



# Tecnológico de Monterrey

Diseño de Compiladores  
(Gpo 1)

Febrero - Julio 2022  
Ing. Elda Quiroga

Proyecto Final

Antonio González Menchaca A01194261

Gerardo Peart Reynoso A01194337

**FIRMA ELECTRONICA**

Monterrey, Nuevo León

DESCRIPCIÓN DEL PROYECTO	1
<b>Propósito y Alcance del Proyecto.</b>	2
<b>Análisis de Requerimientos y descripción de los principales Test Cases.</b>	2
Requerimientos	2
Test Cases	2
<b>Descripción del Proceso</b>	3
Bitacoras	3
Commitments	5
Reflexion	5
<b>DESCRIPCIÓN DEL LENGUAJE</b>	5
Nombre del lenguaje.	5
Descripción genérica del lenguaje	5
Posibles Errores	5
<b>DESCRIPCIÓN DEL COMPILADOR:</b>	6
Equipo, Lenguaje y Librerías utilizadas.	6
Descripción del Análisis de Léxico	7
Patrones de Construcción	7
Enumeración de los "tokens"	8
Descripción del Análisis de Sintaxis	8
Descripción de Generación de Código Intermedio y Análisis Semántico	10
Código de operación y direcciones virtuales	10
Diagramas de Sintaxis	10
Acciones semánticas	17
Tabla de consideraciones semánticas (Cubo Semántico)	20
Tipos de memoria usada en la compilación	22
<b>DESCRIPCIÓN DE LA MÁQUINA VIRTUAL</b>	23
Descripción detallada del proceso de Administración de Memoria en ejecución	24

# 1. DESCRIPCIÓN DEL PROYECTO

### a. Propósito y Alcance del Proyecto.

El propósito de este proyecto es implementar los conocimientos adquiridos en clase de Diseño de Compiladores con el fin de crear un compilador funcional e integrarle a este compilador un funcionamiento que lo distinga sobre los demás.

El alcance de este proyecto, como se mencionó anteriormente es diseñar y construir un compilador funcional. En este caso se va a crear un estilo dashboard donde el usuario va a poder escribir y compilar su código. En este dashboard el usuario va tener la opción de compilar y ejecutar su código ahí mismo, o igualmente podrá descargar un OBJ y ejecutar el código en otro momento. Para distinguirse de los otros compiladores estamos buscando implementar una visualización gráfica sobre los elementos del compilador (Memoria Virtual, Directorio de Funciones, Cuadрупlos)

### b. Análisis de Requerimientos y descripción de los principales Test Cases.

#### Requerimientos

Código a compilar y ejecutar	El programa va a requerir un código a ejecutar, esto se podrá hacer ahí mismo en la plataforma o ser un documento OBJ ingresado por el usuario.
Limites de memoria	Debemos tener una idea del tamaño de los programas a ejecutar para implementar una memoria del tamaño ideal.

#### Test Cases

Test Case	Nombre Archivo	Proposito
TC # 1	tc1.txt	Verificar el funcionamiento completo del compilador: <ul style="list-style-type: none"><li>• Compilacion</li><li>• Ejecución</li><li>• Descarga OBJ</li><li>• Representación visual de datos</li></ul>
TC # 2	tc2.txt	Verificar que en caso que no pueda compilar se ejecute un error e informe al usuario

TC # 3	tc3.txt	Verificar que en caso que se presente un error durante la ejecución se detenga e informe al usuario.
TC # 4	tc1.obj	Verificar el funcionamiento completo del compilador desde un documento OBJ: <ul style="list-style-type: none"> <li>• Compilacion</li> <li>• Ejecución</li> <li>• Representación visual de datos</li> </ul>
TC # 5	tc3.obj	En caso que se haya descargado un OBJ con error se debe desplegar el error al momento de ejecutar.

### c. Descripción del Proceso

#### **Bitacoras**

Semana 0:

Antonio Gonzalez

Esta semana tenemos como meta comenzar a implementar el lexer y parser que realizamos en clase ( Little Duck). Comenzamos a trazar el programa por medio de diagramas de flujo. También queremos sacar toda la gramática de nuestro lenguaje.

Geraro Peart:

En esta semana empezamos a implementar el lexer y el parser que realizamos en clase. Empezamos a disenar el programa con las gramaticas y diagramas de flujo.

Semana 1:

Antonio Gonzalez

Esta semana nos juntamos a finalizar todos los elementos del léxico cual tuvimos problemas para ejecutar la vez pasada, agregamos la gran mayoría de las reglas gramaticales.

Geraro Peart:

En la semana 1 nos juntamos a terminal el lexico

Semana 2:

Antonio Gonzalez

Esta semana realmente no logre a trabajar en el proyecto más que para obtener la tabla semántica que vamos a utilizar,, fue una semana muy complicada pero lo importante es seguir trabajando.

Geraro Peart:

Esta semana no trabaje en el proyecto

Semana 3:

Antonio Gonzalez

Esta semana notamos unos errores dentro de la semántica que tenemos declarada, volvimos a hacer los diagramas y parece que ya están en orden. Comencé a ver como es el funcionamiento de pointer pero no he implementado nada.

Geraro Peart:

En esta semana nos dimos cuenta de algunos errores de los diagramas de flujo y trabajamos en corregirlos

Semana 4:

Antonio Gonzalez

Esta semana no tuve la oportunidad de trabajar en el proyecto, se me dificultó pero la próxima semana queremos ponernos al día

Geraro Peart:

En esta semana decidimos no trabajar en el proyecto

Semana 5:

Antonio Gonzalez

Esta semana ya finalizamos todos los estatutos, su código intermedio y comencemos trabajar con ciclos. Esperamos ponernos al corriente pronto.

Geraro Peart:

Semana 6:

Antonio Gonzalez

Estamos notando que no sirven las funciones que hemos implementado y parece tener un error el los arreglos. Estamos trabajando para solucionarlo.

Geraro Peart:

Semana 7:

Antonio Gonzalez

Este avance tenía como meta trabajar/finalizar en la documentación y la plataforma donde se ejecutara el compilador. Nosotros actualmente vamos un poco atrasados, pero finalizamos los funciones, arreglos y comenzamos a actualizar la documentación. Al ir actualizando la documentación se fueron eliminando y arreglando aspectos del código que no eran útiles. La plataforma por ahora no despliega nada más que dos ventanas, una para el input de texto y otra para el output del compilador. Estos días se le agregara el load file y otras funciones importantes.

