Objetivos generales

- Que el estudiante aplique los conocimientos aprendidos en el curso de compiladores 1, para poder solucionar problemas de diversos lenguajes de programación.
- Familiarizar al estudiante con la herramienta Jison para análisis léxicos y sintácticos.
- Familiarizar al estudiante con el uso de entornos web desarrollados con Angular.

Objetivos específicos

- Que el estudiante genere e interprete un árbol de análisis sintáctico.
- Que el estudiante comprenda la ejecución de las diferentes sentencias de control de un lenguaje de programación.
- Que el estudiante maneje una tabla de símbolos para realizar acciones de un lenguaje determinado.
- Que el estudiante aprenda a manejar variables en el ámbito global y ámbitos y sub ámbitos locales.
- Que el estudiante consolide una base de conocimientos previa al curso de Lenguajes y Compiladores 1.

Descripción de la actividad

CRL(Compi Report Lenguaje) es un lenguaje de programación para que los usuario puedan comprender cómo se generan los subárboles de análisis sintáctico de las sentencias de control de un lenguaje, los métodos y funciones de una clase y las diversas tablas de símbolos que se crean en tiempo de compilación en los ámbitos y sub ámbitos de funciones y sentencias de control. Por lo que el lenguaje permitirá generar reportes gráficos de estas estructuras y así obtener una mejor comprensión del funcionamiento del lenguaje.

El estudiante podrá realizar programas con el lenguaje CRL donde manejara declaración de variables, asignación de variables, métodos y funciones, expresiones racionales, expresiones lógicas y aritméticas, ejecución de sentencias de control y realización de reportes.

Este lenguaje usará el mismo tipo de agrupación e indentación que el lenguaje python por lo que nos olvidaremos de los puntos y comas y las llaves.

IDE CRL

El IDE solicitado facilitará entre otras tareas la creación, descarga, subida y la ejecución de los programas contenidos en archivos de texto que tendrán la extensión crl; además soportará múltiples pestañas para poder trabajar con varios archivos a la vez. El IDE también contará con la capacidad de analizar e interpretar el contenido de cualquiera de los archivos abiertos; mostrará las distintas salidas (texto e imágenes) generadas y los reportes de errores generados como producto del análisis y ejecución de determinado archivo.

Se deberán tener las siguientes funciones mínimas.

1. **Nuevo**: Creará una nueva pestaña en blanco en el editor.
2. **Cargar**: Abrirá un cuadro de diálogo para seleccionar un archivo con extensión crl y mostrará su contenido en una nueva pestaña.
3. **Descargar**: Descarga a nuestra el archivo con de la pestaña seleccionada con extensión crl .
4. **Cerrar pestaña**: Esta opción eliminará la pestaña de la interfaz gráfica.
5. **Ejecutar archivo**: Ejecutará el contenido de la pestaña actual.
6. **Ver reporte de errores**: Enviará al usuario a la sección o al archivo de reporte de errores.
7. **Colección de Imagines**: En este panel se podrán visualizar o descargar los reportes gráficos generados por el programa CRL.
8. **Consola de Salida**: En esta área se visualizarán todas las expresiones o cadenas que se envíen a imprimir desde el programa.

