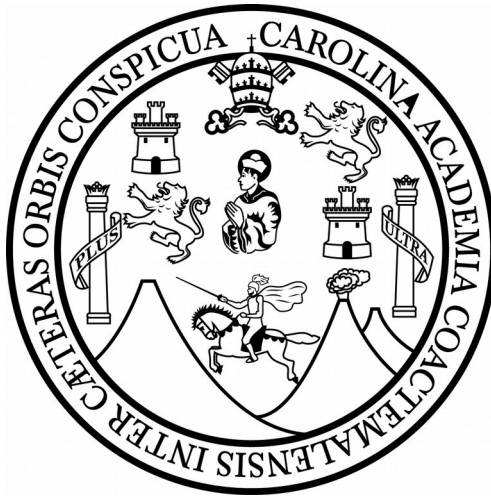


**CENTRO UNIVERSITARIO DE OCCIDENTE**

**DIVISIÓN CIENCIAS DE LA INGENIERÍA**

**CARRERA DE INGENIERÍA CIENCIAS Y SISTEMAS**



**LABORATORIO DE ORGANIZACIÓN LENGUAJES Y COMPILADORES 2**

**“SEXTO SEMESTRE”**

**ING.: JOSÉ MOISÉS GRANADOS GUEVARA**

**AUX: EDVIN TEODORO GONZALEZ RAFAEL**

**ESTUDIANTE:** Bryan René Gómez Gómez – 201730919

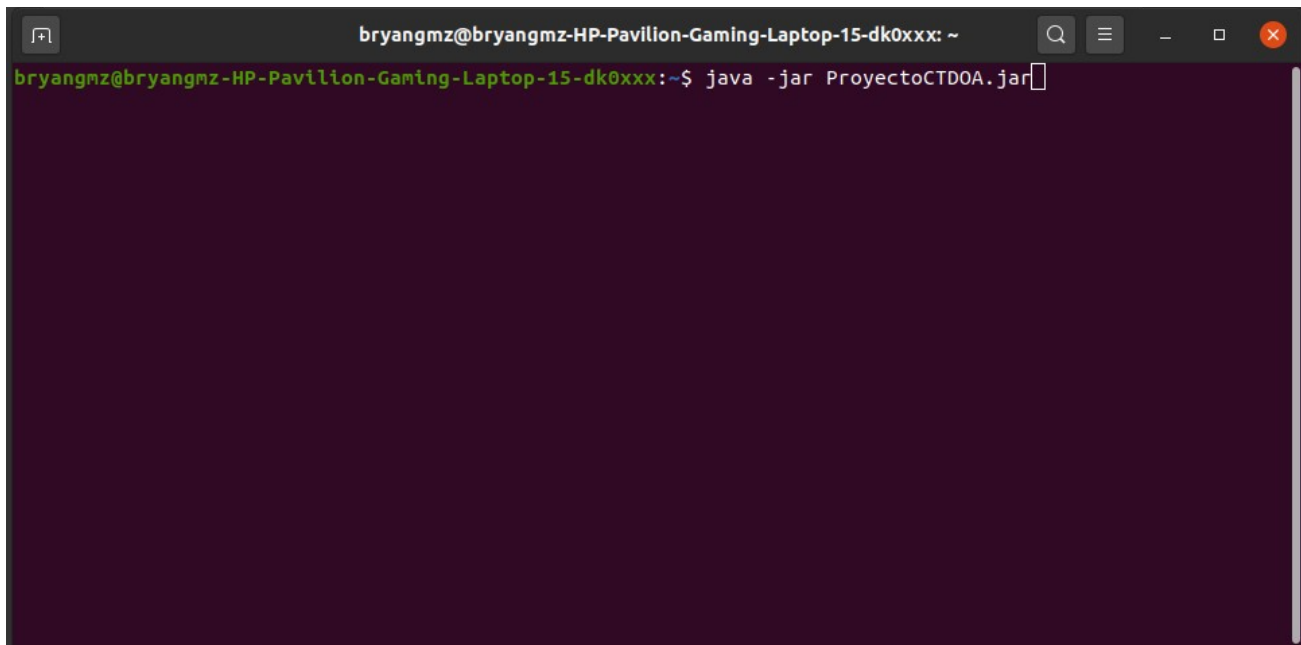
**PROYECTO:** “Manual Usuario – Segundo Proyecto, Fase - 1”

**FECHA:** 16 de Octubre de 2,020

# COMPILADOR - CTDOA - GUATEMALA

La aplicación consiste en un Compilador Multilenguaje en Java, este genera código intermedio el cual este es código tres direcciones con sintaxis de C.

