MANUAL DE USUARIO

MANUAL DEL SOFTWARE

Contenido

Versión 1.0..... **¡Error! Marcador no definido.**
1. INTRODUCCIÓN 4
2. OBJETIVO DE ESTE MANUAL 4
3. DIRIGIDO..... 4
4. CONOCIMIENTOS PREVIOS 4
4.1. PASOS PARA EMPEZAR A UTILIZAR EL PROGRAMA 4
5. DESCRIPCIÓN DEL PROGRAMA 5
6. EDICION. 6
7. ARCHIVO..... 6
8. CONTENIDO DEL SUBMENU ARCHIVO 6
9. CONTENIDO DEL SUBMENU HERRAMIENTAS 7
11. CONTENIDO DEL SUBMENU REPORTES 8
12. CONSOLA DE SALIDA 8
13. DEFINICIÓN..... 8
14. DESCRIPCION DEL LENGUAJE PERMITIDO..... 9
15. REPORTES:..... 16

TABLA DE ILUSTRACIONES

1. **Ilustración 1: INTERFAZ DE USUARIO……5**
2. **Ilustración 2 EDICION (MENU)……..6**
3. **Ilustración 3 ARCHIVO (CONTENIDO)……8**
4. **Ilustración 4 CONTENIDO DE HERRAMIENTAS (MENU)……..9**
5. **Ilustración 5 BOTON ANALIZAR ……..9**
6. **Ilustración 6 CONTENIDO DE HERRAMIENTAS (MENU)…….10**
7. **Ilustración 7 CONSOLA DE SALIDA FUNCIONAN …….10**
8. **Ilustración 9 TIPOS DE DATOS PERMITIDOS EJEMPLO……..14**
9. **Ilustración 10 TIPOS DE CARACTERES ESPECIALES EJEMPLO……...14**
10. **Ilustración 11 PRECEDENCIA DE OPERACIONES EJEMPLO……..14**

1. INTRODUCCIÓN

Interprete que ejecuta instrucciones de alto nivel, definidas en el lenguaje exclusivo para la USAC, en el cual será llamado JPR, programación acorde a la reforma curricular, donde podrán ver las funcionalidades principales y la aplicación de los fundamentos de programación.

2. OBJETIVO DE ESTE MANUAL

El objetivo primordial de este Manual es ayudar y guiar al técnico a informarse y utilizar el programa, explicando sus funcionalidades.

- Conocer cómo utilizar el sistema, mediante una descripción detallada e ilustrada de las opciones.
- Conocer el alcance de toda la información por medio de una explicación detallada e ilustrada de cada una de las páginas que lo conforman el manual técnico.

3. DIRIGIDO

Este manual está orientado a los técnicos u otros tipos de personal encargado del Departamento de Ingeniería del área de ciencias y sistemas. Solamente dichas personas están autorizadas a realizar modificaciones en el sistema.
También a través de este manual el personal podrá estar en la capacidad de supervisar el cumplimiento de políticas, normas, etc. que permitan el correcto funcionamiento del programa.

4. CONOCIMIENTOS RECOMENDADOS

Los conocimientos mínimos que deben tener las personas que operarán las páginas y deberán utilizar este manual son:

- Conocimientos básicos de programación en alto nivel
- Conocimientos básicos de un IDE
- Conocimiento básico de Windows

4.1. PASOS PARA EMPEZAR A UTILIZAR EL PROGRAMA

El usuario tendrá la libertad de programar directamente en el programa o abrir un archivo que este en las condiciones permitidas.

Ilustración 1: INTERFAZ DE USUARIO

5. DESCRIPCIÓN DEL PROGRAMA

El programa contiene un editor del lado izquierdo, color blanco en donde podrá trabajar, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad para el usuario. La función principal del editor será el ingreso de código fuente que será analizado. En este se podrán abrir archivos .jpr y deberá mostrar la línea actual.

Menú Lateral

- ✓ Archivo
- ✓ Edicion
- ✓ Herramientas
- ✓ Analizar
- ✓ Reportes
- ✓ Ayuda
- ✓ Salir

6. **EDICION.**

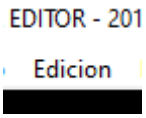

Ilustración 2 EDICION (MENU)

7. **ARCHIVO**

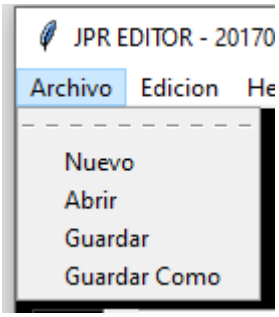

Ilustración 3 ARCHIVO (CONTENIDO)

8. **CONTENIDO DEL SUBMENU ARCHIVO**

El sistema ofrece un menú de archivo donde podrá realizar las siguientes acciones.

