

Manual De Usuario (TAC_Assembly_Generator)

Antes de entrar a uso y funcionamiento de la aplicación, si tienes algún problema con la instalación y ejecución del programa puedes pasar por el “Manual De Instalación y Desinstalación (TAC_Assembly_Generator)”.

Al iniciar la aplicación nos aparece primero la pantalla principal, la cual contiene varios elementos con distintas funcionalidades que se explican posteriormente. Los elementos de la barra superior son muy básicos, aquí se encuentra de primero File que contiene:

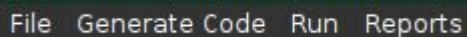
New: Despliega la pestaña denominada Programa, que contendrá exclusivamente el código fuente del programa a compilar.

Open: Esta opción permite abrir un archivo .mlg, este contiene el código fuente de un programa almacenado con anterioridad. El contenido de este archivo será mostrado en la pestaña denominada Programa. Esta opción es únicamente para los archivos con extensión .mlg.

Save: Guarda el contenido de la pestaña activa en ese momento, en la dirección que el usuario desee.

Save As: Permite guardar el contenido de la pestaña activa en ese momento, con diferente nombre pero con la extensión que le corresponda.

Exit: Finaliza la ejecución de la aplicación.



File Generate Code Run Reports

Luego el Generate Code en donde esta la opción de ejecutar el archivo de entrada, esto solo se puede realizar si posteriormente se abrió un archivo mlg o se hizo un nuevo archivo.

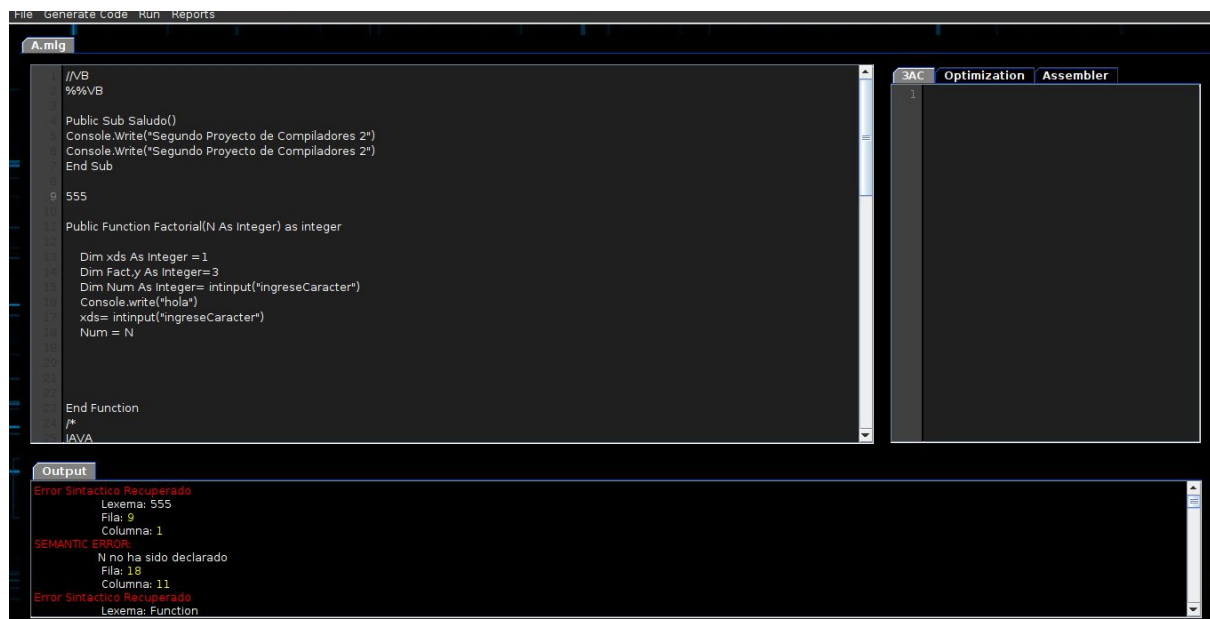
También está la pestaña de Run en donde se encuentran:

3AC: Ejecuta el código tres direcciones generado.