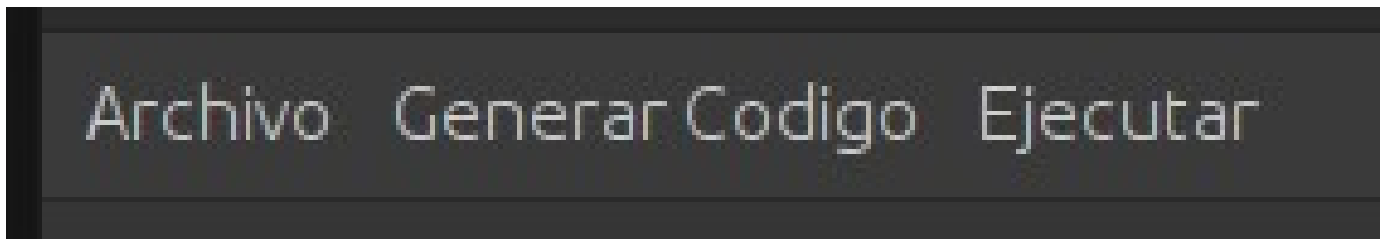
Para iniciar la aplicación debes de tener instalado JDK 8: Debes de abrir la terminal en donde se encuentra el .jar ejecutable. Debes de ingresar el siguiente comando: **\$java -jar ProyectoCTDOA.jar**

A screenshot of a terminal window with a dark background. The title bar at the top reads 'bryangmz@bryangmz-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~'. The terminal shows the command 'bryangmz@bryangmz-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~\$ java -jar ProyectoCTDOA.jar' with a cursor at the end of the line. The rest of the terminal is empty.

## Interfaz Grafica

### Menús

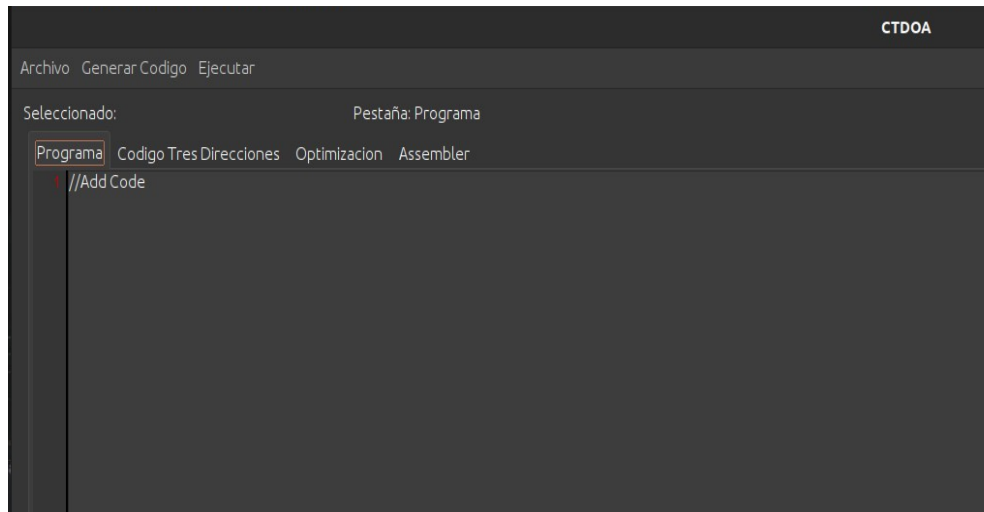
La aplicación fue realizada en Java, esta contiene los siguientes menús:



## Archivo

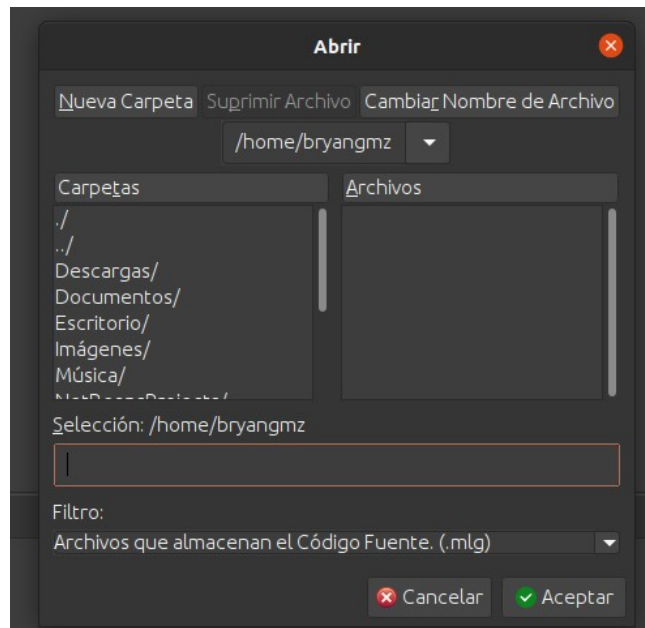
### Nuevo

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



### Abrir

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



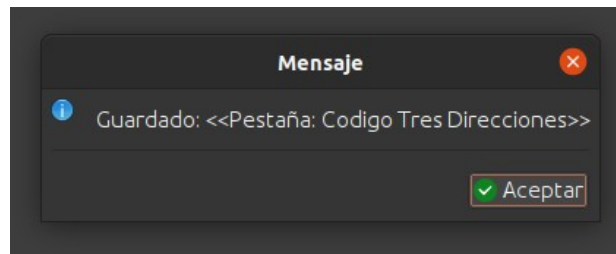
## Guardar

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



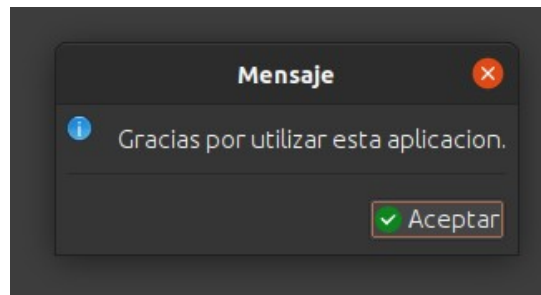
## Guardar Como

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



## Salir

Finaliza la ejecución de la aplicación.



## NOTA:

Los archivos .mlg son los archivos que contienen el código fuente del programa a compilar. Entiéndase los archivos de entrada para su compilador.

Las extensiones que se manejarán para los archivos que se almacenen en Código en Tres direcciones y Código Optimizado serán .cpp y para el Código en Assembler tendrá que ser .asm

## **Generar código**

### **Código en Tres Direcciones**

Esta opción genera el código en tres direcciones del programa que se encuentra en la pestaña denominada Programa y este resultado deberá ser mostrado en la pestaña Código Tres Direcciones.

### **Código Optimizado (En Proceso)**

Esta opción genera el código optimizado en 3 direcciones del programa, que se encuentra en la pestaña denominada Código en Tres Direcciones y será mostrado en la pestaña Optimización. (El método que se utilizará para optimizar el código es por Mirilla).

### **Código Assembler (En Proceso)**

Esta opción genera el código Assembler del programa, que se encuentra en la pestaña de Optimización, y se mostrará el código generado en la pestaña Assembler

### **NOTA:**

Para generar el código deseado, debe verificarse que el código base para realizarse se encuentre en su pestaña correspondiente. Si el código indicado no se encuentra en su pestaña correspondiente, entonces automáticamente debe generar el código anterior para poder mostrar el código que se desea. Obviamente para esto debe existir el archivo .mlg que se encuentra en la pestaña Programa de lo contrario deberá mostrar un mensaje que no se encuentra el código fuente.

