Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Ingeniería en Ciencias y Sistemas
Lab. Organización de lenguajes y compiladores

MANUAL DE USUARIO

Justin Josue Aguirre Román
202004734

Manual de usuario

1) REQUISITOS DEL SISTEMA

Cerciórese de que la computadora satisfaga o supere los siguientes requisitosantes de instalar la aplicación “Compscript”.

CPU	Intel Celeron 800 MHz (Intel Core 2 Duo 2 GHz recomendados)
RAM	256 MB (2 GB recomendados)
Espacio disponible en disco	200 MB
Sistema operativo	Windows 7, Windows Vista o Windows XP

2) MENÚ

Una vez ejecutado el programa nos mostrara en pantalla el menú del mismo. Dándonos las diferentes opciones con que cuenta el software.

I) Funcionalidades

- Cargar archivos
- Guardar archivos
- Nuevo Archivo

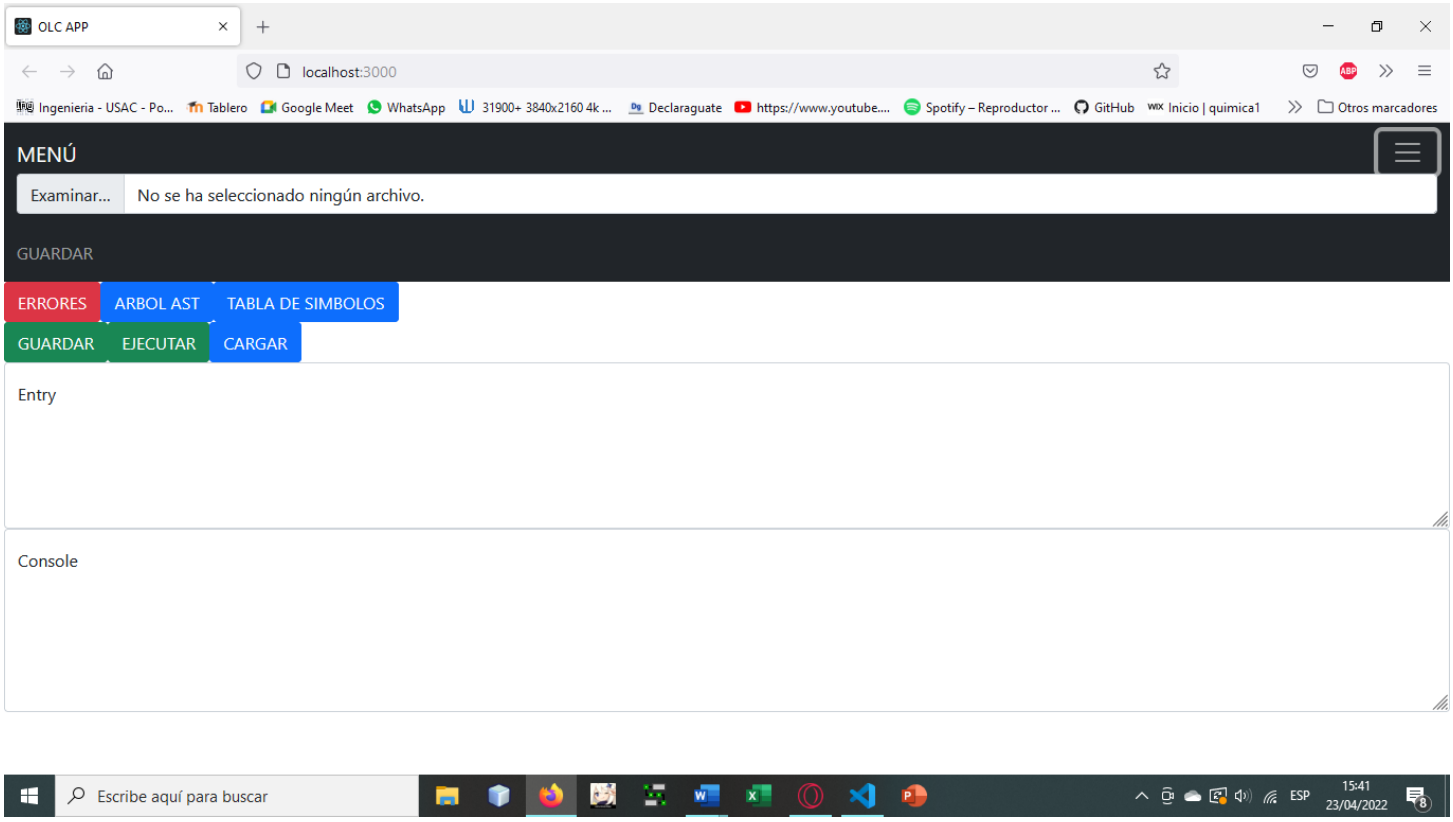
II) Herramientas

- Ejecutar
- Consola de Salida

III) Reportes

- Reporte de Errores
- Árbol AST
- Reporte tabla de símbolos

IV) Salir



I) FUNCIONALIDADES

a) Cargar Archivos: El editor deberá abrir archivos “.cst”.

b) Guardar Archivo: El editor deberá guardar el estado del archivo en el que se estará trabajando.

c) Crear archivos: El editor deberá ser capaz de crear archivos en blanco.