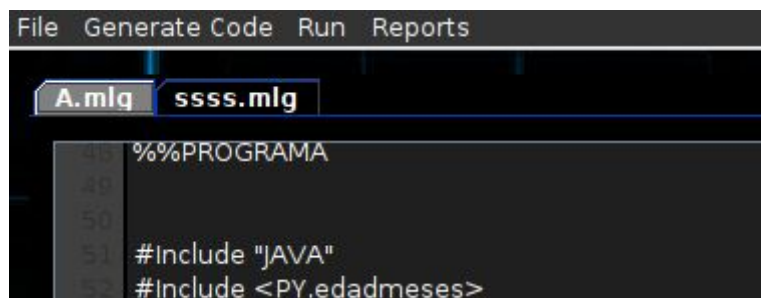
Optimization: Ejecuta el código tres direcciones optimizado.

Assembler: Ejecuta el código assembler generado.

Al momento de abrir o crear un archivo aparecerán varias áreas de texto para poder empezar a trabajar en ella como se muestra en la siguiente imagen.

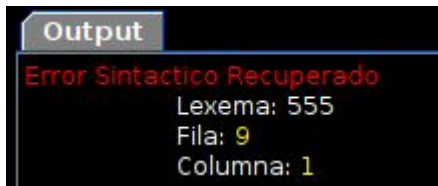


La primera que se encuentra en la parte superior izquierda es el área en donde se escribirá el código de entrada para el TAC, las pestañas de la parte derecha es la salida en donde se encuentra todo el código generado por la aplicación la primera es 3AC que representa el código de tres direcciones, en la parte inferior se mostrarán errores al momento de compilar. Debe tomar en cuenta que no solo puede abrir un archivo a la vez, sino que múltiples y eso se podrá, navegando por las pestañas de la parte superior izquierda.



Cada archivo tendrá su Output, 3AC, Optimization, Assembler y su textarea principal para el mlg. Todas las generaciones y ejecución que se deseen realizar, serán realizadas al archivo que se tiene seleccionado al momento de querer realizar la acción.

Estas últimas dos áreas cuentan con colores de texto para identificar de forma mas facil la información presentada por ejemplo el de Output:



Rojo : Tipo de error o información;

Amarillo: Numero de fila y columna;

Blanco: Información normal;

El TAC:

```
// Variables globales
t16=t14*t15
Arreglo1[ t16]
t14=1
t15=4
t17='y'
y=t17
void C_main(){
t22=0
t23=3
t24=6
t25=t14
t26=t22*t14
t27=t27+t23
t28=Arreglo1[t27]
t32=4
x=t32
etv10:
t33=x
t34=2
If t33 > t34 goto etv7
goto etv8
etv7:
```

Gris: Comentarios;

Amarillo: Funciones y Clases;

Azul: Etiquetas de Bloques;

Blanco: Código Normal;

Archivo de Entrada:

El código mlg debe estar estructurado de la siguiente manera:

```
%%VB
    Módulos en Visual Basic
%%JAVA
    Clases en Java
%%PY
    Funciones y Procedimientos en Python
%%PROGRAMA
Sección de librerías de C
#include "VB"           //incluye el código en VB
```

```
#include "JAVA.*"           //incluye todas las clases declaradas en
                             //la sección de Java
#include "JAVA.nombre"      //incluye una clase en especifica
                             //declarada en la sección de Java
#include "PY"                //incluye el código en Python
Sección declaración de Constantes
Sección de Variables Globales

void main()
{
    // Programa principal
}
```

Las especificaciones para los tres lenguajes superiores se encuentran en el archivo de Componentes Lenguajes que se encuentra ubicado en el archivo de Documentación, todo esto aparecerá de la siguiente manera, en donde se especifica la sintaxis de cada lenguaje.