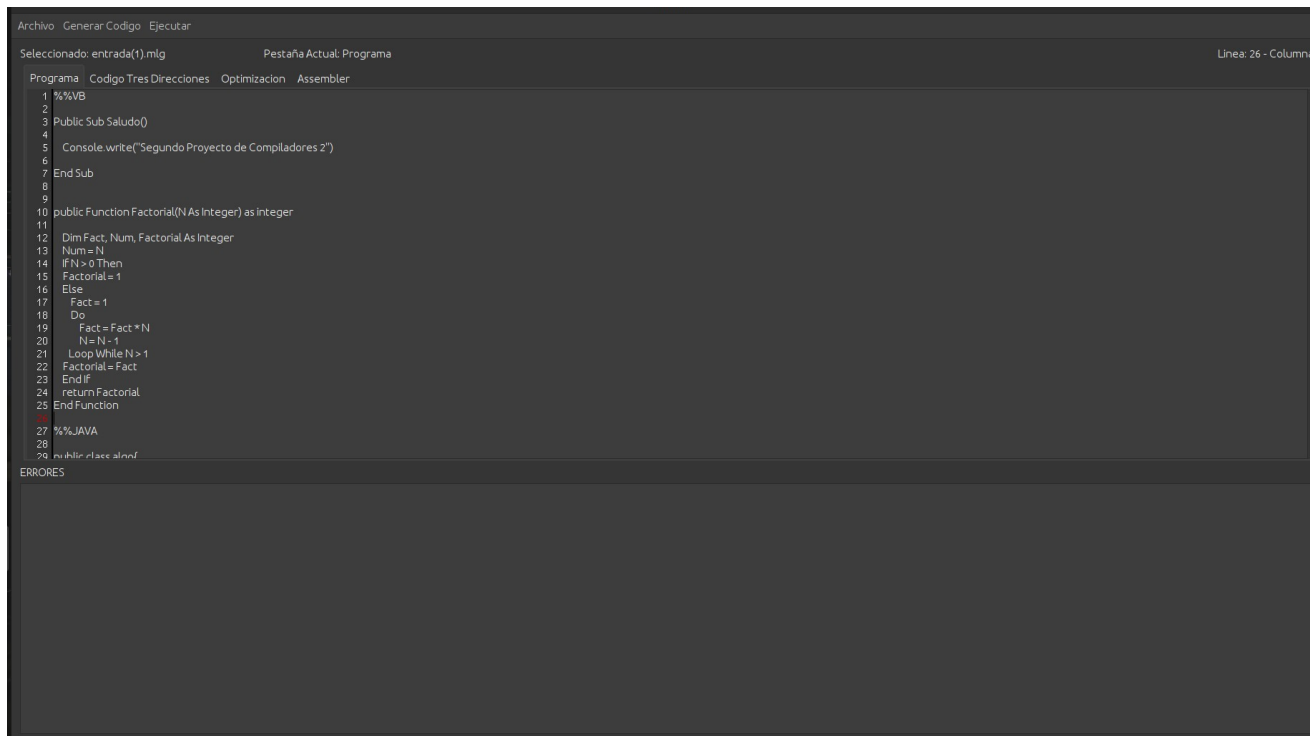
El código generado para código en tres direcciones, optimización y assembler deberán ser mostrados en su respectiva pestaña con el formato necesario de un programa cpp o asm.

Los errores léxicos, sintácticos y semánticos deberán ser mostrado en la sección de Errores, recuerde que estos errores solo podrán estar presente únicamente en el archivo .mlg. El formato para mostrar los errores es:

**Fila - columna - tipo de error - quien lo produjo - breve descripción**

## Ejecutar

Este menú permite ejecutar los diferentes códigos generados.