- Nuevo: Limpia el editor y pregunta si quiere guardar el actual.
- Abrir: Abre un buscador para seleccionar un archivo existente y colocarlo en el editor
- Guardar: Guarda el archivo actual o si es nuevo se guarda como nuevo.
- Guardar como: Se guarda el contenido del editor en un archivo nuevo.

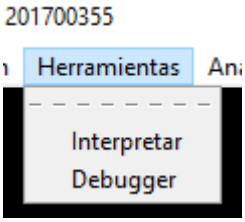

Ilustración 4 CONTENIDO DE HERRAMIENTAS (MENU)

9. CONTENIDO DEL SUBMENU HERRAMIENTAS

El sistema ofrece un menú de herramientas donde podrá realizar las siguientes acciones.

- Interpretar: Ejecuta el editor para proceder al analisis
- Debugger: Funcion que permite avanzar función por función en el análisis.

10. ANALIZAR:

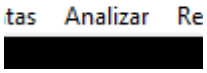

Ilustración 5 BOTON ANALIZAR

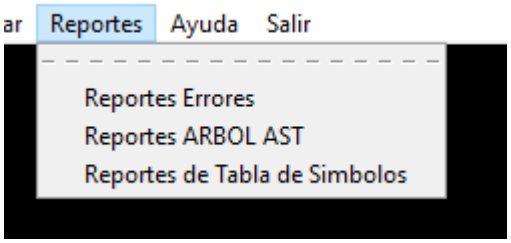

Ilustración 6 CONTENIDO DE HERRAMIENTAS (MENU)

11. CONTENIDO DEL SUBMENU REPORTES

El sistema ofrece un menú de reportes donde podrá realizar las siguientes acciones.

- Reportes Erroes: Ejectura una ventana donde podra escoger donde guardar su reporte de errores
- Reporte ARBOL AST: Genera un archivo pdf donde podra ver el ARBOL generado

12. CONSOLA DE SALIDA

13.1 DEFINICIÓN

La aplicación ofrece una consola de salida en donde mostrara las salidas generadas por la función del editor, así mismo con los errores léxicos, sintácticos y semánticos.

Ilustración 7 CONSOLA DE SALIDA FUNCIONAN

12.1. DESCRIPCION DEL LENGUAJE PERMITIDO

1. CASE SENSITIVE: El lenguaje permitido en el programa no distinguirá entre mayúsculas o minúsculas
2. COMENTARIOS: Los comentarios son una forma elegante de indicar que función tiene cierta sección del código que se ha escrito simplemente para dejar algún mensaje en específico. El lenguaje deberá soportar dos tipos de comentarios que son los siguientes:
    - COMENTARIOS DE UNA LINEA: Estos comentarios deberán comenzar con # y terminar con un salto de línea
3. TIPOS DE DATOS PERMITIDOS: Los tipos de dato que soportará el lenguaje en concepto de un tipo de variable se definen en la ilustración 11
4. CARACTERES ESPECIALES: Las secuencias de escape se utilizan para definir ciertos caracteres especiales dentro de cadenas de texto. Las secuencias de escape disponibles en la ilustración 12.
5. OPERADORES ARITMETICOS PERMITIODOS:
    - SUMA (+)
    - RESTA (-)
    - DIVISION (/)
    - MULTIPLICACION (*)
    - POTENCIA (**)
    - MODULO (%)
    - NEGACION UNARIA (-1)
6. OPERADORES RELACIONALES PERMITIDOS:
    - IGUALACION (==)
    - DIFERENCIACION (=!)
    - MENOR QUE (<)
    - MAYOR QUE (>)
    - MENOR IGUAL (<=)
    - MAYOR IGUAL (>=)
7. OPERADORES LOGICOS PERMITIDOS
    - OR (||)
    - AND (&&)
    - NOT (¡)
8. SIGNOS DE AGRUPACION: Los signos de agrupación serán utilizados para agrupar operaciones aritméticas, lógicas o relacionales. Los símbolos de agrupación están dados por paréntesis ( ).
9. PRECEDENCIA DE OPERACIODERS: La precedencia de operadores nos indica la importancia en que una operación debe realizarse por encima del resto. Disponible en la ilustración 13.
10. CARACTERES DE INALICACION Y ENCAPSULAMIENTO DE SENTENCIAS.
    - Finalización de instrucciones: para finalizar una instrucción puede o no venir el signo punto y coma (;).
    - Encapsular sentencias: para encapsular sentencias dadas por los ciclos, métodos, funciones, etc, se utilizará los signos { y }
11. DECLARACION DE VARIABLES: Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador. Las variables podrán ser declaradas global y localmente. La declaración de variables debe de tener la palabra “var”, seguido de un identificador y su expresión.

