**MANUAL DE USUARIO PARA LA APLICACIÓN “SYSCOMPILER”**

Esta aplicación está enfocada para los estudiantes de IPC1, para que estos aprendan a programar y tener conocimiento de todas las generalidades de un lenguaje de programación.

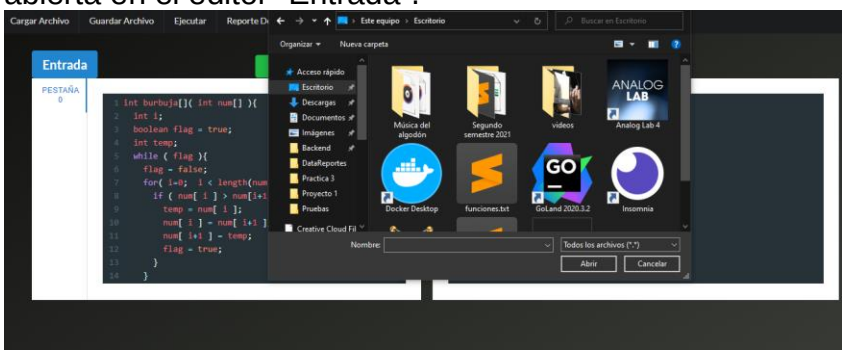
En este manual se describen los requerimientos del sistema.

- **Requerimientos:**
  - Cualquier sistema operativo.
  - Docker, verificando los requisitos del sistema.
  - El repositorio donde se encuentra la aplicación.

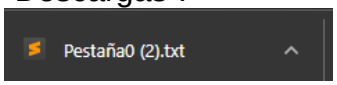
- **Instalación:**
  - Descargar/Clonar el repositorio de github en el dispositivo donde utilizará la aplicación. Luego deberá abrir el símbolo del sistema, para entrar desde ella a la carpeta de la aplicación “syscompiler” y ejecutar el comando: “docker-compose up -d”. Finalmente deberá ingresar al siguiente link: http://localhost:3000/ App.

- **Uso de la aplicación:**
  - Al ingresar a la aplicación podrá ver un menú en la parte superior de la pantalla y en el centro 2 editores de texto.

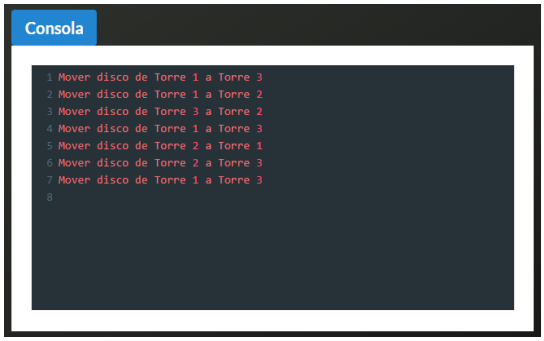

  - **Menú superior:**
    - **Cargar archivo:** Podrá cargar un archivo .sc a la aplicación, el texto del archivo se colocará en la pestaña que está actualmente abierta en el editor “Entrada”.



- **Guardar archivo:** Se descargará un archivo .sc al dispositivo, este archivo tendrá por nombre “Pestaña”#Pestaña y se colocará en “Descargas”.




- **Ejecutar:** Al darle clic a este botón, la aplicación tomará el texto de entrada y se ejecutará, luego mostrará el resultado de la ejecución en la consola. Si el código tiene errores, estos se mostrarán en la consola y en la sección “Reporte de Errores”, que se explicará a continuación.




- **Reporte De Errores:** En esta sección se mostrarán los errores obtenidos durante la ejecución del código y en las líneas donde se intentó recuperar el analizador.

| MENSAJE | TIPO | LINEA | COLUMNA |
|---|---|---|---|
| . No se esperaba: ID | Sintáctico | 10 | 0 |
| Tipo de Declaración incorrecta. Recuperado en esta línea, con ";"". | Sintáctico | 11 | 25 |

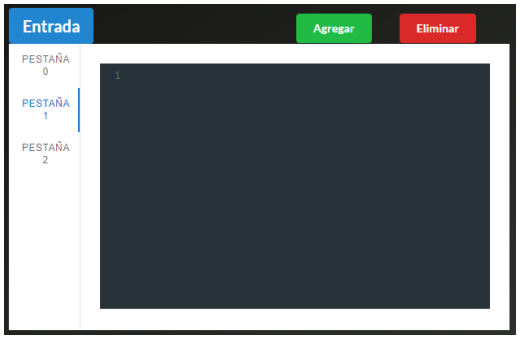
- **Árbol AST:** Aquí podrá visualizar el árbol ast generado durante la ejecución del código, mostrándolo en un pdf para no perder la calidad de la imagen.