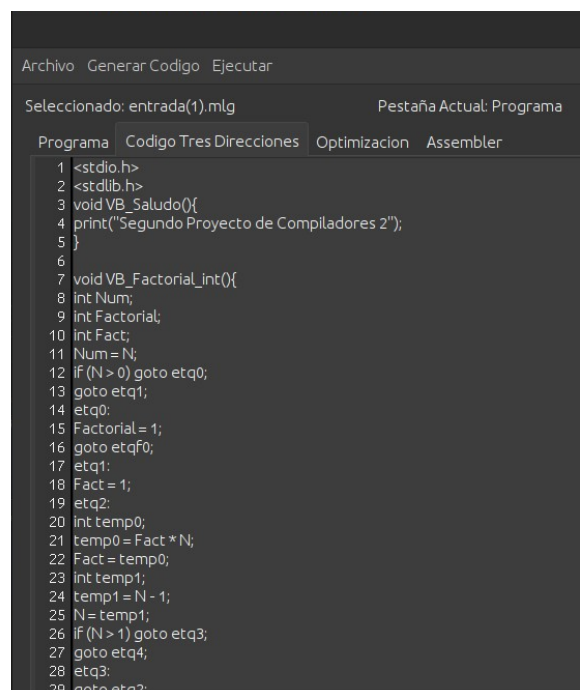


```
Archivo  Generar Codigo  Ejecutar
Seleccionado: entrada(1).mlg  Pestaña Actual: Programa  Línea: 26 - Columna: 1
Programa  Codigo Tres Direcciones  Optimizacion  Assembler
1  %%VB
2
3  Public Sub Saludo()
4
5      Console.WriteLine("Segundo Proyecto de Compiladores 2")
6
7  End Sub
8
9
10 public Function Factorial(N As Integer) as Integer
11
12     Dim Fact, Num, Factorial As Integer
13     Num = N
14     IF N > 0 Then
15         Factorial = 1
16     Else
17         Fact = 1
18     Do
19         Fact = Fact * N
20         N = N - 1
21     Loop While N > 1
22     Factorial = Fact
23     End If
24     return Factorial
25 End Function
26
27 %%JAVA
28
29 public class main {
```

ERRORES

## Código en Tres Direcciones

Ejecuta el código 3 direcciones generado.



```
Archivo  Generar Codigo  Ejecutar
Seleccionado: entrada(1).mlg  Pestaña Actual: Programa
Programa  Codigo Tres Direcciones  Optimizacion  Assembler
1  <stdio.h>
2  <stdlib.h>
3  void VB_Saludo(){
4  print("Segundo Proyecto de Compiladores 2");
5  }
6
7  void VB_Factorial_int(){
8  int Num;
9  int Factorial;
10 int Fact;
11 Num = N;
12 if (N > 0) goto etq0;
13 goto etq1;
14 etq0:
15 Factorial = 1;
16 goto etqf0;
17 etq1:
18 Fact = 1;
19 etq2:
20 int temp0;
21 temp0 = Fact * N;
22 Fact = temp0;
23 int temp1;
24 temp1 = N - 1;
25 N = temp1;
26 if (N > 1) goto etq3;
27 goto etq4;
28 etq3:
29 goto etq2;
```

### **Código Optimizado (En Proceso)**

Ejecuta el código 3 direcciones optimizado.

### **Código Assembler (En Proceso)**

Ejecuta el código assembler generado.

### **NOTA:**

Para ejecutar cada código se debe usar el compilador correspondiente, realizando una llamada desde su aplicación.

## **Sintaxis de los diferentes lenguajes:**

### **VISUAL BASIC**

**NOTA:** Visual no es case sensible

*/\* Tipo de Datos \*/*

**Integer**

**Decimal**

**Char**

*/\* Operadores - Expresiones Aritméticas \*/*

Aritméticos    ^, -, \*, /, \, Mod, +, =

Asignación    =, ^=, \*=, /=, \=, +=, -=

Comparación   =, <>, <, >, <=, >=, Like, Is

Lógicas        Not, And, Or, Xor, AndAlso, OrElse

Concatenar    +, &

*/\* Mensajes en Pantalla \*/*

Console.WriteLine()

Console.Write()