Esquema de la solución

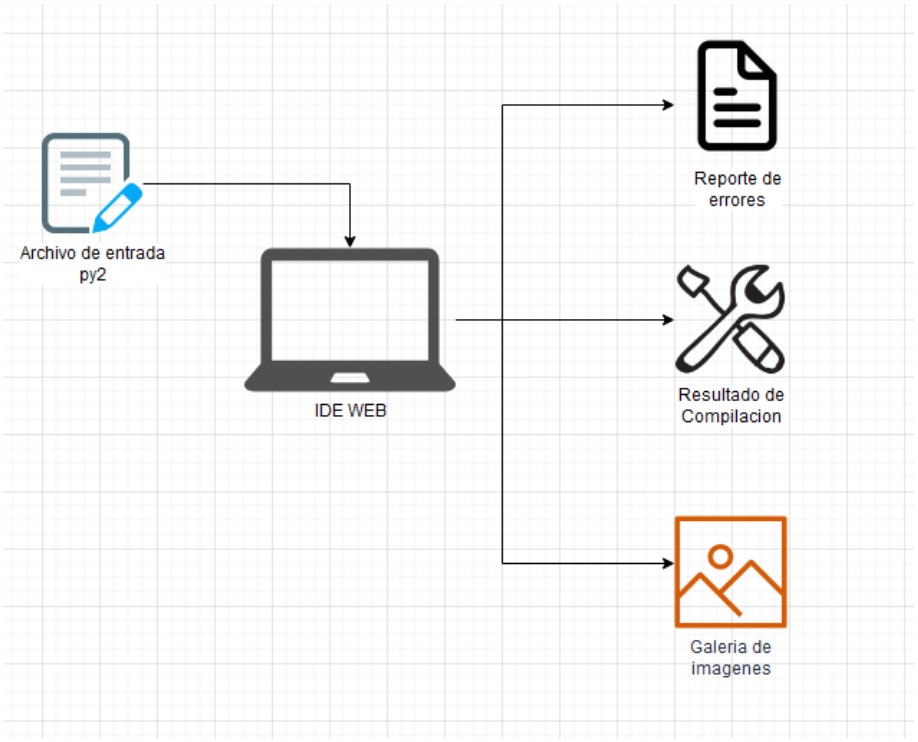


Imagen No.1 Esquema de funcionamiento.

1. Se tendrá uno o varios archivos de entrada con el lenguaje CRL los cuales serán cargados en el IDE de CRL.
2. El analizador de CRL procederá a compilar el o los archivos y a realizar las acciones que se describen en el lenguaje.
3. Si existen errores debe generarse el correspondiente reporte de error que describa lo más posible el error y su tipo.
4. En el IDE deben mostrarse los reportes gráficos que se generaron en la compilación del archivo.

DESCRIPCIÓN DEL LENGUAJE CRL

1. Especificaciones Generales:

El lenguaje CRL será capaz de soportar:

- Comentarios.
- Sensibilidad entre mayúsculas y minúsculas.
- Datos de tipo numérico (no hay diferencia entre entero o decimal), booleano y cadena.
- Realización de operaciones aritméticas, lógicas y relacionales.
- Declaración de variables.
- Definición de funciones con un tipo asociado a su retorno y un conjunto de parámetros.
- Sobrecarga de funciones por medio de sus parámetros.
- Llamadas a funciones (incluyendo recursividad directa o indirecta).
- Uso de sentencias propias del lenguaje.
- Ciclos y sentencias de control de flujo.

2. Encabezado:

El inicio de todo archivo CLR cuenta con una sección de declaraciones en donde se declararán aspectos generales que a continuación se definen y ejemplifican:

- Se pueden Importar otros archivos crl con la directiva ‘Importar’ acompañada del nombre del archivo crl, que como su nombre lo indica, se encargará de Importar el contenido de dicho archivo crl adjunto al contenido del archivo que se está analizando (como que todos fueran un solo archivo). Si el archivo a Importar no existe debe reportarse un error.

- Se puede definir un valor de incerteza para realizar comparaciones entre valores numéricos (ver sección Expresiones Relacionales) por medio de la directiva ‘Incerteza’ acompañada de un valor numérico. Si no se define valor para la incerteza se tomará un valor por defecto de 0.5. La incerteza es independiente para cada archivo.

  Ejemplo de un encabezado:

  Importar aritmeticas.clr
  Importar relacionales.clr
  Incerteza 0.0002

Este ejemplo Importaría el contenido (ya analizado) de los archivos aritmeticas.crl y relacionales.crl; y utilizará una incerteza de 0.0002

3. Comentarios:

En CRL se permite escribir comentarios de la siguiente forma:
Los comentarios de una sola línea empezarán con !! y terminarán con un salto de línea.
Los comentarios de múltiples líneas empiezan con ‘” , y terminarán ‘”

4. Tipos de Datos:

Para las variables se permiten los tipos de datos indicados en la siguiente tabla, mientras que para las funciones se permiten los mismos tipos de datos y adicional a ello se admite también el tipo Void, cuya palabra reservada será Void que indicará que una función no tiene tipo de retorno asignado.

Tipo de dato	Ejemplos
Double	90.1 -120.210 20.06 -30.2
Boolean	true false
String	“Esto es una cadena” “\n”
Int	1 5 66 77
Char	‘a’ ‘1’ ‘r'
Void	Solo para métodos

Tabla No. 1

5. Casteo implícito:

Al momento de que un tipo booleano se vea implicado en una operación numérica permitida, este debe interpretarse como un 0 si su valor es falso, y como un 1 si su valor es verdadero.