Conclusion:

Antonio Gonzalez

Lamentablemente no encontramos manera de solucionar los problemas que tuvimos en recursividad, por lo tanto no completamos nuestro compilador. Fue una gran práctica implementar los conocimientos que vimos en clases, en especial como puede servir el manejo de memoria y cómo las estructuras de datos tienen un impacto en tal. Mi revisión no es hasta el 13, espero tener un programa del nivel esperado para ese entonces.

Geraro Peart:

## Commitments

1. El equipo va a trabajar de manera equitativa.
2. Ambos integrantes deben estar familiarizados con el contenido.
3. Habrá reuniones de manera semanal.
  - a. Se monitorea el avance
  - b. Se definen nuevos trabajos
  - c. Se aclaran dudas
4. Cada quien hará el avance que tenga que hacer a tiempo y de manera correcta.
5. En caso de no poder solucionar un problema nos apoyamos entre nosotros.

## Reflexion

Antonio Gonzalez



Geraro Peart:

## 2. DESCRIPCIÓN DEL LENGUAJE

- a. Nombre del lenguaje.

El lenguaje se ha nombrado: **VizComp**

- b. Descripción genérica del lenguaje

Este lenguaje se está creando como un lenguaje básico, que tendrá la capacidad de resolver expresiones, ciclos condicionales y funciones. De igual manera vamos a ofrecer al usuario un “Insight” de manera visual sobre el proceso de compilación.

- c. Posibles Errores

```
def catalogoErrores(error):  
    print("ERROR NUM: ", error[0])  
    if error[0] == 0:
```

```

        sys.exit("Error en duplicacion de variables: " + str(error[0]))
    elif error[0] == 1:
        sys.exit("Error en duplicacion de funciones: ", error[0])
    elif error[0] == 2:
        sys.exit("Error: Type mismatch: " + str(error[1]) + "-" + str(error[2]))
    elif error[0] == 3:
        sys.exit("Errors: Fake floor inexistente")
    elif error[0] == 4:
        sys.exit("Error: Type mistmatch en parametros", error[1], error[2])
    elif error[0] == 5:
        sys.exit("Error: Numero de parametros no valido")
    elif error[0] == 6:
        sys.exit("Error: Return en funcion VOID. " + error[1])
    elif error[0] == 7:
        sys.exit("Error: Variable no declarada " + error[1])
    elif error[0] == 8:
        sys.exit("Error tipo invalido en for loop")
    elif error[0] == 9:
        sys.exit("Error en la lectura del archivo")
    elif error[0] == 10:
        sys.exit("Error: Llamando dimensiones dentro de una variable no arreglo "
+ str(error[1]))
    else:
        sys.exit("TOKEN inesperado: ", error[1])

```

### 3. DESCRIPCIÓN DEL COMPILADOR:

#### a. Equipo, Lenguaje y Librerías utilizadas.

Equipo de computo	Lenguaje	Librerías		
PC	Python	Lex	-	Lexer
		Yacc	-	Parser
		Pandas	-	DataFrames (Estructura de
		Dash	-	Rep Visual

## b. Descripción del Análisis de Léxico

### Patrones de Construcción

<code>t_ignore = " \t\n"</code>	<code>t_PLUS = r'\+'</code>	<code>t_MINUS = r'\-'</code>
<code>t_MULT = r'\*'</code>	<code>t_MOD = r'\%'</code>	<code>t_DIV = r'\/'</code>
<code>t_EQUALS = r'='</code>	<code>t_SEMI = r';'</code>	<code>t_COLON = r':'</code>
<code>t_COMMA = r'\,'</code>	<code>t_LPAREN = r'\('</code>	<code>t_RPAREN = r'\)'</code>
<code>t_LBRACES = r'\{'</code>	<code>t_RBRACES = r'\}'</code>	<code>t_LBRACKET = r'\['</code>
<code>t_RBRACKET = r'\]'</code>	<code>t_LT = r'&lt;'</code>	<code>t_GT = r'&gt;'</code>
<code>t_LE = r'&lt;='</code>	<code>t_GE = r'&gt;='</code>	<code>t_EQ = r'=='</code>
<code>t_NE = r'!='</code>	<code>t_LOR = r'\ \ '</code>	<code>t_LAND = r'&amp;&amp;'</code>
<code>t_CTE_I = r'\d+'</code>	<code>t_CTE_F =</code> <code>r'((\d*\.\d+)(E[+-]? \d+)</code> <code>)? ([1-9]\d+E[+-]? \d+))</code> <code>'</code>	<code>t_CTE_S = r'\'.*?\''</code>
<code>t_RANGE = r'..'</code>		
<code>def t_ID(t):</code>  <code>r'[a-zA-Z_][a-zA-Z_0-9]*</code> <code>'</code>  <code>t.type =</code> <code>reserved.get(t.value, 'ID</code> <code>')</code>  <code>return t</code>	<code>def t_error(t):</code>  <code>print("Illegal character</code> <code>%s" % repr(t.value[0]))</code>  <code>t.lexer.skip(1)</code>	<code>def t_newline(t):</code>  <code>r'\n+'</code>  <code>t.lexer.lineno +=</code> <code>t.value.count("\n")</code>



## Enumeración de los "tokens"

Reserved Tokens	Tokens
<pre> 'if' : 'IF', 'else' : 'ELSE', 'elif' : 'ELSEIF', 'for' : 'FOR', 'while' : 'WHILE', 'do' : 'DO', 'to' : 'TO', 'programa' : 'PROGRAMA', 'principal' : 'PRINCIPAL', 'return' : 'RETURN', 'lee' : 'LEE', 'escribir' : 'ESCRIBIR', 'int' : 'INT', 'float' : 'FLOAT', 'char' : 'CHAR', 'funcion' : 'FUNCION', 'void' : 'VOID', 'bool' : 'BOOL', 'atributos' : 'ATRIBUTOS', 'metodos' : 'METODOS' </pre>	<pre> 'ID', 'PLUS', 'MINUS', 'MULT', 'MOD', 'DIV', 'EQUALS', 'COLON', 'SEMI', 'COMMA', 'LPAREN', 'RPAREN', 'LBRACKET', 'RBRACKET', 'LBRACES', 'RBRACES', 'LT', 'LE', 'GT', 'GE', 'EQ', 'NE', 'RANGE', 'LOR', 'LAND', 'CTE_I', 'CTE_F', 'CTE_S' </pre>