*/\* Solicitud de Datos al usuario \*/*

Dim variable As Tipo\_Dato

*/\*\* While*

While <Condicion>

    [ Declaraciones ]

End While

*/\*\* Do While*

Do

    [ declaraciones ]

Loop < While> condition

*/\*\* For*

For <Identificador | Identificador = Asignacion > [ As tipoDato ] = start To end [ Step step ]

    [ declaracion ]

Next <Identificador>

*/\*\* If - else*

If <Condicion> Then(Optional)

    [ declaraciones ]

[ ElseIf elseifcondition [ Then ]

    [ declaraciones ] ]

[ Else

    [ declaraciones ] ]

End If

' Sintaxis de una linea es obligatorio el then :

If condition Then (Obligatorio) [ declaraciones ] [ Else [ declaraciones ] ]



/\*\* Switch

Select [ Case ] testexpression

    [ Case expressionlist

        [ statements ] ]

    [ Case Else

        [ elstatements ] ]

End Select

/\*\* Asignacion de Valores

<Identificador> = <valor>

<Identificador> = <Identificador>

/\* Funciones \*/

Function <nombre> [ (parameterlist) ] [ As return type ]

    [ statements ]

End Function

## **Notas**

La instruccion 'Return' asigna simultáneamente el valor devuelto y sale de la función.

/\* Procesos \*/

Sub <nombre> [ (parameter list) ]

    [ statements ]

End Sub

# JAVA

/\* Operadores - Expresiones Aritméticas \*/

Aritméticos    +, -, \*, /, %

Asignación    =, +=, \*=, -=, /=

Comparación    ==, !=, <, >, <=, >=

Lógicas        &&, ||

Concatenar    +

/\* Mensajes en Pantalla \*/

System.out.println()

System.out.print()

/\* Solicitud de Datos al usuario \*/

int <Identificador> = intinput()

float <Identificador> = floatinput()

char <Identificador> = charinput()

<Identificador> = intinput()

<Identificador> = floatinput()

<Identificador> = charinput()

/\*\* While

while ( <Condicion> ) {

    [ Declaraciones ]

}

/\*\* Do While

do {

    [ declaraciones ]

} while ( <Condicion> );

/\*\* For

for(int i = valor inicial; i <= valor final; i = i + paso) {

for(int i = valor inicial; i <= valor final; i = i++) {

    Bloque de Instrucciones....

}

int i;

for(i = valor inicial; i <= valor final; i = i + paso) {

for(i = valor inicial; i <= valor final; i = i++) {

    Bloque de Instrucciones....

}

/\*\*If - else

if ( <Condicion> ) {

    [ declaraciones ]

} elseIf ( <Condicion> ) {

    [ declaraciones ]

else {

    [ declaraciones ]

}

/\*\* Switch

switch ( <Caso> ) {

case <caso>:

[ declaracion ]

break; //opcional

return; //opcional

default:

[ declaracion ]

break; //opcional

return; //opcional

}

/\* Declaraciones de Variables \*/

<TipoDato> <Identificador>;

<TipoDato> <Identificador> = <Valor>;

/\* Asignacion de Valores \*/

<Identificador> = <Valor>

<Identificador> = <Identificador>

/\* Metodos \*/

public void <Identificador> ( <Parametros> ) {

[ Bloque de Instrucciones ]

}

*/\* Funciones \*/*

```
public <TipoDato> <Identificador> ( <Parametros> ) {  
    [ Bloque de Instrucciones ]  
}
```

*/\* Declaracion de Clases \*/*

```
public class <Identificador> {  
    [ Instrucciones ]  
}
```

```
public class <Identificador> {  
    [ Instrucciones ]  
}
```