Operación de ejemplo	Resultado
true + 5.90	6.90
90.206 * false	0
1+false	2

Tabla No. 2

Para realizar una concatenación entre una cadena y un valor booleano es necesario convertir los valores booleanos a su equivalente en texto, es decir, sí el valor booleano es verdadero, en la cadena resultante de la concatenación se debe agregar el carácter “1”; caso contrario el valor booleano sea falso, el texto a concatenar será el carácter “0”.

Operación de ejemplo	Resultado
false + “ hola mundo”	“0 hola mundo”
“hola mundo” + true	“hola mundo 1”

Tabla No. 3

En las concatenaciones donde intervienen datos de tipo *Double/Int* se debe obtener el texto que representa el valor numérico y agregarlo a la cadena resultante.

Operación de ejemplo	Resultado
“La suma de 5 + 3 es: ” + 8	“La suma de 5 + 3 es: 8”
3.1416 + “ es el valor de Pi”	“3.1416 es el valor de Pi”

Tabla No. 4

En las operaciones aritméticas de un char entre un double o int, se deberá tomar el valor ascii

Operación de ejemplo	Resultado
1+’a’	98
2.2*b	215.6

Tabla No. 5

6. Expresiones Aritméticas:

Las expresiones aritméticas soportadas por el lenguaje son las siguientes las cuales según el tipo de los operadores y el operador da como resultado un nuevo valor y tipo, tal como se describe en la siguiente tabla.

	+	Boolean	Double	String	Int	Char
+	Boolean	Double	Double	String	Int	Error
	Double	Double	Double	String	Double	Double
	String	String	String	String	String	String
	Int	Int	Double	String	Int	Int
	Char	Error	Double	String	Int	Int
	-	Boolean	Double	String	Int	Char
-	Boolean	Error	Double	Error	Int	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Int	Double	Error	Int	Int
	Char	Error	Double	Error	Int	Int
	*	Boolean	Double	String	Int	Char
*	Boolean	Error	Double	Error	Int	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Int	Double	Error	Int	Int
	Char	Error	Double	Error	Int	Int
	/	Boolean	Double	String	Int	Char
/	Boolean	Error	Double	Error	Double	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Double	Double	Error	Double	Double

	Char	Error	Double	Error	Double	Double
%	**%**	**Boolean**	**Double**	**String**	**Int**	**Char**
	Boolean	Error	Double	Error	Double	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Double	Double	Error	Double	Double
	Char	Error	Double	Error	Double	Double
^	**^**	**Boolean**	**Double**	**String**	**Int**	**Char**
	Boolean	Error	Double	Error	Double	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Double	Double	Error	Int	Int
	Char	Error	Double	Error	Double	Double

Tabla No. 6

7. Expresiones Relacionales:

Además de las expresiones aritméticas, también es posible definir operaciones relaciones, cuyo valor siempre será un valor booleano. Estas operaciones solo son aplicables entre expresiones del mismo tipo, de lo contrario se debe reportar error. Entre datos numéricos, las comparaciones se comportan de manera natural, haciendo la comparación que su nombre indica; Para los booleanos true = 1 y false = 0

Igual	Diferente	Menor	Mayor	Menor o igual	Mayor o igual	Incerteza
==	!=	<	>	<=	>=	~

Tabla No. 7

Para las cadenas, las comparaciones deben realizarse siguiendo el orden lexicográfico de cada uno de los caracteres que componen las cadenas a comparar. Por ejemplo:

Cadena 1	**Cadena 2**	**Resultado**
"Agua"	"Aguacate"	**Cadena 2 es mayor que Cadena 1**
La cadena es idéntica en sus primeros 4 caracteres, pero como la cadena 2 aún tiene caracteres el resultado Sería que la cadena 2 es "mayor que" la cadena 1.		
"compiladores 1"	"compiladores1"	**Cadena1 es menor que Cadena 2**
En este caso la cadena1 es menor que la cadena 2 porque el ascii asociado al espacio en blanco (32) es Menor que el valor del ascii asociado al dígito '2' (50).		
"japón"	"japón"	**Cadena 2 es igual que Cadena 1**
Las cadenas son idénticas.		