### c. Descripción del Análisis de Sintaxis

Nombre	Gramatica Formal
< Programa >	Programa $\rightarrow$ programa ID ; Programa2 Programa2 $\rightarrow$ ATRIBUTOS Programa2   Programa3 Programa3 $\rightarrow$ METODOS Programa3   PRINCIPAL
< Atributos >	Atributos $\rightarrow$ atributos VAR
< Metodos >	Metodos $\rightarrow$ metodos FUNCIONES

<b>&lt; Principal &gt;</b>	Principal $\rightarrow$ principal ( <i>ESTATUTOS</i> Principal2 Principal2 $\rightarrow$ <i>ESTATUTOS</i> Principal2   )
<b>&lt; Funciones &gt;</b>	Funciones $\rightarrow$ TIPO ID Funciones2 \ void ID Funciones2 Funciones2 $\rightarrow$ ( <i>PARAMETROS</i> ) { <i>ESTATUTOS</i> }
<b>&lt; Var &gt;</b>	Var $\rightarrow$ TIPO : Var2 Var2 $\rightarrow$ ID Var3   ID [ Cte-i ] Var3   ID [ Cte-i ] [ Cte-i ] Var3 Var3 $\rightarrow$ , Var2   ; Var   ;
<b>&lt; Parametros &gt;</b>	Parametros $\rightarrow$ TIPO ID , Parametros   TIPO ID
<b>&lt; Tipo &gt;</b>	Int   Float   Char   Bool
<b>&lt; EXP &gt;</b>	EXP $\rightarrow$ T-EXP   T-EXP
<b>&lt; T-EXP &gt;</b>	T-EXP $\rightarrow$ G-EXP   G-EXP &&
<b>&lt; G-EXP &gt;</b>	G-EXP $\rightarrow$ M-EXP G-EXP2 M-EXP G-EXP2 $\rightarrow$ >   <   >=   <=   ==   !+
<b>&lt; M-EXP &gt;</b>	M-EXP $\rightarrow$ T M-EXP2   T M-EXP2 $\rightarrow$ + M-EXP   - M-EXP
<b>&lt; T &gt;</b>	T $\rightarrow$ F T2   F T2 $\rightarrow$ * T   /T
<b>&lt; F &gt;</b>	F $\rightarrow$ ( EXP )   Cte-i   Cte-f   Cte-c   VAR   LLAMADA   ID   ID [ Cte-i .. Cte-i ]   ID [ Cte-i .. Cte-i , Cte-i .. Cte-i ]
<b>&lt; Estatutos &gt;</b>	Estatutos $\rightarrow$ ASIGNA   LLAMADA   LEE   ESCRIBE   CONDICION   CICLO-W   CICLO-F   FUNCIONES   RETORNA
<b>&lt; Asigna &gt;</b>	Asigna $\rightarrow$ VAR = EXP
<b>&lt; Llamada &gt;</b>	Llamada $\rightarrow$ ID ( EXP Llamada2 Llamada2 $\rightarrow$ , EXP Llamada2   )
<b>&lt; Lee &gt;</b>	Lee $\rightarrow$ lee ( Lee2 Lee2 $\rightarrow$ Cte-i )   Cte-f )   Cte-c )
<b>&lt; Escribe &gt;</b>	Escribe $\rightarrow$ escribir ( EXP Escribe2 Escribe2 $\rightarrow$ , EXP   )
<b>&lt; Condicion &gt;</b>	Condicion $\rightarrow$ if ( EXP ) Condicion2   if ( EXP ) Condicion3 Condicion2 $\rightarrow$ elif ( EXP ) Condicion2   Condicion3 Condicion3 $\rightarrow$ { <i>ESTATUTO</i> }   else { <i>ESTATUTO</i> }
<b>&lt; Ciclo-W &gt;</b>	Ciclo-W $\rightarrow$ while Ciclo-W2   do Ciclo-W3 Ciclo-W2 $\rightarrow$ ( EXP ) { <i>ESTATUTOS</i> } Ciclo-W2   ( EXP ) { <i>ESTATUTOS</i> } Ciclo-W3 $\rightarrow$ { <i>ESTATUTOS</i> } while ( EXP ) Ciclo-W3   { <i>ESTATUTOS</i> } while ( EXP )

<b>&lt; Ciclo-F &gt;</b>	Ciclo-F $\rightarrow$ ( for Ciclo-F2 Ciclo-F2 $\rightarrow$ Ciclo-F3 Ciclo-F2   Ciclo-F3 Ciclo-F3 $\rightarrow$ <i>VAR = EXP</i> to do ) { <i>ESTATUTOS</i> }
--------------------------	---



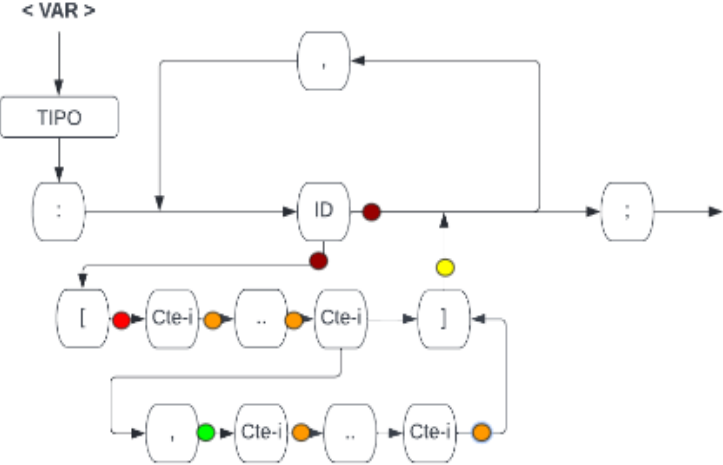
d. Descripción de Generación de Código Intermedio y Análisis Semántico

