**Manual de Usuario Proyecto 2.**

Descargar Repositorio en GitHub.

Desde la consola de la carpeta de la aplicación App-Compiladores1:

- Instalar todas las dependencias desde la consola con el comando **npm install**.
- Aumentar la memoria de angular con: **node --max_old_space_size=6144 ./node_modules/@angular/cli/bin/ng -o serve**
- Navegar en **http://localhost:4200/**
- También se puede ejecutar la aplicación con **ng serve App-Compiladores1**

**Aplicación.**

La aplicación tiene una barra para navegar entre las distintas ventanas.

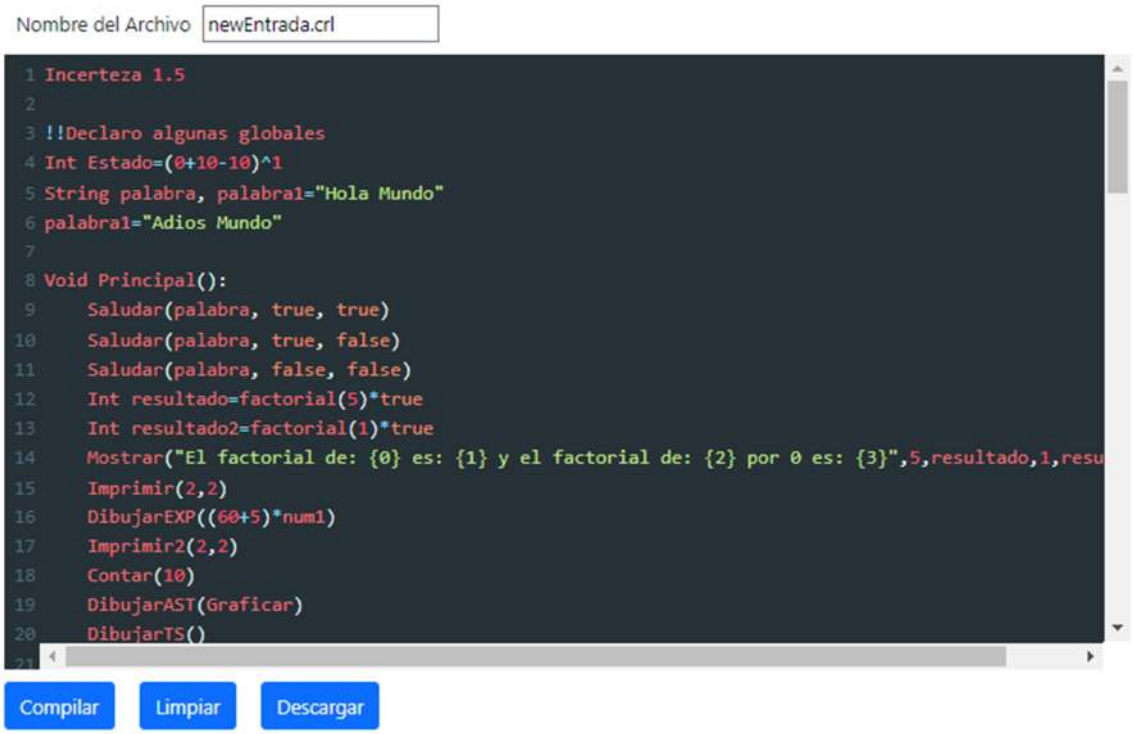

La ventana del editor consta de un área de texto donde se escribirá el código CRL (Explicado en su respectiva sección).

Consta de un apartado para subir un archivo con extesion .crl. Donde al presionar el botón de importar, el contenido de dicho archivo se agregara al área de texto.


También hay una caja de texto para ingresar el nombre de nuestro archivo para posteriormente poder descargarlo. Abajo del área de texto están los botones de:

- Compilar: Compila el código para mostrar su resultado en la consola.
- Limpiar: Limpia todo el código del área de texto.

- Descargar: Descarga el archivo a nuestra computadora.




La consola mostrara los resultados de las instrucciones “Mostrar”.


## Reporte de Errores

En este apartado se podrán visualizar todos los errores léxicos, sintácticos y semánticos. Junto con la fila en donde se dio el error y la columna.

## Reporte de Errores

| Tipo | Mensaje | Linea | Columna |
|---|---|---|---|
| SINTACTICO | Error en el lexema: Strisng | 5 | 8 |
| SINTACTICO | Error en el lexema: , | 5 | 17 |
| SEMANTICO | La variable palabra no esta definida | 9 | 9 |
| SEMANTICO | Parametro Pos: 1 invalido | 9 | 1 |

### Reporte de Tablas

En este apartado se podrán visualizar todas las tablas que se hayan generado con la instrucción DibujarTS(). Se mostrarán símbolos, valores, tipo, fila y columna.

## Simbolos de la tabla dibujada en la fila: 22

| Id | Valor | Tipo | Fila | Columna |
|---|---|---|---|---|
| variable1 | Hola Mundo | CADENA | 9 | 1 |
| caracter1 | 97 | CARACTER | 10 | 1 |
| resultado | 120 | ENTERO | 14 | 1 |
| resultado2 | 1 | ENTERO | 15 | 1 |

## Reporte de Graficas.

En este apartado se visualizaran todas las graficas generadas con las instrucciones DibujarAST() y DibujarEXP.

Grafica No.1


Grafica No.2


## Gramática

El lenguaje CRL será capaz de soportar:

- Comentarios.
- Sensibilidad entre mayúsculas y minúsculas.
- Datos de tipo numérico, 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.

**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 puede definir un valor de incerteza para realizar comparaciones entre valores numéricos por medio de la directiva 'Incerteza' acompañada de un valor numérico de tipo decimal. Si no se define valor para la incerteza se tomará un valor por defecto de 0.5.

