**Ingeniera en Sistemas de** **información**

Sintaxis y semántica de los lenguajes

**Proyecto final - Interprete de Lenguaje**

**Docentes :**

Ing. Andrés Pascal

Ing. Claudia Alvarez

**Alumnos**

Carolina Bidal

Nicolas Bonnin

Nahuel Montesino

Agustin Raffo

**Fecha de Entrega:**

**Índice**

[**1.Características del lenguaje**](#_ea0hedf2ehro) **3**

[**2. Gramática BNF Asociada**](#_2htz3canqpgs) **3**

[2.1 Gramática en forma LL1](#_t0opxznp373d) 4

[**3. Componentes léxicos**](#_xzbv4lafheev) **4**

[3.1 Semántica asociada a cada variable](#_m2h9pkojaoz0) 5

[**4. Descripción del programa**](#_w6xj4mm9sy5m) **14**

[4.1 Método de uso](#_af09qxd1v984) 14

[4.3 Ejemplos de la sintaxis del lenguaje](#_z4xfvca0678i) 15

[4.4 Programas pedidos por la cátedra](#_8k7xefum8578) 16

[**5. Consideraciones a tener en cuenta**](#_5rqatsps1ifh) **17**

# **1.Características del lenguaje**

1. Un programa es una secuencia de sentencias.
2. Una sentencia es una asignación, una lectura, una escritura, un Si-Entonces-Sino o un ciclo Mientras.
3. El lado derecho de una asignación es una expresión aritmética sobre números reales, incluyendo suma, resta, producto, division, potencia y raíz.
4. Las variables no se declaran.
5. Una lectura contiene una cadena que se mostrará por pantalla y la variable a leer.
6. Una escritura contiene una cadena que se mostrará por pantalla y una expresión aritmética.
7. Las condiciones del “Mientras” y el “Sí” deben permitir operadores lógicos (además de los relacionales).
8. Se permiten hacer modificaciones o agregados a esta descripción, siempre que tengan su justificación.

# **2. Gramática BNF Asociada**

<PROG> ::= <SENT> “;” <PROG> | <SENT>;

<SENT> ::= <ASIGNACION> | <LEER> | <ESCRIBIR> | <CONDICIONAL> | <CICLO>

<CONDICIONAL> ::= “IF” <CONDICION> “THEN” <PROG> “END” |

“IF” <CONDICION> “THEN” <PROG> “ELSE” <PROG> ”END”

<CICLO> ::= “WHILE” <CONDICION> “DO” <PROG> “END”

<CONDICION> ::= <CONDAND> “OR” <CONDICION> | <CONDAND>

<CONDADN> ::= <CONDNOT> “AND” <CONDAND> | <CONDNOT>

<CONDNOT> ::= “NOT” <CONDNOT> | <EXPRESION> “oprel” <EXPRESION> |

“[” <CONDICION> “]”

<ASIGNACION> ::= “id” “=” <EXPRESION>

<EXPRESION> ::= <TERMINO> “+” <EXPRESION> | <TERMINO> “-” <EXPRESION> |<TERMINO>

<TERMINO> ::= <FACTOR> “\*” <TERMINO> | <FACTOR> “/ <TERMINO> | <FACTOR>

<FACTOR> ::= “id” | “num” | “(” <EXPRESION> “)”

<LEER> ::= "READ" “(” “STRING” “,” “id” “)”

<ESCRIBIR> ::= “WRITE” “(” “STRING” “,” <EXPRESION> “)”

## **2.1 Gramática en forma LL1**

<PROG> ::= <SENT> “;” <PROG> | e

<SENT> ::= <ASIGNACION> | <LEER> | <ESCRIBIR> | <CONDICIONAL> | <CICLO>

<CONDICIONAL> ::= “IF” <CONDICION> “THEN” <PROG> <SINO> ”END”

<SINO> ::= ”ELSE” <PROG> | e

<CICLO> ::= “WHILE” <CONDICION> “DO” <PROG> “END”