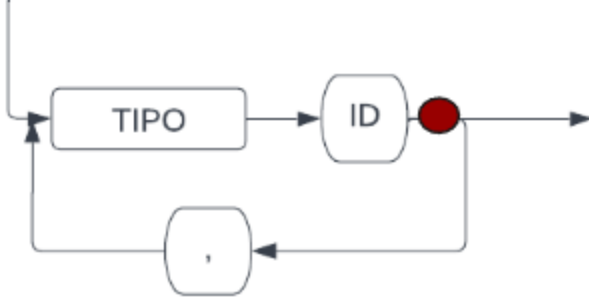
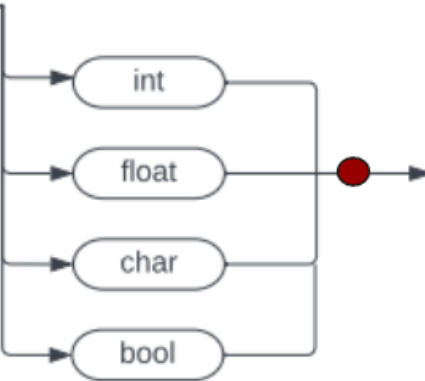
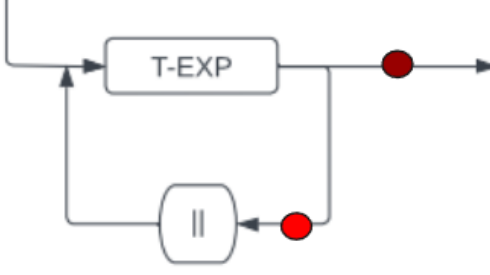
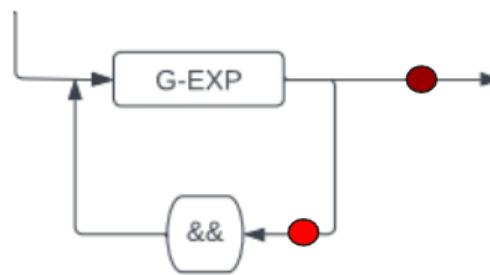
**Código de operación y direcciones virtuales**

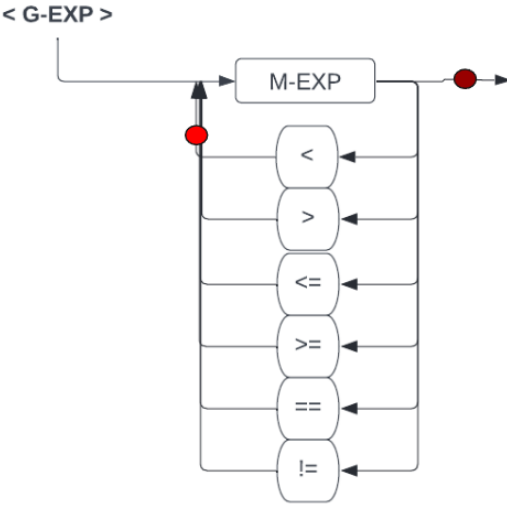
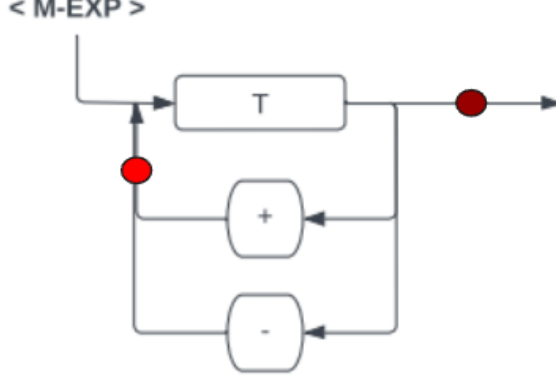
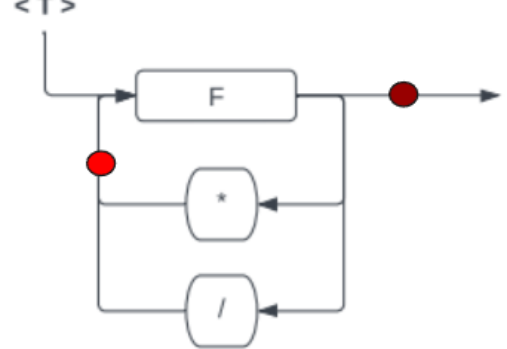
	<b>Const</b>	<b>Local</b>	<b>Global</b>	<b>Temp</b>
<b>Int</b>	0 - 999	4000 - 4999	8000 - 8999	12000 - 12999
<b>Float</b>	1000 - 1999	5000 - 5999	9000 - 9999	13000 - 13999
<b>Char</b>	2000 - 2999	6000 - 6999	10000 - 10999	14000 - 14999
<b>Bool</b>	3000 - 3999	7000 - 7999	11000 - 11999	15000 - 15900

**Diagramas de Sintaxis**

<p><b>&lt; PROGRAMA &gt;</b></p>	1. Genera goToMain 2. Crea el directorio de funciones 3. Ingresa la localización del main
<p><b>&lt; ATRIBUTOS &gt;</b></p>	1. Genera tabla de variables
<p><b>&lt; METODOS &gt;</b></p>	

	
	<ol style="list-style-type: none"> <li>1. Agregar valor al directorio de funciones</li> <li>2. Agregamos la cantidad de parámetros al directorio</li> <li>3. Agregamos la cantidad de variables y posición del cuádruplo</li> </ol>
	<p><b>RO</b>-Actualizamos el id actual y lo agregamos a la tabla de variables</p> <p><b>RC</b>-Marcamos la nueva variable como arreglo/matriz</p> <p><b>NA</b>-Guardamos el límite inferior, superior y r.</p> <p><b>VE</b>-Incrementamos la cantidad de dimensiones</p> <p><b>AM</b>-Cerramos la lista de nodos y calculamos el offset.</p>