El archivo texto que se desea ejecutar podrá ser editado en cualquier momento de ejecución del software, para corregir errores que pueda haber con respecto a la entrada.

II) HERRAMIENTAS

a) Ejecutar: hará el llamado al intérprete, el cual se hará cargo de realizar los análisis léxico, sintáctico y semántico, además de ejecutar todas las sentencias.
Cabe recalcar que durante este proceso el software hará una lectura de la entrada indicada, mensaje que tiene que seguir ciertos parámetros para cada uno de las instrucciones que se deseen.

Las semánticas y estructuras del mensaje son:

COMENTARIO: 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.

Estructura:

```
// Este es un comentario de una línea
/* Este es un comentario
Multilínea Para este lenguaje */
```

DECLARACION: Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador.

Estructura:

<Tipo> id;

<Tipo> id1, id2, id3, id4;

<Tipo> identificador = **<Expresión>**;

<Tipo> id1, id2, id3, id4 = **<Expresión>**;

ASIGNACION: Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador.

Estructura:

identificador = **<Expresión>**;

id1, id2, id3, id4 = **<Expresión>**;

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.

Estructura:

(**<Tipo>**) **<Expresión>**;

INCREMENTO: si incrementamos una variable, se incrementará de uno en uno su valor.

DECREMENTO: si decrementamos una variable, se disminuirá de uno en uno su valor.

Estructura:

<Expresion>’+’’+’’;

<Expresion>’ ‘-‘‘-’’;’

VECTORES: Los vectores 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.

Estructura:

DECLARACION TIPO 1 (una dimensión)

<Tipo> ‘Id’ ‘[’ ‘]’ = new **<Tipo>** ’[’ **<Expresion>** ’]’ ‘;’

<Tipo> ‘Id’ ‘[’‘]’‘[’‘]’= new **<Tipo>**’[’ **<Expresion>** ’]’’[’**<Expresion>**’]’ ‘;’

DECLARACION TIPO 2

<Tipo> ‘Id’ ‘[’ ‘]’ = ’[’ **<LISTAVALORES>** ’]’ ‘;’

VECTORES: La sentencia if ejecuta las instrucciones sólo si se cumple una condición. Si la condición es falsa, se omiten las sentencias dentro de la sentencia.

Estructura:

‘if’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Instrucciones>** ‘}’

| ‘if’ ‘(’**<Expresion>**‘)’‘{‘**<Instrucciones>**‘}’‘else’‘{‘ **<Instrucciones>** ‘}’

| ‘if’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Instrucciones>** ‘}’ ‘else’ **<IF>**

SWITCH: Estructura principal del switch, donde se indica la expresión a evaluar.

Estructura:

‘switch’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<CaseList> <Default>** ‘}’

| ‘switch’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<CaseList>** ‘}’

| ‘switch’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Default>** ‘}’

CASE: Estructura que contiene las diversas opciones a evaluar con la expresión establecida en el switch.

Estructura:

‘case’ **<Expresion>** ‘:’ **<Instrucciones>**

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

Estructura:

‘while’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Instrucciones>** ‘}’

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.

Estructura:

‘for‘(’<Declaracion>|<Asignacion>’;’<Condicion>’;’<Actualizacion>‘)’‘{‘<Instruccion> ‘}’

DO_WHILE: El ciclo o bucle Do-While, es una sentencia que ejecuta al menos una vez el conjunto de instrucciones que se encuentran dentro de ella y que se sigue ejecutando mientras la condición sea verdadera

Estructura:

‘do’ ‘{‘ <Instrucciones> ‘}’ ‘while’ ‘(’ <Expresion> ‘)’ ‘;’

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.

Estructura:

‘break**’ ;**

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.

Estructura:

‘continue**’ ;**

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.

Estructura:

‘return**’ ;**

| **‘**return**’ <Expresion> ;**

FUNCIONES: Una función es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros. Para las funciones es obligatorio que las mismas posean un valor de retorno que coincida con el tipo con el que se declaró la función.