<CONDICION> ::= <CONDAND> <T>

<T> ::= “OR” <CONDICION> | e

<CONDAND> ::= <CONDNOT> <S>

<S> ::= “AND” <CONDAND> | e

<CONDNOT> ::= “NOT” <CONDNOT> | <EXPRESION> “oprel” <EXPRESION> | “[” <CONDICION> “]”

<ASIGNACION> ::= “id” “=” <EXPRESION>

<EXPRESION> ::= <TERMINO> <DTERMINO>

<DTERMINO> ::= “+” <EXPRESION> | “-” <EXPRESION> | e

<TERMINO> ::= <FACTOR> <DFACTOR>

<DFACTOR> ::= “\*” <TERMINO> | “/” <TERMINO> | e

<FACTOR> ::= “id” | “num” | “(” <EXPRESION> “)”

<LEER> ::= “READ” “(” “STRING” “,” “id” “)”

<ESCRIBIR> ::= “WRITE” “(” “STRING” “,” <EXPRESION> “)”

# **3. Componentes léxicos**

Palabras reservadas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IF | THEN | END | ELSE | WHILE |
| WRITE | DO | NOT | AND | OR |
| READ |  |  |  |  |

Constantes numéricas (números reales)

* num

Constantes de carácter (reconocidas entre “ ”)

* string

Operadores relacionales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| < | > | == | != | = |

Operadores aritméticos

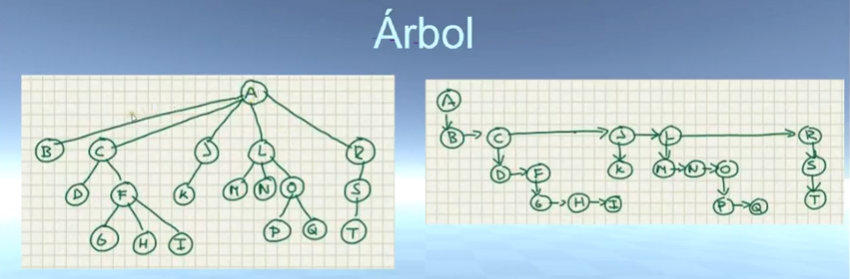
|  |  |  |  |
| --- | --- | --- | --- |
| + | - | \* | / |

Símbolos de puntuación

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ; | [ | ] | ( | ) | , |

## **3.1 Semántica asociada a cada variable**

La semántica se realizará siguiendo la siguiente estructura de árbol situada a la derecha en la siguiente imagen.

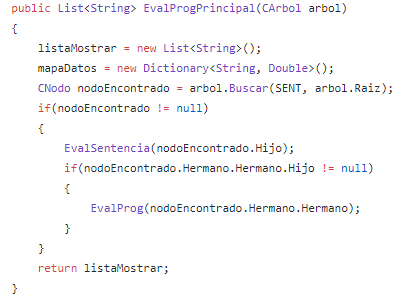
****

En la cual observamos que cada nodo tendrá un solo hijo y si queremos agregar mas hijo a este, serán hermanos del primer hijo.

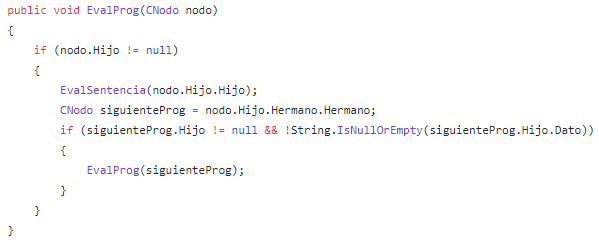
Los nodos del árbol tendrán un atributo llamado dato, en el cual se guardarán los lexemas y en caso de ser un número o string, su valor correspondiente.

En nuestro evaluador utilizaremos un diccionario(clave valor), para ir guardando los diferentes estados de los identificadores a lo largo de la evaluación.