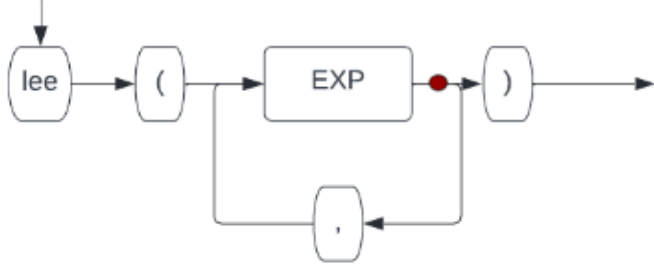

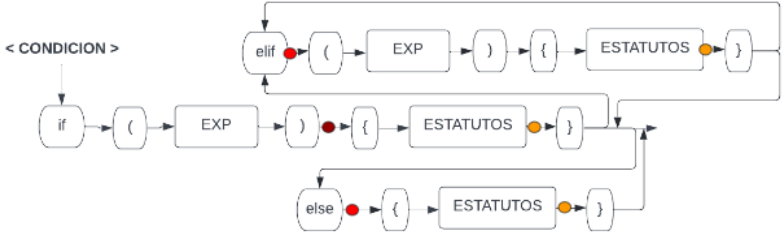
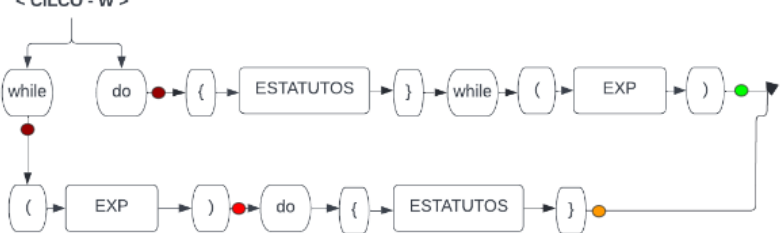
<p><b>&lt; PARAMETROS &gt;</b></p> 	<ol style="list-style-type: none"> <li>1. Marcamos el id actual y lo agregamos a la tabla de variables.</li> <li>2. Agregamos el tipo de parámetros a la pila de parámetros.</li> </ol>
<p><b>&lt; TIPO &gt;</b></p> 	<ol style="list-style-type: none"> <li>1. Actualizamos el pointer de tipo actual</li> </ol>
<p><b>&lt; EXP &gt;</b></p> 	<p><b>RO</b>-Revisamos la semántica de los operadores, creamos su cuádruplo correspondiente</p> <p><b>RC</b>-Agregamos el or a la pilaO</p>
<p><b>&lt; T-EXP &gt;</b></p> 	<p><b>RO</b>-Revisamos la semántica de los operadores, creamos su cuádruplo correspondiente</p> <p><b>RC</b>-Agregamos el or a la pilaO</p>

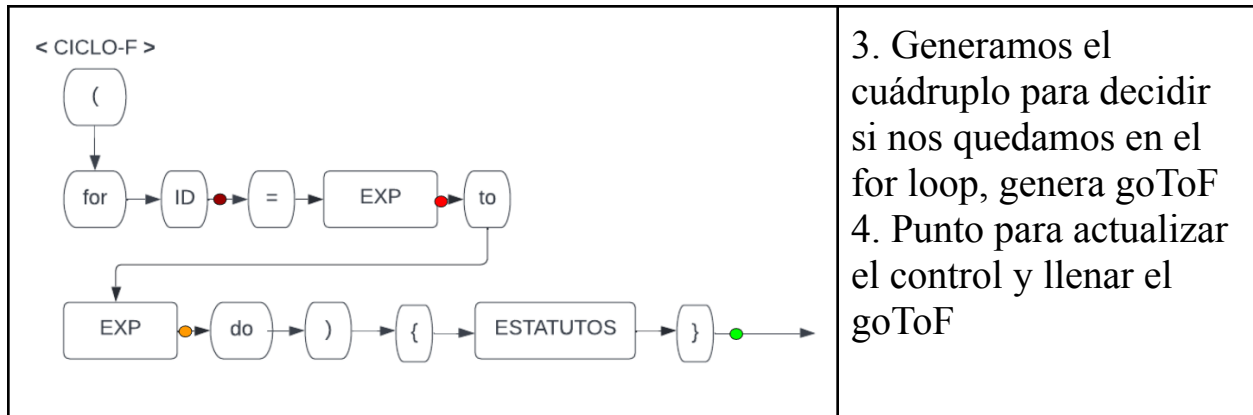
	<p><b>RO</b>-Revisamos la semántica de los operadores, creamos su cuádruplo correspondiente</p> <p><b>RC</b>-Agregamos el or a la pilaO</p>
	<p><b>RO</b>-Revisamos la semántica de los operadores, creamos su cuádruplo correspondiente</p> <p><b>RC</b>-Agregamos el or a la pilaO</p>
	<p><b>RO</b>-Revisamos la semántica de los operadores, creamos su cuádruplo correspondiente</p> <p><b>RC</b>-Agregamos el or a la pilaO</p>

	<p><b>RO-</b> Actualizamos la pila de tipos con el tipo de constante actual</p> <p><b>RC-</b> Agregamos un fakefloor</p> <p><b>NA-</b> Removemos el fakefloor de la pila</p> <p><b>VE-</b> Agregamos la variable a la pilaO y su tipo a ptye</p> <p><b>AM-</b> Creamos el primer cuádruplo de dimensiones</p> <p><b>AC-</b> Incrementamos la dimensión actual</p> <p><b>AO-</b> Generamos los otros cuádruplos para dimensiones.</p>
	<p><b>RO-</b> Ingresa id y su operando a su</p>

<p><b>&lt; ASIGNA &gt;</b></p>	<p>respectiva fila  <b>RC-</b> Verificamos que la variable sea arreglo y que exista. Ingresamos la información a la pila.  <b>NA-</b> Agregamos fake bottom a la pila  <b>VE-</b> Creamos el primer cuádruplo de dimensiones  <b>AM-</b> Incrementamos la dimension actual  <b>AC-</b> Generamos los otros cuádruplos para dimensiones.  <b>AO-</b> Removemos el fake floor de la pila</p>
<p><b>&lt; LLAMADA &gt;</b></p>	<p><b>RO-</b> Verificamos que exista la función que se está llamando.  <b>RC-</b> Generamos el cuádruplo para el parámetro  <b>NA-</b> Incrementamos el contador de parámetros  <b>VE-</b> Verificamos que la cantidad de parámetros sea correcta, creamos GOSUB</p>



<p>&lt; LEE &gt;</p> 	<p>1. Genera cuádruplo de lee</p>
<p>&lt; ESCRIBE &gt;</p> 	<p>1. Genera el cuádruplo print</p>
<p>&lt; CONDICION &gt;</p> 	<p><b>RO-</b> Generamos el cuádruplo goToF  <b>RC-</b> Generamos el cuádruplo goTo  <b>NA-</b> Llenamos la información de los goto</p>
<p>&lt; CILCO - W &gt;</p> 	<p><b>RO-</b> Guardamos la posición en la pila de saltos  <b>RC-</b> Generamos el cuádruplo para goToF  <b>NA-</b> Llenamos el cuádruplo goToF  <b>VE-</b> Generamos el cuádruplo para goToV</p>
	<p>1. Agregamos el valor definido el en for loop a la pilaO  2. Agregamos el cuádruplo para controlar el for</p>