Tabla No. 8

Función del operador de la Incerteza ~ (el código ascii del operador es 126)

- Al principio de todo archivo CRL es posible definir un grado de incerteza, que será un valor que servirá para utilizar este operador (ver sección de encabezados), dicho grado de incerteza se aplicará para comparar valores numéricos, como por ejemplo:

  10.54 ~ 10.50

- Con un valor de incerteza de **0.5**, el resultado de esta comparación sería **true**. Porque el valor absoluto de la diferencia que existe entre los dos valores (|10.54 - 10.50| = **0.04**) es menor o igual que el valor de incerteza declarada.

- Si en otro caso la incerteza fuese de **0.01**, el resultado de la misma comparación sería false. Porque el valor absoluto de la diferencia entre los valores sería mayor que la incerteza declarada.

- El mismo operador "~" será utilizado para comparar cadenas, dicha comparación ignorará los espacios en blanco al inicio y al final de las cadenas, tampoco distinguirá entre mayúsculas y minúsculas, bajo estas condiciones se realizará la comparación de las cadenas (la incerteza NO tiene nada que ver en las comparaciones entre cadenas). Por ejemplo, la siguiente comparación sería verdadera:

  " correo@mail.com" ~ "Correo@mail.com "

- Se ignoraría el hecho de que una de las dos palabras inicia con C mayúscula y la otra con c minúscula, y además ignoraría los espacios en blando al final de la

cadena de la derecha, por lo tanto las cadenas son semejantes. Otro ejemplo, la siguiente comparación sería falsa:

"Compi1" ~ "Compi 1"

- A pesar de que las dos cadenas cuentan con los mismo caracteres, los espacios que se encuentren en medio de cada cadena no deben ser ignorados, por lo tanto estas cadenas no cumplen con ser semejantes.

8. Expresiones Lógicas:

También es posible utilizar expresiones de índole lógico, para este tipo de expresiones los operandos siempre tienen que ser de tipo booleano y el resultado de igual forma siempre será un valor booleano, de lo contrario será error.

And (y lógico)	Or (o lógica)	Xor	Not (negación lógica)
&&	||	|&	!

Tabla No. 9

9. Precedencia y asociatividad de operadores

En la siguiente tabla se definirá la precedencia y asociatividad para cada uno de los operadores definidos en esta sección de expresiones, ordenados de menor a mayor precedencia. Los paréntesis ( ) son los únicos símbolos que se utilizarán para agrupar operaciones y "romper" la precedencia normal de las operaciones.

Símbolo	Precedencia	Asociatividad
+,-	1	Izquierda
* / %	2	Izquierda
^	3	Derecha
- (unario)	4	No aplica
==, !=, <, >, <=, >=, ~	5	No aplica
||	6	Izquierda
!&	7	Izquierda
&&	8	Izquierda
!	9	Izquierda
( )	10	No aplica

Tabla No. 10

Nota: A partir de esta sección del enunciado los elementos encerrados entre corchetes [ ] y escritos con un color gris indica opcionalidad, es decir que, dichos elementos pueden o no venir dentro de la sintaxis de CRL.

10. Declaración de Variables

Para la declaración de variables se utilizará una sintaxis en donde primero se escriba el tipo de la variable y luego una lista de identificadores separados por coma, para finalizar siempre con un salto de línea, de manera opcional se podrá realizar una asignación inicial anteponiendo un signo igual (=) seguido de una expresión cuyo valor será asignado directamente a cada una de las variables definidas en la lista de identificadores.

- Las variables globales son aquellas declaradas fuera del cuerpo de las funciones y son accesibles desde cualquier función.
- Las variables locales son accesibles únicamente dentro del ámbito donde fueron declaradas.

<TIpo> <Lista_IDS> **[=<Expresión>]**
Ejemplos:
Double a,b,c=50.5
Char a=’a’
Int valor=’a’+1-2
String str1=hola(“mundo”)
En el último caso se asigna a str1 el valor de retorno de una función String llamada hola.

11. Asignaciones

Las asignaciones siguen una sintaxis básica; Primeramente el identificador de la variable sobre la que se realiza la asignación, seguido del signo igual (=), luego la expresión que representa el nuevo valor para la variable y por último un salto de línea.

<TIpo> = **<Expresión>**
Ejemplos:
c=50.5
a=’a’
valor=’a’+1-2
str1=hola(“mundo”)

Durante la asignación de una expresión a una variable el valor de dicha expresión puede ser convertido de manera implícita siguiendo la siguiente convención de tipos.