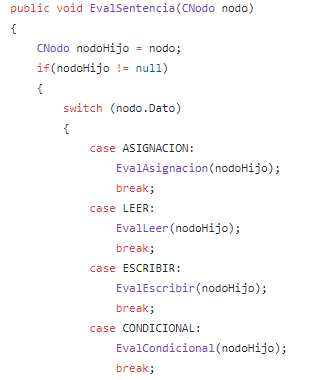
1) <PROG> ::= <SENT> “;” <PROG> | e

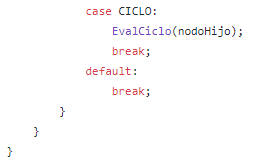


Evalúa los PROG siguientes



2) <SENT> ::= <ASIGNACION> | <LEER> | <ESCRIBIR> | <CONDICIONAL> | <CICLO>

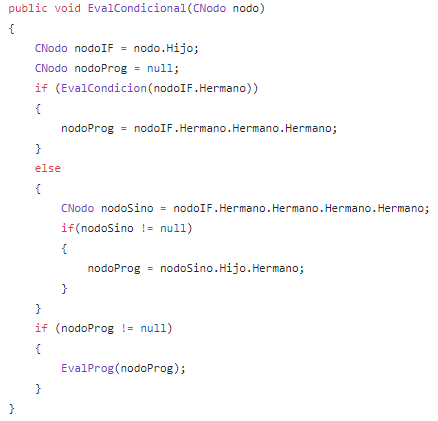




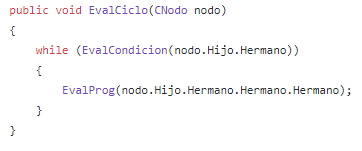
·

3) <CONDICIONAL> ::= “IF” <CONDICION> “THEN” <PROG> <SINO> ”END”

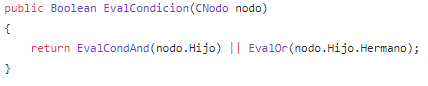
<SINO> ::= ”ELSE” <PROG> | e



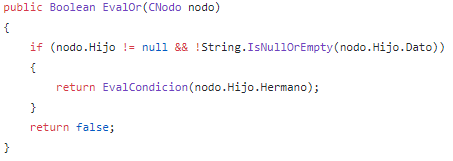
4) <CICLO> ::= “WHILE” <CONDICION> “DO” <PROG> “END”



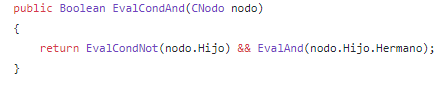
5) <CONDICION> ::= <CONDAND> <T>



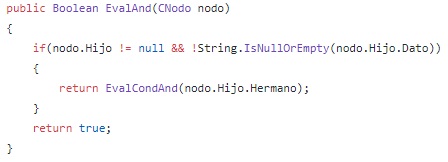
6) <T> ::= “OR” <CONDICION> | e



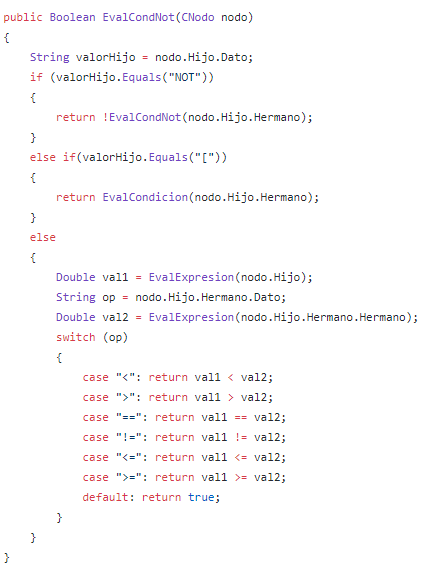
7) <CONDAND> ::= <CONDNOT> <S>



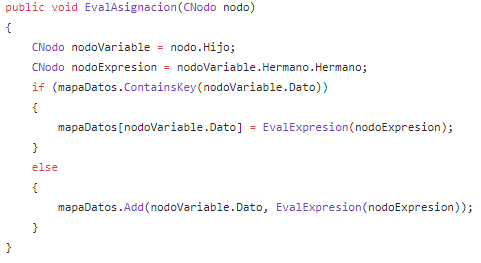
8) <S> ::= “AND” <CONDAND> | e



9) <CONDNOT> ::= “NOT” <CONDNOT> | <EXPRESION> “oprel” <EXPRESION> | “[” <CONDICION> “]”