### Acciones semánticas

Accion	Ubicación	Descripción
addDir	Programa, Atributos, Funciones,	Se puede ingresar de varios puntos, en programa crea la dirección de funciones. Crea tablas de variables para los atributos y funciones. Generamos el endfunc y liberamos la memoria temporal.
setCurrentID	Parametros Variables	Apunta al id actual
addValueVarT	Parametros Variables	Agrega la variable a la tabla de variables
addFakeBottom	F Asigna	Genera el fake bottom en pila [ Para editar el orden de operación y los arreglos.
popFakeBottom	F Asigna	Una vez finalizado se remueve el fake bottom
addO	Expresiones	Agrega operando a pila Oper
addOT	Expresiones	Agregamos el operando para and y or
checarExp	Expresiones	Obtenemos dos expresiones, verificamos su tipo con el operando y generamos el cuádruplo requerido.
addIntType	Expresiones	Agrega el valor constante a la tabla de constantes y crea su espacio en la memoria virtual.
addFloarType	Expresiones	Agrega el valor constante a la tabla de constantes y crea su espacio en la memoria virtual.
addCharType	Expresiones	Agrega el valor constante a la tabla de constantes y crea su espacio en la memoria virtual.

addBoolType	Expresiones	Agrega el valor constante a la tabla de constantes y crea su espacio en la memoria virtual.
pushID	Asignacion	En caso de no ser arreglo le da push a pilaO, tambien hacemos push en pOper
gotmain	Programa	Genera cuádruplo gotomain
setmainloc	Programa	Actualiza el salto para el gotomain
cuadprint	Escribe	Genera el cuádruplo para el print
cuadlee	Lee	Genera el cuádruplo para lee
returnFlag	Retornar	Sirve para flag en caso que una funcion void trate de retornar valor
cuadRetornar	Retornar	Genera el cuádruplo para retornar valor
getGT	Ciclos - For	Generamos el primer cuádruplo goTo y guardamos donde se encuentra para el salto a futuro.
genGT	Ciclos - For	Generamos el cuádruplo goTo y volvemos a guardar la posición para el salto.
fillGoto	Ciclos - For	Saca el salto de la pila de saltos, este valor se le asigna al goTo o goToF. Esto depende del valor ingresado.
storeWhile	Ciclos - While	Guardamos el valor del salto para el while dentro de la pila de saltos.
genDoWhile	Ciclos - While	En caso de ser do while obtenemos el salto de la pila, el resultado de la pilaO y generamos el cuádruplo.
fillWhile	Ciclos - While	En caso de ser while do se obtenemos el salto de la pila, el resultado de la pilaO y generamos los cuádruplos Goto y GotoF
addForID	Ciclos - For	Creamos el punto neurálgico para controlar el for loop por medio de un vControl. Generamos el cuádruplo vControl y uno de asignación.
comFor	Ciclos - For	Generamos un vFlnal para comparar la expresión. Luego generamos un gotoF para salir del for loop.
actFor	Ciclos - For	Esta función sirve para actualizar el vControl del for loop, actualizar el gotoF anterior y agregar un goTo para salir del for loop..
newParamType	Parametros	Ingresamos el tipo del nuevo parámetro a la param table.

addParamCount	Parametros	Agregamos la cantidad de parámetros y la tabla de parámetros al directorio de funciones.
genParameter	Parametros	Generamos el cuádruplo para los parámetros,
verLastParam	Parametros	Al ser el ultimo parametro verificamos que se hayan ingresado la cantidad correcta de parámetros y generamos el gosub
addContF	Funciones	Agregamos la cantidad de variables locales y la posición donde comienza la función a la tabla de parámetros
callFunc	Funciones	Cuando se llama una función verificamos que existe, se obtiene la tabla de parámetros y genera el ERA
setL	Arreglos	Agregamos el límite superior e inferior al nodo, en caso de ingresar el superior se calcula la r
newArray	Arreglos	Si se inicializa un array y lo marca, inicializamos dimensión y r en 1, creamos los nodos en un diccionario vacío.
cerrarNodes	Arreglos	Cuando ya termine de declarar dimensiones debe de calcular el offset. Por cada nodo calcula la m, en el caso del último nodo se agrega -k Sumamos el tamaño a la memoria virtual
cerrarCuadA	Arreglos	Generamos el cuádruplo verify, si tiene más de una dimensiones se debe de calcular el offset.
cerrarCuadB	Arreglos	Generamos la memoria virtual para los cuádruplos de las matrices, sumamos el valor del cuádruplo para obtener el offset. Al final generamos la memoria para el temporal y lo agregamos al cuádruplo.
agregarVar	Arreglos	Agregamos el arreglo y su tipo a la pila
verifyAEx	Arreglos	Verificamos que sea una llamada a un arreglo, en caso de serlo genera la dimensión y apunta al primer nodo.
updateDim	Arreglos	Actualizamos al nodo al que le apuntan, la pila dim y mandamos a llamar a su respectivo nodo.
increaseDim	Arreglos	Incrementamos la dimensión del nodo

Cuadruplo	Funcion
-----------	---------

GOTOMAIN - - 0	Genera el cuádruplo gotomain
GOTOMAIN - - len(cuad)	Asigna a gotomain la posición actual
END - - -	Marca el final del código ejecutable.
ENDFUNC - - -	Marca el final de una función.
= LeftO - newId	Asigna un valor a una variable
= LeftO - resultado	Asigna un valor a un apuntador (Arreglo o matriz)
Oper LeftO RightO tempcount	Resuelve una expresión aritmética y lo guarda en un temporal, esto incluye todas las operaciones aritméticas.
Print - - resultado	Imprime un valor
Lee resultado - result	Lee un valor y lo asigna a una variable
Return - - resultado	Retorna cierto resultado
GoToF resultado - 0	Genera el gotof - Donde el 0 representa el salto a dar
GoTo resultado - 0	Genera un salto goto - Donde el 0 representa el salto a dar
GoToV resultado - 0	Genera un salto goToV - Donde el 0 representa el salto a dar
PARAMETER argument par -	Genera el cuádruplo para lectura de parámetros, toma el argumento y lo asigna al parámetro.
ERA - - func	Prende el activation record de func
GOSUB func - n	Genera GOSUB para ir a la función, n representa su posición.
VERIFY val LI LS	Genera VERIFY para con la posición del arreglo.