Destino	Tipo de la expresión	Acción de conversión (casteo implícito)
String	String	Sin acción
	Double	El número se convierte a cadena
	Boolean	La cadena almacenará “1” si es true y “0” si es false

	Int	El entero se convierte a cadena: 1 == “1”
	Char	El Char se convierte a cadena, ‘a’ es igual a “a”
Double	String	Error en la asignación
	Double	Sin acción
	Boolean	Si es true asigna un 1, de lo contrario asigna un 0
	Int	El entero pasa a ser double: 1==1.0
	Char	El char pasa a ser doublé: ‘a’ == 97.0
Boolean	String	Error en la asignación
	Double	Error en la asignación
	Boolean	Sin acción
	Int	Error en la asignación
	Char	Error en la asignación
Int	String	Error en la asignación
	Double	Solo se toma el numero entero: 1.2 == 1
	Boolean	Si es true asigna un 1, de lo contrario asigna un 0
	Int	Sin acción
	Char	El char pasa a ser int: ‘a’ == 97
Char	String	Error en la asignación
	Double	Error en la asignación
	Boolean	Error en la asignación
	Int	El Int pasa a ser char, se debe tomar en cuenta el rango de la tabla asscii
	Char	Sin acción

Tabla No. 11

12. Funciones/Métodos:

Una función/Método es una subrutina de código que se identifica con un nombre y que puede contar con parámetros (y un tipo para su retorno). Las funciones serán declaradas definiendo primero su tipo, luego su identificador, seguido una lista de parámetros (que puede no venir) delimitada por paréntesis. Cada parámetro estará compuesto por su tipo, seguido de su identificador y cada uno de ellos estará separado del otro por una coma. Después de la definición de la función se colocan dos puntos : y se declarará el cuerpo de la misma, identando su contenido exactamente igual a la sintaxis python.

<Tipo> <ID> ([<Parametros>]):

    <INSTRUCCIONES>

Ejemplo:

```
String Saludo (String nombre):
    Retorno “Hola ”+nombre+” :D”
```

Las funciones pueden ser sobrecargadas, lo que significa que pueden haber dos o más funciones con el mismo nombre, pero con distinta “llave”. La llave que identifica a una función de otra serán los tipos y/o cantidad de sus parámetros.

Para los métodos, el tipo deberá ser obligatoriamente Void y no deberá retornar nada. Si es posible que los métodos tengan retorno, pero sin valor.

Void <ID> ([<Parametros>]):

    <INSTRUCCIONES>

Ejemplo:

```
Void Saludo (String nombre):
    Retorno
```

13. Función Principal:

La función principal es la subrutina que arrancará las acciones del intérprete, esta subrutina tendrá siempre la misma estructura

Void Principal():

<INSTRUCCIONES>

Ejemplo:

```
Void Principal():
    Boolean bandera=true
    String gane=Compi1(bandera)
```

14. Llamadas a Funciones:

Una llamada a función se realiza escribiendo el identificador de la función; seguido de los valores que tomarán los parámetros para dicha llamada, separados por coma y envueltos en un juego de paréntesis; finalizando con un salto de línea

<ID>([<LISTA_PARAMETROS>])

Ejemplos:

HolaMundo()

FuncionParam(“Jonathan”+” Mateo”, true)

15. Sentencia de Retorno:

Esta sentencia finalizará la ejecución de la función y devolverá el valor indicado por la expresión que se encuentra adyacente a ella; Si durante la ejecución del código se encuentra la palabra ‘Retorno’ sin ninguna expresión asociada, se terminará la ejecución del cuerpo de la función. Esta sentencia inicia con la palabra reservada ‘Retorno’ luego, puede o no venir una expresión y finaliza con un salto de línea.

Retorno [<EXPRESION>]

Ejemplos:

Retorno 5+1+3

Retorno

16. Sentencia Si:

Esta instrucción inicia con la palabra reservada 'Si', seguida de una condición encerrada entre paréntesis y que cuenta con un cuerpo de instrucciones que se realizarán en caso de que la condición sea verdadera. Además cuenta de manera opcional con un cuerpo de instrucciones que se ejecutará en caso la condición no se cumpla, este cuerpo de instrucciones viene seguido del primer conjunto de sentencias separado únicamente por la palabra reservada 'Sino', este cuerpo de instrucciones también viene delimitado por la indentación.