## Python

**NOTA:** No hay tipo de dato

*/\* Operadores - Expresiones Aritméticas \*/*

Aritméticos    + −, \*, /, //, %, \*\*

Asignación    =, +=, -=, \*=, /=, //=, \*\*=, &=

Comparación   <> Diferente, ==, != Distinto, <, >, <=, >=

Lógicas        and, or, not

Concatenar    +, ,

```
/* Mensajes en Pantalla */
```

```
print("<Cadena>" + "Cadenas")  
print('Cadena' + 'Cadena') ...  
print("<Cadena>", "Cadenas")...  
print('Cadena', 'Cadena') ...
```

```
/* Solicitud de Datos al usuario */
```

```
<Identificador> = intinput()  
<Identificador> = floatinput()  
<Identificador> = charinput()
```

```
/**While por Conteo
```

```
while < Condicional >:  
    < Instrucciones >
```

```
/** For
```

```
/* Range podría ser: (Parámetro) || (Parámetro, Parámetro) || (Parámetro, Parámetro, Parámetro)  
for <var> in range(<param>, <param>, <param>):  
    < Instrucciones >
```

```
/** If - else
```

```
if <Condicion>:  
    [ declaraciones ]  
[ elif <Condicion>:  
    [ declaraciones ] ]  
[ else:  
    [ declaraciones ] ]
```

/\* Funciones - Métodos \*/

def NOMBRE(LISTA\_DE\_PARÁMETROS):

< SENTENCIAS >

RETURN [EXPRESION] (Opcional)

## Estructura archivo .mlg

%%VB

*Módulos en Visual Basic*

%%JAVA

*Clases en Java*

%%PY

*Funciones y Procedimientos en Python*

%%PROGRAMA

**Sección de librerías de C**

*#include "VB"*

**Sección declaración de Constantes**

**Sección de Variables Globales**

void main() {

*// Programa principal*

}

Los lenguajes que se utilizarán para conformar la sección de librerías son:

- Visual Basic
- Java
- Python

Mientras que el programa principal utiliza C, para cada sección del programa se deberá respetar la sintaxis de cada lenguaje de programación.

En los lenguajes VB, Java y Python se utilizarán de forma limitada, únicamente:

- Expresiones aritméticas + - \* / %
- mensajes a pantalla
- solicitud de datos a usuarios: `intinput`, `floatinput`, `charinput`

- ciclos (for i, while, do while)
- manejo de condiciones (if, switch)
- declaración de variables
- asignaciones variables
- Funciones
- Procedimientos
- Clases (Java)

Además los únicos tipos que se manejarán en estos lenguajes son entero, real y caracter.

### **NOTA:**

Las declaraciones de funciones, métodos en Visual Basic y métodos en Java se manejarán de forma pública ( PUBLIC).

El programa principal que se maneja en sintaxis de C se especifica a continuación:

Especificaciones del programa principal

### **Tipos de datos**

Los tipos de datos que se utilizarán son:

- int
- char
- float

**NOTA:** La variable puede o no ser iniciada al momento de definirla.

### **Arreglos**

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

Tipo    arreglo[dim1];

Tipo    arreglo[dim1][dim2][dim3];

**NOTA:** Al momento de declarar un arreglo, las dimensiones pueden ser expresadas usando literales enteras, constantes o expresiones aritméticas sobre literales y constantes.

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;
```

```
const float pi = 3.14159;
const char c = 'X';          // X es una constante tipo char
const int X = 10;            // X es un tipo int
```

## Operadores

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

Ejemplo

```
var = ( 1 + 2 - 3 * 4 / 5 ) % (5 * -3);
```

## 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*/
```

## Sentencia 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;
}
```

Donde **condición** puede ser cualquier condición que involucre operadores aritméticos, relacionales y lógicos, y cualquier tipo de variable. Si el valor resultante de la condición es mayor que 0, la condición

es VERDADERA, si el resultado de la condición es menor o igual a 0 la condición es FALSA.  
Ejemplo:

```
(Var + 1 > var2 * 2 && var3 != var4 )
```

### **Sentencia switch**

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

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

### **Ciclo for**

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

**Valor\_inicial** y **valor\_de\_aumento** será un número entero o una variable tipo entero, **condición** puede ser cualquier condición, refiérase a la parte **condición** de la instrucción if

### **Ciclo while**

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

**Condición** puede ser cualquier condición, refiérase a la parte condición de la instrucción **if**

Ciclo do while

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

**Condición** puede ser cualquier condición, refiérase a la parte condición de la instrucción **if**.

Llamadas a funciones y procedimientos

Variable = VB.nombre\_funcion(parámetros);

Variable = PY.nombre\_funcion(parámetro);

VB.nombre\_procedimiento(parámetros);

PY.nombre\_procedimiento(parámetros);

#### NOTA:

Para funciones y procedimientos, el paso de parámetros es por valor.

#### **scanf**

Esta instrucción permitirá asignar valores a las variables.

scanf(“ *mascara* “, &variable);

*Mascara* se refiere a texto y al indicador de que tipo de dato leerá, ejemplo:

scanf(“Valor %d”, &var); //Leerá del teclado un valor para asignar a variable tipo int

scanf(“Valor %c”, &var); //Leerá del teclado un valor para asignar a variable tipo char

scanf(“Valor %f”, &var); //Leerá del teclado un valor para asignar a variable tipo float

#### **printf**

Esta instrucción permitirá desplegar mensajes y los valores de las variables

printf(“texto”);            ó            printf (“El valor es *mascara* “, var);

En la primera instrucción solo se mostrará *texto*, en la segunda se mostrará el texto y el valor que contenga la variable *var*, puede haber más de una variable por mostrar.

*Mascara* se refiere a texto y al indicador de que tipo de dato desplegará, ejemplo:

printf(“Valor %d”, var); //Desplegará el valor de una variable tipo int

printf (“Valor %c”, var); //Desplegará el valor de una variable tipo char

printf (“Valor %f”, var); //Desplegará el valor de una variable tipo float

#### **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.

## Clases

Las clases se manejaran de la siguiente manera

### *Declaración*

```
JAVA.Nombre_Clase      Nombre_Var1, Nombre_Var2;  
JAVA.Nombre_Clase      Nombre_Var1(parámetros);      //llamando a constructor
```

### *Llamada a métodos*

```
Variable = JAVA.nombre_objeto.método(parámetros);  
JAVA.nombre_objeto.método(parámetros);
```

## REPORTES:

### Sintaxis:

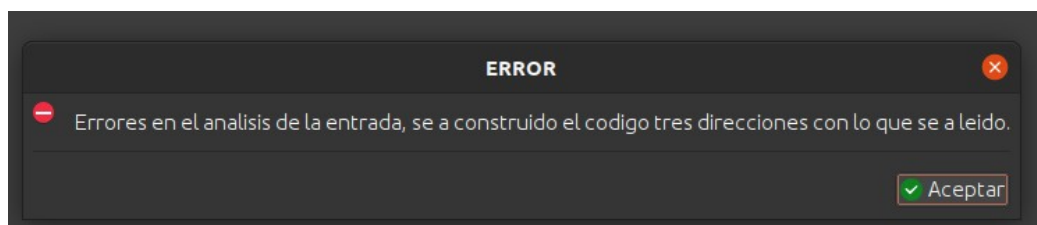
Error <Sintaxis | Semántico >

**Línea:**        <#Línea>

**Columna:**    <#Columna>

**Tipo Error:** < Tipo >

**Mensaje:**    < Descripción >



```
Error de Sintaxis:
  Linea #:      << 170 >>
  Columna #:    << 38 >>
  Token NO Reconocido:  << a >>
  Mensaje (Informacion):
                        -> Error no se encuentra definida la variable

Error Semantico:
  Linea #:      << 170 >>
  Columna #:    << 38 >>
  Valor:        << PRINTF >>
  Mensaje (Informacion):
                        -> Error, la cantidad de valores asignar, no hacen match con la cantidad de caracteres especificados.
```

### **REQUISITOS MÍNIMOS PARA ESTE PROGRAMA:**

- Microsoft .NET Framework versión 4.0
- Windows XP, Vista, 7, 8, 8.1, 10
- Linux Ubuntu 15.04 y Superior:
- Espacio en disco: 100 MB de espacio libre en disco
- Pentium 1 GHz o superior con 1GB de RAM o más