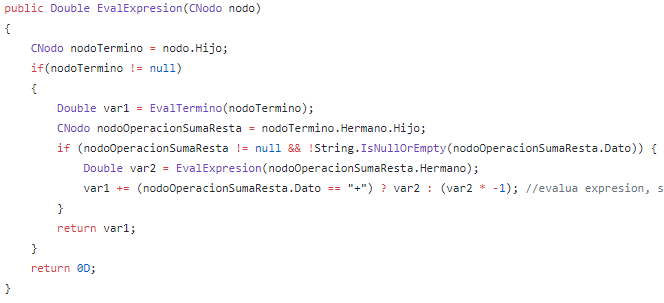


10 ) <ASIGNACION> ::= “id” “=” <EXPRESION>



11)<EXPRESION> ::= <TERMINO> <DTERMINO>

<DTERMINO> ::= “+” <EXPRESION> | “-” <EXPRESION> | e

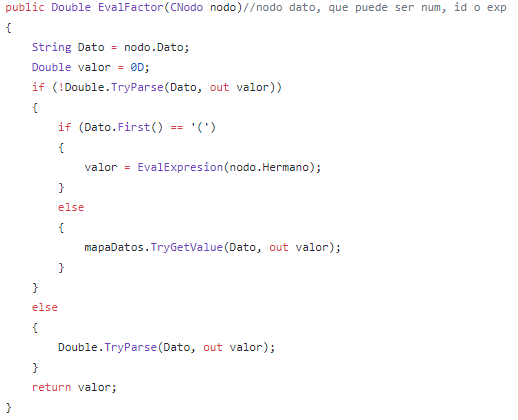


12) <TERMINO> ::= <FACTOR> <DFACTOR>

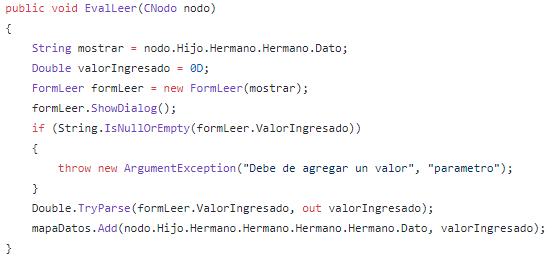


13) <DFACTOR> ::= “\*” <TERMINO> | “/” <TERMINO> | e

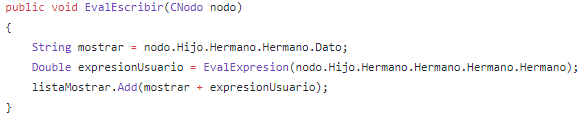
<FACTOR> ::= “id” | “num” | “(” <EXPRESION> “)”



14) <LEER> ::= “READ” “(” “STRING” “,” “id” “)”



15) <ESCRIBIR> ::= “WRITE” “(” “STRING” “,” <EXPRESION> “)”



# **4. Descripción del programa**

El intérprete permite ingresar (por medio de un textbox) un programa fuente, realizar el analisis lexico y sintactico de este para luego ejecutarlo en el evaluador.

Para el análisis léxico se utiliza una MDD (máquina discriminadora discreta), que es la encargada de reconocer los componentes léxicos.

Para el análisis sintáctico se utiliza un analizador sintáctico descendente predictivo no recursivos para gramáticas ll1. Esté utilizando las TAS(la cual se encuentra cargada con anterioridad en el programa) y una PILA irá armando el árbol de derivaciones correspondiente al programa ingresado.

El evaluador toma el árbol obtenido anteriormente y comienza a evaluar el mismo partiendo de la raíz hacia los hijos de manera recursiva, de manera que el programa se vaya ejecutando linea a linea.