    #Ejemplos

    var numero; # null

    var cadena = “hola” #String

    var var_1 = ‘a’; # char

    var verdadero; # null
12. CASTEOS: Los casteos son una forma de indicar al lenguaje que convierta un tipo de dato en otro, por lo que, si queremos cambiar un valor a otro tipo, es la forma adecuada de hacerlo. Para hacer esto, se colocará la palabra reservada del tipo de dato destino entre paréntesis seguido de una expresión.

    El lenguaje aceptará los siguientes casteos:

    • Int a double • Double a Int

• Int a String
• Int a Char
• Double a String
• Char a int
• Char a double
• String a Int
• String double
• String a Boolean

#Ejemplos

```
var edad = (int) 18.6; #toma el valor entero de 18
var letra = (char) 70 #tomar el valor 'F' ya que el 70 en ascii es F
var numero = (double) 16; #toma el valor 16.0
```

13. INCREMENTO Y DECREMENTO: Los incrementos y decrementos nos ayudan a realizar la suma o resta continua de una variable de uno en uno, es decir si incrementamos una variable, se incrementará de uno en uno, mientras que, si realizamos un decremento, hará la operación contraria. Solamente realizará operaciones con tipo de dato Int y Double

#EJEMPLOS

```
var edad = 18.5;
edad++; #tiene el valor de 19.5
edad--; #tiene el valor 18.5
```

14. ARREGLOS: Los arreglos son una estructura de datos de tamaño fijo que pueden almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo; int, double, boolean, char o string. El lenguaje permite el uso de arreglos de múltiples dimensiones
15. DECLARACION DE ARREGLOS: Al momento de declarar un arreglo, tenemos dos tipos que son:

• Declaración tipo 1: En esta declaración, se indica por medio de una expresión numérica del tamaño y la dimensión que se desea el arreglo, tomando el valor null para cada espacio. Sin embargo, estos valores no podrán cambiar de tipo.

• Declaración tipo 2: En esta declaración, se indica por medio de una lista de valores separados por coma, los valores que tendrá el arreglo, en este caso el tamaño del arreglo será el de la misma cantidad de valores de la lista. Si es de más dimensiones el arreglo, la lista deberá de contener las listas necesarias para la asignación de todos los elementos del arreglo.

#Ejemplo de declaración tipo 1

```
int[ ] arr1 = new int[4]; #se crea un arreglo de 4 posiciones, con null en cada posición
int[ ][ ] matriz1 = new int[2][2]; #se crea un arreglo de 4 posiciones, con null en cada posición
int[ ][ ] matriz2 = matriz1; # la referencia de matriz1 está en matriz2
```

#Ejemplo de declaración tipo 2

```
string[ ] arreglo2 = {"hola", "Mundo"}; #arreglo de 2 posiciones, con "Hola" y "Mundo"
int[ ][ ] matriz = { {11, 12} , {21,22} }; #arreglo de 2 dimensiones, con 2 posiciones en cada dimensión.
char[ ][ ][ ] cubo = { { {'a', 'b'} , {'c','d'} } , { {'e', 'f'} , {'g','h'} } , { {'i', 'j'} , {'k','l'} } }
 #arreglo de 3 dimensiones de 2 dimensiones, con 2 posiciones en cada dimensión. ([3][2][2])
```

16. ACCESO A LOS ARREGLOS: Para acceder al valor de una posición de un arreglo, se colocará el nombre del arreglo seguido de '[' EXPRESION ']'.

#Ejemplo de acceso

```
string[ ] arr2 = {"hola", "Mundo"}; #creamos un arreglo de 2 posiciones de tipo string
```

17. MODIFICACION DE ARREGLOS: Para modificar el valor de una posición de un arreglo, se debe colocar el nombre del arreglo seguido de '[' EXPRESION ']' = EXPRESION

18. SENTENCIAS DE CONTROL: Estas sentencias modifican el flujo del programa introduciendo condicionales. Las sentencias de control para el programa son el IF y el SWITCH.