**Tabla de consideraciones semánticas (Cubo Semántico)**

<b>INT</b>	<b>Int</b>	<b>Float</b>	<b>Char</b>	<b>Bool</b>
<b>+</b>	int	float	err	err
<b>-</b>	int	float	err	err

<b>%</b>	int	int	err	err
<b>*</b>	int	float	err	err
<b>/</b>	float	float	err	err
<b>&lt;</b>	bool	bool	err	err
<b>&gt;</b>	bool	bool	err	err
<b>!=</b>	bool	bool	err	err
<b>==</b>	bool	bool	err	err
<b>&lt;=</b>	bool	bool	err	err
<b>&gt;=</b>	bool	bool	err	err
<b>&amp;&amp;</b>	err	err	err	err
<b>  </b>	err	err	err	err
<b>=</b>	True	False	False	False

<b>FLOAT</b>	<b>Float</b>	<b>Char</b>	<b>Bool</b>
<b>+</b>	float	err	err
<b>-</b>	float	err	err
<b>%</b>	int	err	err
<b>*</b>	float	err	err
<b>/</b>	float	err	err
<b>&lt;</b>	bool	err	err
<b>&gt;</b>	bool	err	err
<b>!=</b>	bool	err	err
<b>==</b>	bool	err	err
<b>&lt;=</b>	bool	err	err
<b>&gt;=</b>	bool	err	err

<b>&amp;&amp;</b>	err	err	err
<b>  </b>	err	err	err
<b>=</b>	True	False	False

<b>CHAR</b>	<b>Char</b>	<b>Bool</b>	<b>BOOL</b>	<b>Bool</b>
+	char	err	+	err
-	char	err	-	err
%	err	err	%	err
*	char	err	*	err
/	err	err	/	err
<	err	err	<	err
>	err	err	>	err
!=	err	err	!=	bool
==	err	err	==	bool
<=	err	err	<=	err
>=	err	err	>=	err
<b>&amp;&amp;</b>	err	err	<b>&amp;&amp;</b>	bool
<b>  </b>	err	err	<b>  </b>	bool
<b>=</b>	True	False	<b>=</b>	True

### e. Tipos de memoria usada en la compilación

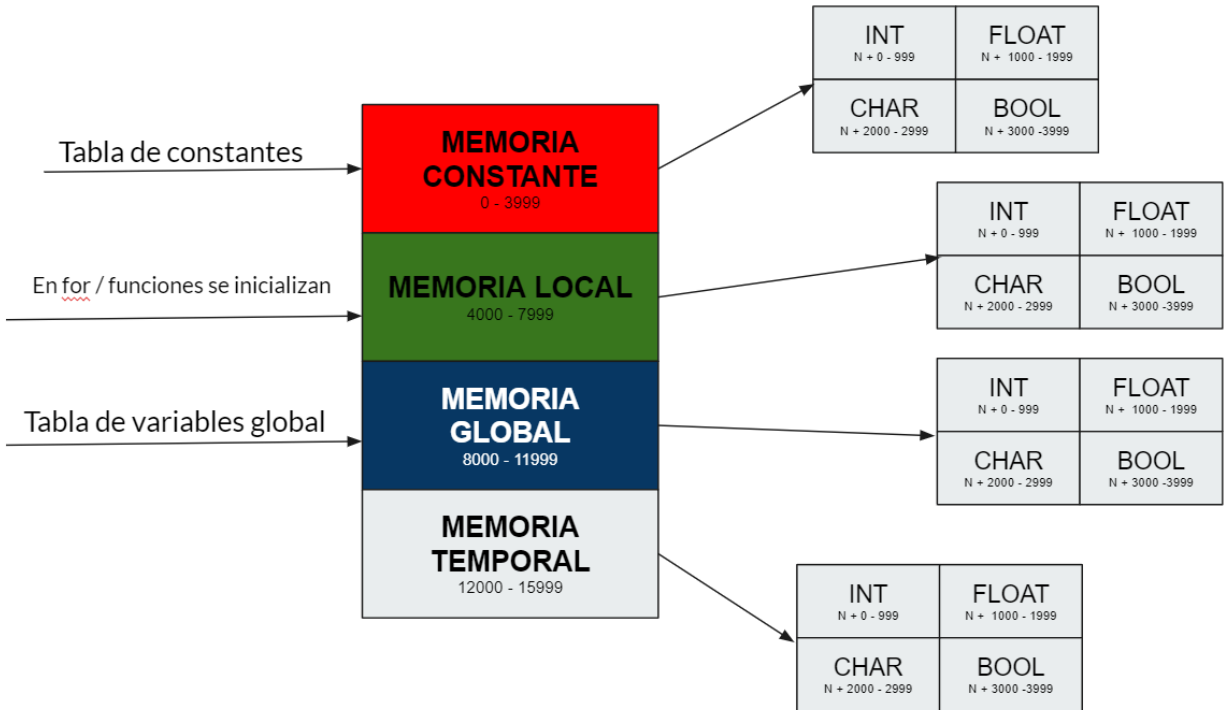
Variables		
Directorio de funciones Tabla de constantes Tabla de Variables	Diccionario	Ambas tablas se crearon utilizando diccionarios, la razón detrás de esta decisión es la facilidad que se presenta para analizar datos. Es decir de un solo elemento yo podía ingresar en atributos y

		<p>con tener acceso a la llave yo podía sacarlo libremente.</p> <p>Esto nos apoyó mucho a manejar la compilación y los diferentes tipos de datos.</p> <p>Un ejemplo de cómo logre acceder datos es: dic [ función actual ] [ variable ] [ memoria ].</p>
Pila Operandos, Operadores, ptype, pelse, pjums, pParam, pCurrentCall	Pilas	Las pilas fueron un gran apoyo para almacenar datos que no se iban a utilizar de manera inmediata, gracias a estas pudimos implementar orden de operación. Guardamos saltos en la generación de todos los goto
Cuádruplos	Clase	<p>Nosotros pensamos que la mejor manera de guardar los cuádruplos fue crear una clase y apendarlos a una lista, de esta manera los cuádruplos iban a ser iterables y todos los elementos se iban a guardar de una manera accesible, un ejemplo de como guardamos los cuádruplos.</p> <p>Cuad [ loc ] [ cuádruplo.top]</p>
Dim, vControl, cont, countCuad, parameterCounter, tempCount, dim	Contadores	Los contadores fueron una gran manera de poder ir monitoreando el comportamiento, estos se usarán para almacenar información, calcular valores de memoria y saltos.