- **Reporte de Símbolos:** Se redirigirá a una página, donde mostrará en una tabla los símbolos obtenidos durante la ejecución del programa.

| ID | DATO | TIPO | ENTORNO | LINEA | COLUMNA |
|---|---|---|---|---|---|
| burbuja | FUNCION | VECTOR INT | Global | 1 | 0 |
| fibonacci | FUNCION | INT | Global | 19 | 0 |
| parImpar | MÉTODO | VOID | Global | 36 | 0 |
| torresHanoi | MÉTODO | VOID | Global | 50 | 0 |
| main | MÉTODO | VOID | Global | 61 | 0 |
| discos | VARIABLE | INT | torresHanoi | 56 | 4 |
| torre1 | VARIABLE | INT | torresHanoi | 56 | 4 |
| torre2 | VARIABLE | INT | torresHanoi | 56 | 4 |
| torre3 | VARIABLE | INT | torresHanoi | 56 | 4 |

- **Agregar:** Aquí se podrán agregar más pestañas en la entrada del programa.


- **Eliminar:** Se podrá eliminar la pestaña que está actualmente abierta y los números de las pestañas se actualizará para tener un orden.




- **Sintaxis para el uso de la aplicación:**
  - Comentario de una línea:
    - //Este es un comentario de una línea

  - Comentario de múltiples líneas:
    - /*
      Este
      Es
      Un comentario
      Multilínea
      */
  - Tipos de datos:

| TIPO | DEFINICION | DESCRIPCION | EJEMPLO | OBSERVACIONES | DEFAULT |
|---|---|---|---|---|---|
| Entero | Int | Este tipo de datos aceptará solamente números enteros. | 1, 50, 100, 25552, etc. | Del -2147483648 al 2147483647 | 0 |
| Doble | Double | Admite valores numéricos con decimales. | 1.2, 50.23, 00.34, etc. | Se manejará cualquier cantidad de decimales | 0.0 |
| 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. | True |
| 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, \". | '\u0000' (carácter 0) |
| 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 las secuencias de escape: \" comilla doble \\ barra invertida \n salto de linea \r retorno de carro \t tabulación | "" (string vacío) |

- Declaración de variables:

```
<TIPO> identificador;
<TIPO> id1, id2, id3, id4;
<TIPO> identificador = <EXPRESION>;
<TIPO> id1, id2, id3, id4 = <EXPRESION>;
```

- Casteos:

```
'('<TIPO>')' <EXPRESION>
```

- Incremento y Decremento:

```
<EXPRESION>'+''+';'
<EXPRESION> '-''-';'
```

- Vectores:

```
//DECLARACION TIPO 1
<TIPO> <ID>'[' ']' = new <TIPO>'[' <EXPRESION> ']' ';'

//DECLARACION TIPO 2
<TIPO> <ID>'[' ']' = '{' <LISTAVALORES> '}' ';'
```

- Acceso a vectores:

```
<ID> '[' EXPRESION ']'
```

- Modificación de vectores:

```
<ID> '[' EXPRESION ']' = EXPRESION';'
```

- Listas:

```
'DynamicList''<' <TIPO> '>' <ID> = new 'DynamicList''<' <TIPO> '>' ';'
```

- Agregar valor a una lista:

```
'append' '('<ID>','<EXPRESION>')' ';'
```

- Acceso a listas:

```
'append' '('<ID>','<EXPRESION>')' ';'
//Ejemplo de agregar valor a una lista
DynamicList <int> lista2 =new DynamicList <int>; //creamos un lista de tipo int SE ENCUENTRA VACÍA
append(lista2,5);
append(lista2,4);
append(lista2,100);
```

- Modificación de vectores:

```
'getValue' '(' <ID> ',' EXPRESION ')'
//Ejemplo de acceso
DynamicList <int> lista2 =new DynamicList <int>; //creamos un lista de tipo int
//Agregamos valores a la lista
append (lista2,5);
append (lista2,4);
append (lista2,100);
//accesamos a un valor de la lista
int valor = getValue(lista2,1); // valor = 4
```

- Sentencias de control:

  If; if-else; switch

- Sentencias cíclicas:

  For, while, do-while

- Sentencias de transferencia:

  Break, continue, return

- Funciones:

  Tipo Id ( Parámetros | null ) { Instrucciones }

- Métodos:

  Void Id ( Parámetros | null ) { Instrucciones }

- Llamadas a funciones/métodos:

```
Id( Parámetros | null )
```

- WriteLine:

```
writeLine("Hola mundo");          //Hola mundo
```

- toLower:

```
toLower("hOlaA mUNdO");           //hola mundo
```

- toUpper:

```
toUpper ("hOlaA mUNdO");          //HOLA MUNDO
```

- Length:

```
Length("asd");                    //3
Length(vector1[ ]);               //12
Length(lista1);                   //9
```

- Truncate:

```
int nuevoValor = truncate(3.53);  //nuevoValor = 3
```

- Round:

```
Double valor = round(5.8);        //valor = 6
```

- Typeof:

```
String tipo = typeof(15);         //tipo = "int"
String tipo2 = typeof(15.25);     //tipo = "double"
String tipo3 = typeof(lista2);    //tipo3 = "lista"
```

- To String:

```
String valor2 = toString(true);   //valor = "true"
```

- toCharArray:

```
DynamicList caracteres = toCharArray("Hola");
        /*
        caracteres [[0]] = "H"
        caracteres [[1]] = "o"
        caracteres [[2]] = "l"
        caracteres [[3]] = "a"
        */
```

- Start With:

```
start with funcion1();        //Ejecuta primero el
                                método "funcion1"
```