#Ejemplo de cómo se implementar un ciclo if

```
if (x <50)
{
 Print("Menor que 50");
 #Más sentencias
}
```

```
//Ejemplo de cómo se implementar un ciclo if-else
if (x <50)
{
 print("Menor que 50");
 //Más sentencias
}
else
{
 print("Mayor que 100");
 }
#Ejemplo de cómo se implementar un ciclo else if
if (x > 50)
{
 print("Mayor que 50");
 #Más sentencias
}
else if (x <= 50 && x > 0)
{
 print ("Menor que 50");
 if (x > 25)
 {
 print("Número mayor que 25");
 #Más sentencias
 }
 else
 {
 print("Número menor que 25");
 #Más sentencias
 }
 #Más sentencias
}
else
{
 print("Número negativo");
 #Más sentencias
}
```

19. SWITCH: Switch case es una estructura utilizada para agilizar la toma de decisiones múltiples, trabaja de la misma manera que lo harían sucesivos else-if.
20. CASE: Estructura que contiene las diversas opciones a evaluar con la expresión establecida en el switch
21. DEFAULT: Estructura que contiene las sentencias si en dado caso no haya salido del switch por medio de una sentencia **break**

```
# EJEMPLO DE SWITCH
var edad = 18;
switch( edad ) {
 Case 10:
 Print("Tengo 10 anios.");
 # mas sentencias
 Break;
 Case 18:
 Print("Tengo 18 anios.");
 # mas sentencias
 Case "25":
 Print("Tengo 25 anios en String.");
```

```
# mas sentencias
 Break;
 Default:
 Print(“No se que edad tengo. :(”);
 # mas sentencias
 Break;
}
#* Salida esperada
Tengo 18 anios.
No se que edad tengo. :(
*#
```

22. WHILE: El ciclo o bucle While, es una sentencia que ejecuta una secuencia de instrucciones mientras la condición de ejecución se mantenga verdadera.

```
#Ejemplo de cómo se implementar un ciclo while
While(x<100){
        If(x>50)
        {
                Print(“mayor que 50”);
        }
        Else
        {
                Print(“menor que 50”);
        }
        x++
}
```

23. FOR: El ciclo o bucle for, es una sentencia que nos permite ejecutar N cantidad de veces la secuencia de instrucciones que se encuentra dentro de ella

```
#Ejemplo 1: declaración dentro del for con incremento
for ( var i=0; i<3;i++ ){
print(“i=”+i)
 #más sentencias
 }
#*RESULTADO
 i=0
 i=1
 i=2
 *#
```

24. BREAK: La sentencia break hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del break en la misma iteración no se ejecutara y este se saldrá del ciclo

```
#Ejemplo en un ciclo for
for(var i = 0; i < 9; i++){
 if(i==5){
print(“Me salgo del ciclo en el numero “ + i);
 break;
}
print(i);
}
```

25. CONTINUE: La sentencia continue puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continue siempre debe de estar dentro de un ciclo, de lo contrario será un error.

```
#Ejemplo de continue en un ciclo for
for(var i = 0; i < 9; i++){
 if(i==5){
 print(“Me salte el numero ” + i);
 continue;
 }
```

```
print(i);
#mas sentencias
}
```

26. RETURN: La sentencia return finaliza la ejecución de un método o función y puede especificar un valor para ser devuelto a quien llama a la función. Puede ser devuelto cualquier dato primitivo, null o un arreglo

```
#Ejemplos
func mi_metodo(){
var i;
for(i = 0; i < 9; i++){
if(i==5){
return i; #se detiene y retorna 5
}
print(i);
}
}
func sumar(int n1, int n2){
var n3;
n3 = n1+n2;
return n3; #retorno el valor
}
```