Codigo	Visual Basic	Java	Python
Tipos:	Integer Decimal Char	int float char	
Solicitud de Datos a Usuarios	intinput floatinput charinput	intinput floatinput charinput	intinput floatinput charinput
Expresiones Aritmeticas	+ Suma - Resta o sustracción * Multiplicación / División \ División entera o parte entera del cociente de una división Mod Residuo de la división entera ^ Potencia o exponenciación	+ Suma - Resta o sustracción * Multiplicación / División % Residuo de la división entera	+ Suma - Resta o sustracción * Multiplicación / División //Cociente % Residuo de la división entera **Potencia
Mensajes a Pantalla	Console.WriteLine(<Mensaje que va a tener>) Console.Write(<Mensaje que va a tener>)	System.out.println("<...>"); System.out.print("<...>");	print(<id> string [<id> string]*)
Ciclos	For VAR = VI To VF ... Next VAR while [condición] ... end while Do While [condición] Loop do ... Loop While[condición]	for(<tipo> <var> = <inicial>; <var> <Comparacion> valor final; <incremento>) { ... } while([condición]){ ... } do{ ... }while([condición]);	for [id] in range([int] [, int]?): TAB ... [else]? for [id] in [arreglo]: TAB ... [else]? while [condición]: Tab ...

Para el principal será similar, pero con otros componentes:

Tipos de datos:

- int
- char
- float

Arreglos:

Arreglos de N dimensiones de cualquier tamaño. Los arreglos pueden ser de cualquier tipo.

Tipo arreglo[dim1];

Tipo arreglo[dim1][dim2][dim3];

Al momento de usar un arreglo, dentro de las dimensiones podrán venir expresiones aritméticas, arreglos, funciones, constantes, etc. Ejemplo:

```
int arreglo[25+4][CONSTANTE_ENTERA];
x = vector[5*4/7+8*9(4+2)][matriz[1][2*7]];
```

Constantes:

Definición de constantes:

```
const tipo nombre_constante = valor;
```

Operadores:

Todos los operadores aritméticos, números enteros y reales ambos pueden ser negativos, los operadores aritméticos pueden ser: +, -, *, /, % (modulo ó residuo), = asignación y paréntesis.

Comentarios:

Los comentarios de línea o de bloque que se hagan deberán de ser pasados a código tres direcciones y al código ensamblador.

//Comentario de línea

/*Comentario
de bloque*/

If:

En la instrucción If, dentro de la condición pueden venir expresiones aritméticas y relacionales (<, >, =, !=), esto implica operadores && (and), || (or), !(not); **con o sin** Else.

```
if ( condición )
{
    SENTENCIAS;
}
else
{
    SENTENCIAS;
}
```

Switch:

La instrucción Switch, con *n* case y **con o sin** Default.

```
switch (variable)
{
    case valor1:
        SENTENCIAS;
        break;
    case valor2:
        SENTENCIAS;
        break;
    ....
    case valor2:
        SENTENCIAS;
        break;
    default:
        SENTENCIAS;
        break;
}
```

For:

```
for ( variable = valor_inicial; condición; variable = variable + valor_de_aumento )  
{  
    SENTENCIAS;  
}
```

While:

```
while ( condición )  
{  
    SENTENCIAS;  
}
```

Do While:

```
do  
{  
    SENTENCIAS;  
}  
while ( condición );
```

Llamadas a funciones y procedimientos:

```
Variable = VB.nombre_funcion(parámetros);  
Variable = PY.nombre_funcion(parámetro);  
VB.nombre_procedimiento(parámetros);  
PY.nombre_procedimiento(parámetros);
```

Scanf:

Esta instrucción permitirá asignar valores a las variables.

```
scanf(" mascara ", &variable);
```

Printf:

Esta instrucción permitirá desplegar mensajes y los valores de las variables

```
printf(" texto ");      ó      printf("El valor es mascara ", var);
```

Clrscr:

limpiar la pantalla, ejemplo:

```
clrscr();  
Getch
```

Esta instrucción leerá una tecla del teclado para poder seguir la ejecución del programa. El valor que regresa **puede o no** ser asignado a una variable tipo **char** o **int**, si es asignado a una variable tipo char la variable tendrá el carácter que se presionó; si es asignado a una variable tipo int la variable tendrá el valor ASCII del carácter que se presionó.