSE  instruccionesincierre:a   {:  RESULT = r
;
```