27. FUNCIONES: Una función es una subrutina de código que se identifica con un nombre, un conjunto de parámetros y de instrucciones. Para este lenguaje las funciones serán declaradas indicando que serán funciones, luego un identificador para la función, seguido de una lista de parámetros dentro de paréntesis (esta lista de parámetros puede estar vacía en el caso de que la función no tenga parámetros). Cada parámetro debe estar compuesto por su tipo seguido de un identificador, para el caso de que sean varios parámetros se debe utilizar comas para separar cada parámetro y en el caso de que no se usen parámetros no se deberá incluir nada dentro de los paréntesis. Luego de definir la función y sus parámetros se declara el cuerpo de la función, el cual es un conjunto de instrucciones delimitadas por llaves {}. Como observación, se podrán ingresar arreglos como parámetros en las funciones, donde se indica su tipo seguido de la nomenclatura de un arreglo vacío (arreglo[]) dependiendo en el número de dimensiones que tenga el arreglo. La función puede llevar o no un valor de retorno, si no posee alguna sentencia return y no devuelve ningún valor, será considerado como un método.

```
#Ejemplo de declaración de una función de enteros
Func conversion(double pies, string tipo){
if (tipo == “metro”)
{ return pies/3.281; }
else {
return -1;
}
}
```

28. LLAMADAS: Una llamada a una función cumple con la tarea de ejecutar el código de una función del código de entrada ingresado. La llamada a una función puede devolver un resultado que ha de ser recogido, bien asignándolo a una variable del tipo adecuado o bien integrándolo en una expresión.

```
#Ejemplo de llamada de una función
Print(“Ejemplo de llamada a función”);
var num = suma(6,5) #num = 11
Print(“El valor de num es: ” + num);
#* Salida esperada
Ejemplo de llamada a función
Aquí puede venir cualquier sentencia :D
El valor de num es: 11
*#
```

29. FUNCION PRINT: Esta función nos permite imprimir expresiones con valores únicamente de tipo int, double, booleano, string y char.

```
#Ejemplo
print(“Hola mundo!!”)
print(“Sale compi \n” + valor);
```

```
print(suma(2,2))
```

30. FUNCION READ: Esta función nos permite obtener valores que queramos ingresar en el momento de ejecución del código, el valor ingresado se tomará como un string, por lo que, si se quiere ingresar un número, para tomarlo como tal se debe de castearlo. Se recomienda utilizar una pausa, poder escribir el valor requerido en el área de consola, y al momento de presionar "ENTER", que el programa capture el valor de la última línea de la consola y seguir con su ejecución.

```
#Ejemplo
Var input = read() #Ingresa 100 (String)
Var numInput = (int)input # int 100
print("el valor de input es: " + input); # el valor de input es: 100
```

31. FUNCION TO LOWER: Esta función recibe como parámetro una expresión de tipo String y retorna una nueva cadena con todas las letras minúsculas.

```
#Ejemplo
var cad_1 = toLower("hOla MunDo"); # cad_1 = "hola mundo"
var cad_2 = toLower("RESULTADO = " + 100); # cad_2 = "resultado = 100"
```

32. FUNCION TOUPPER: Esta función recibe como parámetro una expresión de tipo cadena retorna una nueva cadena con todas las letras mayúsculas.

```
#Ejemplo
var cad_1 = toUpper("hOla MunDo"); # cad_1 = "HOLA MUNDO"
var cad_2 = toUpper("resultado = " + 100) # cad_2 = "RESULTADO = 100"
```

33. FUNCION NATIVA LENGTH: Esta función recibe como parámetro un arreglo o una cadena y devuelve el tamaño de este.

```
#Ejemplo
Int[ ] arr1 =new int[ 20 ];
string[ ] arr2 = {"hola", "Mundo"};
var tam_arr1 = length(arr1); # tam_arr1 = 20
```

34. FUNCION NATIVA TRUNCATE: Esta función recibe como parámetro un valor numérico. Permite eliminar los decimales de un número, retornando un entero.

```
# Ejemplo
var nuevoValor = truncate(3.53); # nuevoValor = 3
var otroValor = truncate(10); # otroValor = 10
var decimal = 15.4654849;
var entero = truncate(decimal); # entero = 15
```

35. FUNCION NATIVA ROUND: Esta función recibe como parámetro un valor numérico. Permite redondear los números decimales según las siguientes reglas: Si el decimal es mayor o igual que 0.5, se aproxima al número superior Si el decimal es menor que 0.5, se aproxima al número inferior. Retorna un valor entero

```
# Ejemplo
var valor = round(5.8); //valor = 6
var valor2 = round(5.4); //valor2 = 5
```

36. TYPEOF: Esta función retorna una cadena con el nombre del tipo de dato evaluado.

```
#Ejemplo
Int[][] arr2 =new int[3][2]
String tipo = typeof(15); # tipo = "INT"
String tipo2 = typeof(15.25) # tipo = "DOUBLE"
String tipo3 = typeof(arr2); # tipo3 = "ARREGLO->INT"
```