Ejemplo de un encabezado:
Importar aritmeticas.clr
Importar relacionales.clr
Incerteza 0.0002

**Comentarios**:

Los comentarios de una sola línea empezarán con !! y terminarán con un salto de línea. Los comentarios de múltiples líneas empiezan con "", y terminarán ""

Ejemplos:

- !! Este es un comentario de una línea
- ""

  Este es un comentario multilínea

  ""

**Tipos de Datos**

Para las variables se permiten los tipos de datos Int, Double, String, Char, Boolean, 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.

Ejemplos:

- Int: 54, 1, 26
- Double: 1.02, 20.9, 30.74
- String: “Hola Mundo”, “cadena”
- Boolean: true, false
- Char: ‘a’, ‘5’, ‘*’

**Expresiones Aritméticas:**

CRL soporta sumas, restas, multiplicaciones, modulo, y potencias. Entre cualquier tipo de dato.

Ejemplos:

- 15 * 6
- 1.25 + 0.65
- “Hola: “ + 50
- 10 + true
- 11 / ‘c’

Nota: El valor de un carácter será su respectivo valor en la tabla ASSCII

**Expresiones Relacionales**

El lenguaje soporta expresiones relaciones cuyo valor será solo de tipo boolean. Solo se pueden realizar comparaciones entre valores del mismo tipo de dato, sino se marcará un error. Las comparación que puede realizar son las siguientes:

- ==
- ¡=
- >
- <

- >=
- <=
- ~

Nota: Las comparaciones entre cadenas se realizan comparando la suma de sus valores de todos los caracteres basándose en la tabla ASSCII.

**Incerteza:**

Al principio de todo archivo CRL es posible definir un grado de incerteza, que será un valor que servirá para utilizar este operador, 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.

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 mismos 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.

**Operaciones 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. Estas operaciones pueden ser:

- AND: &&
- OR: | |
- XOR: |&
- NOT: !

**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.
Ejemplos:
String cadena1 = “Hola mundo”

Int variable1 = 50
Double decimal1, decimal2, decimal3 = 0.25
Char carácter1 = ‘g’

**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.

Ejemplos:

- a = 0.5
- cadena1 = “Esto es una cadena”
- variable1 = 100

| 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 |

## Funciones

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.

Ejemplos:

```
Int sumarNumeros(Int num1, Int num2):
    Int suma = num1 + num2
    Retorno suma

Void saludar():
    Mostrar("Hola mundo")
```

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

**Función Principal**

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

```
Void Principal():
    <Instrucciones>
```

**Nota:** Al final de cada función se debe dejar un espacio en blando antes de agregar una función nueva.

**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. También se pueden asignar a variables.

```
HolaMundo()
Int valorSuma = funcionSumar(5, 5)
```

**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.

Ejemplos:

Retorno 5+6
Retorno “Hola”
Retorno

**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 (a<max):
    <instrucciones>
Sino:
    <instrucciones>

**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.

Ejemplo:

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

    <instrucciones>

**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 (Trabajo>Descanso):

    <instrucciones>

**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.

Mientras (Trabajo>Descanso):

    Detener

**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.

Ejemplo:

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

    Continuar

**Sentencia 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.

Ejemplo:

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

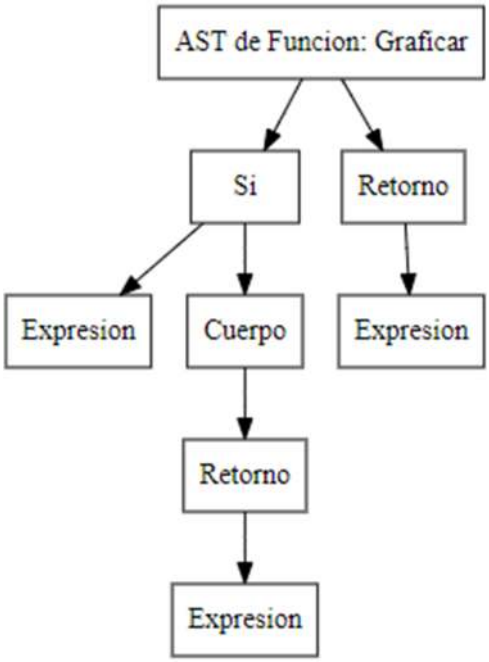
Salida -> Fecha 20 de abril de 2022

**Sentencia 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.

Ejemplo:

DibujarAST(Graficar )

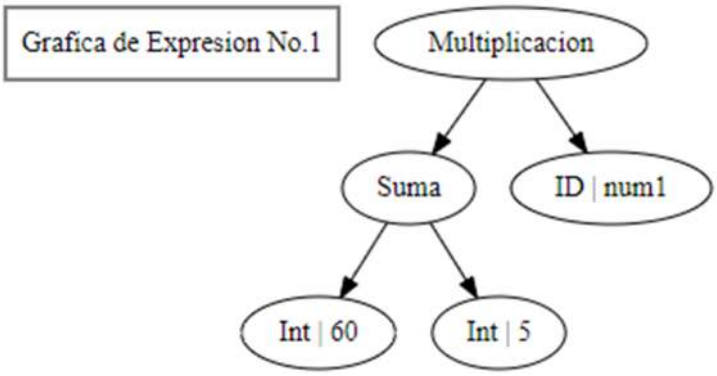



**Función DibujarEXP**

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

Ejemplo:

DibujarEXP((60+5)*num1)




**Funcion 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.

Simbolos de la tabla dibujada en la fila: 20

| Id | Valor | Tipo | Fila | Columna |
|---|---|---|---|---|
| resultado | 120 | ENTERO | 12 | 1 |
| resultado2 | 1 | ENTERO | 13 | 1 |