## **4.1 Método de uso**

1. Se debe ingresar el texto por pantalla o elegir algún programa de ejemplo que se encuentra precargado en los botones (para obtener más información de la sintaxis puede pulsar el botón “Info”)
2. Luego se debe presionar el botón “Reconocer elementos” para realizar el el analisis lexico.
3. Con el botón “Validar Sintaxis” realizamos el análisis sintáctico.
4. Para finalizar presionamos el botón “Evaluar”, que nos permite ejecutar el programa, si la fases anteriores se realizaron correctamente.

## **4.3 Ejemplos de la sintaxis del lenguaje**

**Asignación**

variable = datoNumerico;

Operaciones

+ - / \*

*Ejemplo:* 1 + 1;

*Separadores entre operaciones*: ( )

**Relacionales**

< > == !=

**Obtener dato por usuario**

READ("Texto a mostrar", variable);

**Mostrar dato por pantalla**

WRITE("Texto a mostrar", variable);

**Condicional**

IF (condicion) THEN codigo a ejecutar SINO codigo a ejecutar END;

**Ciclo**

WHILE (condicion) DO codigo a ejecutar END;

**Condición**

***Negar expresión****:* NOT

***Agregar más condiciones****:* AND OR

***Separadores entre condiciones***: [ ]

## **4.4 Programas pedidos por la cátedra**

**1-** Escribir un programa en este lenguaje que calcule la sumatoria de los primeros N números (N se ingresa por pantalla).

**READ("ingrese un numero", n);**

**IF n < 0 THEN**

**n = n \* (-1);**

**x = 1;**

**ELSE**

**x = 0;**

**END;**

**c = 1;**

**a = 0;**

**WHILE c < n DO**

**a = a + c;**

**c = c + 1;**

**END;**

**a = a + c;**

**IF x == 1 THEN**

**a = a \* (-1);**

**END;**

**WRITE("Sumatoria Resultado" , a);**

**2-** Escribir un programa en este lenguaje que, dados dos puntos (se ingresan las coordenadas X e Y de cada punto), calcule su distancia euclidiana.

**READ("ingrese el valor x1", x1);**

**READ("ingrese el valor y1", y1);**

**READ("ingrese el valor x2", x2);**

**READ("ingrese el valor y2", y2);**

**a1 = x2 - x1;**

**a2 = y2 - y1;**

**a1 = a1 \* a1;**

**a2 = a2 \* a2;**

**a = a1 + a2;**

**c = 1;**

**e = 0;**

**c1 = a - (c \* c);**

**c2 = 0;**

**IF (c + 1)\*(c + 1) > a THEN**

**c2 = ( ( c + 1) \* (c + 1) ) - a;**

**ELSE**

**c2 = a -( (c + 1) \* (c + 1) );**

**END;**

**WHILE c1 > c2 DO**

**e = (c + 1)\*(c +1);**

**c = c + 1;**

**c1 = a - (c \* c);**

**IF (c + 1)\*(c + 1) > a THEN**

**c2 = ( (c + 1) \* (c + 1) ) - a;**

**ELSE**

**c2 = a -( (c + 1) \* (c + 1) );**

**END;**

**END;**

**r = (a + e) / ( c \* 2 );**

**WRITE("Norma Euclidiana" , r);**

Para hacer la raíz cuadrada usa el metodo babilonico.

# **5. Consideraciones a tener en cuenta**

* Las palabras reservadas deberán estar en Mayúscula.
* En la ventana ingresar dato, no se podrá salir a menos que ingrese un dato correctamente.
* Reconoce constantes reales.
* Las strings están separadas entre “ “ sí no es identificador.
* Los identificadores comienzan con una letra y son seguidos por letras y números.
* Se debe prestar atención a la hora de ingresar datos, a la hora de ingresar algún valor numérico(ej: en la sumatoria de los primeros N número , se debe ingresar un valor entero, ya que se reconocen números reales)
* No acepta notación científica para números reales (es decir deberán cargarse a mano)
* Los números negativos no deben tener espacio entre el signo y el número

(-5), si no se considera como operación resta.