37. MAIN: Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia MAIN para indicar en dónde se iniciará a ejecutar el programa.

```
#Ejemplo
# declaraciones y asignaciones de variables globales
# declaraciones de funciones
main(){
print("hola soy un mensaje");
#*
Muchas más instrucciones, llamadas a funciones, etc
*#
}
```

declaraciones de funciones

TIPO	DEFINICION	DESCRIPCION	EJEMPLO	OBSERVACIONES
Entero	Int	Este tipo de datos aceptará solamente números enteros.	1, 50, 100, 25552, etc.	Del -9.223.372.036.854.775.808 al 9.223.372.036.854.775.807
Doble	Double	Admite valores numéricos con decimales.	1.2, 50.23, 00.34, etc.	Se manejará cualquier cantidad de decimales
Booleano	Boolean	Admite valores que indican verdadero o falso.	true, false	Si se asigna un valor booleano a un entero se tomará como 1 o 0 respectivamente.
Caracter	Char	Tipo de dato que únicamente aceptará un único carácter, y estará delimitado por comillas simples. ' '	'a', 'b', 'c', 'E', 'Z', '1', '2', '^', '%', ')', '=', '!', '&', '/', '\\', '\n', etc.	En el caso de querer escribir comilla simple escribir se escribirá \ y después comilla simple \', si se quiere escribir \ se escribirá dos veces \\, existirá también \n, \t, \r, \", \'.
Cadena	String	Es un grupo o conjunto de caracteres que pueden tener cualquier carácter, y este se encontrará delimitado por comillas dobles. " "	"cadena1", "-- ** cadena 1"	Se permitirá cualquier carácter entre las comillas dobles, incluyendo los caracteres especiales con secuencias de escape: \" comilla doble \\ barra invertida \n salto de línea \r retorno de carro \t tabulación
Nulo	Null	Este tipo de dato representa la ausencia de valor.	null	Cuando una variable solamente se declara, se le asignará automáticamente este valor. El valor nulo también se podrá asignar a una variable manualmente.

Ilustración 9 TIPOS DE DATOS PERMITIDOS EJEMPLO

SECUENCIA	DESCRIPCION	EJEMPLO
\n	Salto de línea	“Hola\nMundo” -> “Hola Mundo”
\\	Barra invertida	“C:\\miCarpeta\\Personal” -> “C:\miCarpeta\Personal”
\“	Comilla doble	\“Esto es una cadena\” -> “Esto es una cadena”
\t	Tabulación	“\tEsto es una tabulación” -> “ Esto es una tabulación”
\’	Comilla Simple	\’Estas son comillas simples\’ -> ’Estas son comillas simples’

Ilustración 10 TIPOS DE CARACTERES ESPECIALES EJEMPLO

NIVEL	OPERADOR	ASOCIATIVIDAD
0	-	Derecha
1	**	No asociativa
2	/, *, %	Izquierda
3	+, -	Izquierda
4	==, =!, <, <=, >, >=	Izquierda
5	!	Derecha
6	&&	Izquierda
7	||	Izquierda

NOTA: el nivel 0 es el nivel de mayor importancia.

Ilustración 11 PRECEDENCIA DE OPERACIONES EJEMPLO

12.1.1. REPORTES:

12.1.1.1 TABLA DE SIMBOLOS:

Este reporte mostrará la tabla de símbolos después de la ejecución. Se deberán mostrar todaslas variables, funciones y métodos declarados, así como su tipo y toda la información que seconsidere necesaria.

Identificador	Tipo	Tipo	Entorno	Valor	Línea	Columna
Factor1	Variable	Entero	Función multiplicar	null	15	4
Factor2	Variable	Decimal	Función multiplicar	0.7	16	7

12.1.1.2 TABLA DE ERRORES:

El reporte de errores debe contener la información suficiente para detectar y corregir errores en el código fuente

#	Tipo de Error	Descripción	Línea	Columna
1	Léxico	El carácter "$" no pertenece al lenguaje.	7	32
2	Sintáctico	Encontrado Identificador "Ejemplo", se esperaba Palabra Reservada "Valor"	150	12
3	Semántico	No es posible operar String / Char	200	23

12.1.1.3 AST:

Este reporte muestra el árbol de sintaxis producido al analizar los archivos de entrada. Este debe de representarse como un grafo. Se deben mostrar los nodos que el estudiante considere necesarios para describir el flujo realizado para analizar e interpretar sus archivos de entrada.