Estructura:

'Id' '(' **<Parametros>** ')' ':' <Tipo> '{' **<Instrucciones>** '}'

Parametros -> **<Parametros>** , **<Tipo>** 'id'

| **<Tipo>** 'id'

METODOS: Un método también es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros, aunque a diferencia de las funciones estas subrutinas no deben de retornar un valor.

Estructura:

'Id' '(' **<Parametros>** ')' ':' 'void' '{' **<Instrucciones>** '}'

Parametros -> **<Parametros>** , **<Tipo>** 'id'

| **<Tipo>** 'id'

LLAMADA: La llamada a una función específica la relación entre los parámetros reales y los formales y ejecuta la función. Los parámetros se asocian normalmente por posición, aunque, opcionalmente, también se pueden asociar por nombre. Si la función tiene parámetros formales por omisión, no es necesario asociarles un parámetro real.

Estructura:

LLAMADA ->'Id' '(' **<Parametros>** ')'

|'Id' '(' ')'

Parametros -> **<Parametros>** , 'id'

| 'id'

PRINT: Esta función nos permite imprimir expresiones con valores únicamente de tipo entero, doble, booleano, cadena y carácter. Esta función no concatena un salto de línea al final del contenido que recibe como parámetro.

Estructura:

'Print' '(' <Expresion>')';

PRINTLN: Esta función nos permite imprimir expresiones con valores únicamente de tipo entero,doble, booleano, cadena y carácter. Esta función concatena un salto de línea al finalizar el contenido que recibe como parámetro.

Estructura:

‘Println’ ‘(‘ <Expresion>‘)’;

TO_LOWER: Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras minúsculas.

Estructura:

‘toLower’ ‘(‘ <Expresion>‘)’;

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

Estructura:

‘toUpper’ ‘(‘ <Expresion>‘)’;

ROUND: Esta función recibe como parámetro un valor numérico.

Estructura:

‘round ‘(‘ <Valor>‘)’;

LENGTH: Esta función recibe como parámetro un vector, una lista o una cadena y devuelve el tamaño deeste.

Estructura:

‘length ‘(‘ <Valor>‘)’;

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

Estructura:

‘typeof ‘(‘ <Valor>‘)’;

TO_STRING: Esta función permite convertir un valor de tipo numérico o booleano en texto.

Estructura:

‘toString ‘(‘ <Valor>‘)’;

TO_CHAR_ARRAY: Esta función permite convertir una cadena en un vector de caracteres

Estructura:

‘toCharArray ‘(‘ <Valor>‘)’;

RUN: Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia RUN para indicar que método o función es la que iniciará con la lógica del programa.

Estructura:

‘run ‘(’ ’)’ ‘;’

‘run’ ‘(’ <LISTAVALORES> ’)’ ‘;’

LISTAVALORES-> LISTAVALORES ‘,’ EXPRESION

| EXPRESION

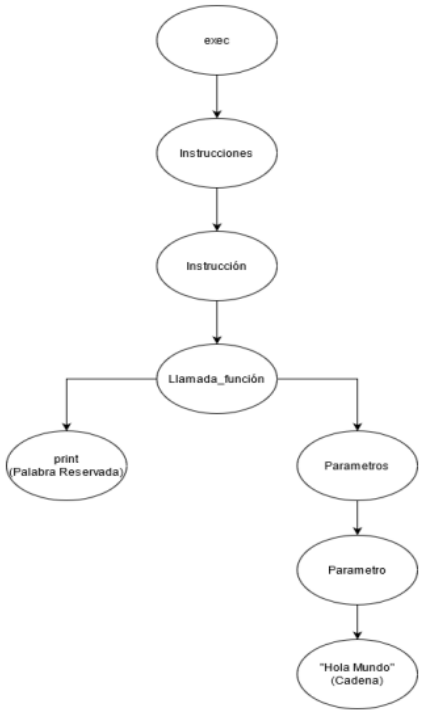
b) Consola de Salida: En esta área se mostrarán los resultados, mensajes y todo lo que sea indicado dentro del lenguaje.

III) REPORTES

a) **Reporte de Errores**: Se mostrarán todos los errores encontrados al realizar el análisis léxico, sintáctico y semántico.

#	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

b) **Generar Árbol AST** (Árbol de Análisis Sintáctico): se debe generar una imagen del árbol de análisis sintáctico que se genera al realizar los análisis.



c) **Reporte de Tabla de Símbolos**: Se mostrarán todas las variables, métodos y funciones que han sido declarados dentro del flujo del programa.

Identificador	Tipo	Tipo	Entorno	Línea	Columna
Factor1	Variable	Entero	Función multiplicar	15	4
Factor2	Variable	Decimal	Función multiplicar	16	7
Resultado	Variable	Decimal	Función multiplicar	17	7
MostrarMensaje	Método	Void	-	50	6

IV) SALIR

Detiene la ejecución del software.