Si(<Expresion>):

    <INSTRUCCIONES>

[Sino:

    <INSTRUCCIONES>

]

Ejemplo

Si (a<max):
    Mostrar ("a es menor a max")
Sino:
    Mostrar ("a es mayor a max")

17. Sentencia Para:

Este ciclo inicia con la palabra reservada *'Para'* consta de una declaración inicial de una variable de tipo *Int*; un punto y coma; una condición para mantenerse en el ciclo; un incremento (++) o decremento (--) que al final de cada iteración realizada afectará directamente a la variable declarada al inicio de este ciclo y un cuerpo de sentencias a ejecutar siempre y cuando la condición sea verdadera. El ciclo de cada iteración será: Evaluar la condición; si es verdadera, ejecutar cuerpo y realizar incremento o decremento, regresar a evaluar la condición; si es falsa, salir del ciclo.

Para (Int <ID>=<EXPRECION>;<EXPRECION>;<OP*>:

    <INSTRUCCIONES>

OP puede ser sustituido por ++ o –

Ejemplo:

Para (Int i=1;i<11;++):

    Mostrar("5 x {0}={1}", i,5*i)

18. Sentencia Mientras:

Este ciclo consta de una estructura que inicia con la palabra reservada 'Mientras', seguido de una condición (envuelta entre paréntesis) y al final constará con un cuerpo de sentencias (delimitadas por la identacion) que se ejecutarán toda vez la condición del ciclo sea verdadera.

Mientras (<EXPRESION>):

    <INSTRUCCIONES>

Ejemplo:

Mientras (Trabajo>Descanso):

    Mostrar ("Hay esperanzas de de ganar compi1")

19. Sentencia Detener:

Esta sentencia es aplicable toda vez se encuentre dentro del cuerpo de una sentencia Mientras o Para. Al encontrarla la ejecución deberá detenerse y regresar el foco de control al nivel superior de la sentencia que ha sido detenida.

Detener

Ejemplo:

Mientras (Trabajo>Descanso):

    Mostrar ("Hay esperanzas de de ganar compi1")

    Detener

20. Sentencia Continuar:

Esta sentencia es aplicable toda vez se encuentre dentro del cuerpo de una sentencia Mientras o Para. Al encontrar esta sentencia dentro de un ciclo se detendrá la ejecución del mismo y saltará directamente a evaluar la condición para la siguiente iteración; en el caso particular de la sentencia Para, se deberá ejecutar el incremento o decremento antes de saltar a evaluar la condición para la próxima iteración del ciclo.

Continuar

Ejemplo:

Para (Int i=1;i<11;++):

    Continuar

21. Función Mostrar:

La función ‘Mostrar’ se encargará de imprimir en consola la lista de expresiones que reciba como parámetros colocando dichos parámetros bajo el formato de una máscara determinada por un String. En dicha máscara, cada parámetro está asociado con los caracteres {!!}, donde !!, es el índice del parámetro que acompaña al formato. Los caracteres “{!!}” serán sustituidos por el valor del parámetro que corresponda, los índices inician en 0.

Mostrar (<String> [,<expresiones>])

Ejemplo:

Mostrar (“Fecha: {0} de {1} de {2} \n”,20, “Abril”, 2022)

>Fecha 20 de abril de 2022

22. Función DibujarAST:

La función 'DibujarAST' es una función de CRL que se encargará de dibujar el AST del cuerpo de una función, si existen varios funciones con el mismo nombre se generarán, tantas imágenes como funciones con determinado nombre existan, agregando al nombre del método, su "llave" de parámetros; En esta imagen NO se deberán dibujar los nodos de expresiones, en su lugar se colocarán nodos para representar la existencia de una expresión de forma resumida.