#### 4. DESCRIPCIÓN DE LA MÁQUINA VIRTUAL



### a. Descripción detallada del proceso de Administración de Memoria en ejecución



#### Memoria Global

Dirección de Memoria	Diccionario { memoria : valor }
----------------------	---------------------------------

- Un diccionario para donde se le asignan los espacios reservados en compilación, la cantidad de variables es fija y se define en la compilación.

#### Memoria Local

Dirección de Memoria	Pila de diccionarios [ { memoria : valor } ]
----------------------	--

- En estas pilas se guardan los valores locales de una función en forma de diccionario, los parámetros se la asigna en este módulo de memoria., Por ser memoria local los valores de esta memoria no permanecen hasta el final. Por lo tanto, cuando termina una función se saca de la pila

#### Memoria Constante

Dirección de Memoria	Diccionario { memoria : valor }
----------------------	---------------------------------

- Un diccionario que se genera durante la etapa de compilación, se extrae en el obj y se vuelve a usar en esta etapa. Los valores son constantes, es decir son los valores ingresados por el usuario como: 7, 4/0, "Hola"

## Memoria Temporal

Dirección de Memoria	Diccionario { memoria : valor }
----------------------	---------------------------------

- Un diccionario donde se asignan las variables temporales, estas direcciones de memoria se generaron durante la compilación, no importa que se sobrescriben porque son temporales y su uso es inmediato.

Pila	Funcion
returnLoc	Esta pila sirve para guardar el punto de retorno después de una función, siempre se ingresa la función en la pila cuando se manda a llamar. El valor de la pila se saca en return o al finalizar la función, depende del tipo de función.
currentFun	Esta pila nos sirve como apoyo en la funciones que retornan un valor, con la función actual podemos saber cual es la dirección de memoria de la función. El valor se agrega al llamar la función y se saca al retornar.

Contadores	Funcion
Count	Se utiliza para marcar el paso de los cuádruplos, en caso de haber un salto se actualiza el valor de count a ese paso. En cualquier otra situación se suma uno y procede al siguiente paso.

Funciones	Funcion
getTpeVal	Toma como parámetros la posición y el valor de un pointer, los convierte al tipo apropiado dependiendo de la dirección de memoria.
getPointer	Toma como parámetros el valor de un pointer y la memoria requerida. Regresa la dirección de memoria de un pointer.
checkPointer	Toma como parámetros un apuntador y la memoria, en caso de ser una matriz o arreglo retorna la posición del pointer llamando getPointer
solveExpresion	Toma como parámetros un operador y un arreglo con dos valores, resuelve la expresión del operador y regresa el valor.

- getTypeValParam(pos)
  - Con esta función sacabamos el tipo de acuerdo a la memoria
- generatePointerMemory()
  - Con esta función podemos ver cuando una memoria virtual es un pointer

## 5. PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE :

- Incluir pruebas que "comprueben" el funcionamiento del proyecto:
  - Codificación de la prueba (en su lenguaje).
  - Resultados arrojados por la generación de código intermedio y por la ejecución.

Código	Cuadрупlos	Resultado
<pre> programa proyecto; atributos   int: first,a; metodos   int funcion getFibonacciNumberAt (int f) {   int: a, b;   if (f &lt;= 1) {     return (f)   }   else {     escribir(f)     a = (getFibonacciNumberA t(f-1)); </pre>	<pre> CODIGO INTERMEDIO GOTOMAIN,,,20 &lt;=,4000,0,15000 GotoF,15000,,5 return,,,4000 Goto,,,19 ERA,,,getFibonacciNu mberAt -,4000,1,12000 PARAMETER,12000,p ar0, GOSUB,getFibonacciN umberAt,,0 =,8002,,12001 =,12001,,4001 ERA,,,getFibonacciNu </pre>	<pre> Compilacion exitosa! 10 55 9 34 8 21 7 13 </pre>

<pre>         b = (getFibonacciNumberAt(f-2));         return (a + b)     } }  principal(     first = 10;     a = getFibonacciNumberAt(first);     escribir(first, a)     first = 9;     a = getFibonacciNumberAt(first);     escribir(first, a)     first = 8;     a = getFibonacciNumberAt(first);     escribir(first, a)     first = 7;     a = getFibonacciNumberAt(first);     escribir(first, a) ) </pre>	<pre> mberAt -,4000,2,12002 PARAMETER,12002,par0, GOSUB,getFibonacciNumberAt,,0 =,8002,,12003 =,12003,,4002 +,4001,4002,12004 return,,,12004 ENDFUNC,,, =,3,,8000 ERA,,,getFibonacciNumberAt PARAMETER,8000,par0, GOSUB,getFibonacciNumberAt,,0 =,8002,,12005 =,12005,,8001 print,,,8000 print,,,8001 =,4,,8000 ERA,,,getFibonacciNumberAt PARAMETER,8000,par0, GOSUB,getFibonacciNumberAt,,0 =,8002,,12006 =,12006,,8001 print,,,8000 print,,,8001 =,5,,8000 ERA,,,getFibonacciNumberAt </pre>	
---	---	--

	PARAMETER,8000,pa r0, GOSUB,getFibonacciN umberAt,,0 =,8002,,12007 =,12007,,8001 print,,,8000 print,,,8001 =,6,,8000 ERA,,,getFibonacciNu mberAt PARAMETER,8000,pa r0, GOSUB,getFibonacciN umberAt,,0 =,8002,,12008 =,12008,,8001 print,,,8000 print,,,8001 END,,, DIRECTORIO DE FUNCIONES global 8000 8001 funcion getFibonacciNumberAt Memoria 8002 Vars {'f': {'Type': 'int', 'Memoria': 4000}, 'a': {'Type': 'int', 'Memoria': 4001}, 'b': {'Type': 'int', 'Memoria': 4002}} ParamTable int, #VL 3 #VT 6	
--	---	--

	<p>TABLA DE CONSTANTES</p> <p>0,1 1,1 2,2 3,10 4,9 5,8 6,7</p>	
<div> <div>INGRESA ARCHIVO OBJ AQUI</div> <div>Compilation time 0.022005081176757812</div> <div>           Compilacion exitosa!            10            55            9            34            8            21            7            13         </div> </div>		

6. DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO:

7.

<https://github.com/Antoniogm0898/Complidores>