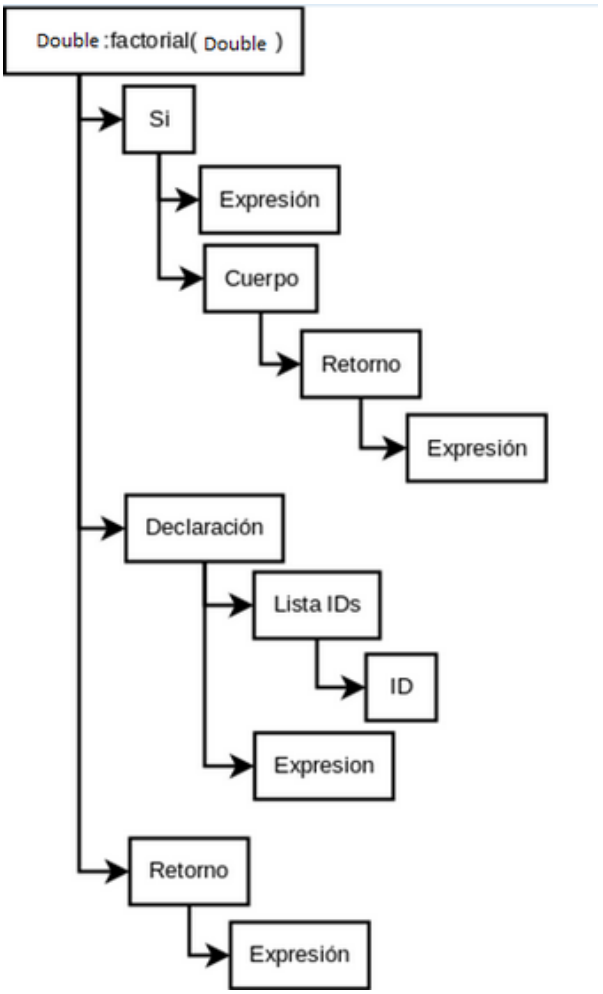
DibujarAST(<ID>)

Ejemplo:

DibujarAST(Factorial)

Resultado gráfico de dibujar la función factorial:

```
Double Factorial (Double n):
        Si(n<=0):
                Retorno 1
        Double res=n*Factorial(n-1)
        Retorno res
```



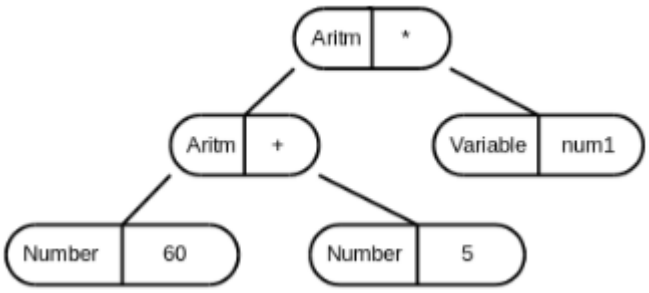
23. Función DibujarEXP:

La función 'DibujaEXP' crea una imagen del AST asociado a una expresión en particular, incluyendo todos los tipos de operaciones y los datos más específicos, es decir, valores puntuales Double, String, Boolean, Identificadores

DibujarEXP (<EXPRESION>)

Ejemplo:

DibujarEXP((60+5)*num1)



23. Función DibujarTS::

La función 'DibujarTS' crea una imagen de la tabla de símbolos del ámbito o sub ámbito donde se invoque dicha función, por lo que se limitará a representar la información como tipo de ámbito o subámbito, variables contenidas con su respectiva información.

DibujarTS()

Ejemplo:

```
Mientras (X<10):
        DibujarTS()
        Double i, j=10
        String cad_1=”Hola mundo”
        Si (i==10):
                Boolean val=true
```

El resultado de la tabla de símbolos únicamente debería sacar las variables del ámbito del mientras, pero no del sub ámbito del SÍ, quedando de la siguiente forma, suponiendo que el mientras se encuentra dentro de la función Suma:

Ámbito	Funcion_Suma_SubAmbito_Mientras
Variable i	Valor:10 Línea 15 Columna 20, TIpo:Double
Variable j	Valor:10 Línea 15 Columna 20, TIpo:Double
Variable cad_1	Valor:”Hola mundo” Línea 16 Columna 20, TIpo:String

Importante

- Toda sección de edición de archivos debe tener una numeración de línea para poder detectar los errores con mayor facilidad.
- Usar herramientas Jison para cualquier tipo de análisis/proceso léxico y sintáctico.
- El proyecto deberá realizarse en un entorno web utilizando Angular.
- Las copias obtendrán nota de cero y se notificará a coordinación.
- Si se va a utilizar código de internet, entender la funcionalidad para que se tome como válido.

Entrega

La fecha de entrega es el día de XX Mayo a las 14:00 horas Los componentes a entregar utilizando un repositorio git son:

- Código fuente
- Manual técnico: Detalle de la organización de su proyecto, análisis de gramática para analizador léxico y gramática para analizador sintáctico, diagrama de clases.
- Manual de usuario.

Calificación

Pendiente de definir.