



# **TeamPlusPlus**

## **Documentación Técnica**

### **Diseño de Compiladores**

#### **Profesores:**

Ing. Elda Guadalupe Quiroga González

Dr. Héctor Gibrán Ceballos Cancino

#### **Autores:**

---

Héctor Gibrán González Leal  
[A01282778]

---

Pedro Fernando Villezca Garza  
[A01282613]

2 de junio del 2021

# Índice

Descripción del Proyecto	3
Propósito y Alcance del Proyecto	3
Análisis de Requerimientos	4
Requerimientos Funcionales	4
Casos de Uso	5
Casos de Prueba	16
Proceso de Desarrollo	21
Bitácoras de Avance	21
Reflexión	25
Descripción del Lenguaje	27
Nombre del Lenguaje	27
Principales Características del Lenguaje	27
Listado de Errores	27
Descripción del Compilador	32
Equipo de Cómputo, Lenguaje y Utilerías	32
Análisis Léxico	33
Patrones de Construcción	33
Tokens	33
Análisis Sintáctico	34
Análisis Semántico y Generación de Código Intermedio	39
Administración de Memoria en Compilación	49
Descripción de la Máquina Virtual	56
Equipo de Cómputo, Lenguaje y Utilerías	56
Administración de Memoria en Ejecución	56
Pruebas del Funcionamiento del Lenguaje	62
Documentación del Código del Proyecto	76

# Descripción del Proyecto

## Propósito y Alcance del Proyecto

El propósito de este proyecto es diseñar y construir un lenguaje de programación básico orientado a objetos, llamado TeamPlusPlus. Para su desarrollo, se llevó a cabo la implementación de un compilador, encargado de la revisión básica y generación de código intermedio para los programas de este lenguaje; así como una máquina virtual, con la tarea de tomar dicho código intermedio y realizar la ejecución del mismo.

El alcance básico del proyecto requiere que el lenguaje soporte:

- Declaración de variables de tipos primitivos (enteros, flotantes y caracteres) y de hasta dos dimensiones
- Declaración de funciones con parámetros de tipos primitivos, así como la opción de retornar valores primitivos
- Declaración de clases con atributos y métodos
- Expresiones aritméticas, relacionales y lógicas
- Estatutos condicionales (if-else) y de repetición (while y from-to)
- Entrada y salida de datos en la terminal (read y write)
- Comentarios de una sola línea

Adicionalmente, para esta versión, se decidió incluir soporte para:

- Asignación de valores al momento de inicializar variables
- Estatutos condicionales con múltiples opciones (if-elif-else y switch)
- Dos niveles de accesibilidad para atributos y métodos (public y private)
- Herencia simple de clases

# Análisis de Requerimientos

## Requerimientos Funcionales

Partiendo del alcance establecido para el proyecto, se definieron los siguientes requerimientos funcionales:

<b>RF01</b>	El lenguaje debe permitir la declaración de variables de tipos primitivos: entero, flotante y caracter con la opción de asignar un valor inicial.
<b>RF02</b>	El lenguaje debe permitir la declaración de funciones con un valor de retorno opcional de tipo primitivo.
<b>RF03</b>	La declaración de funciones debe permitir cualquier cantidad, incluyendo ninguno, de parámetros de tipos primitivos.
<b>RF04</b>	El lenguaje debe permitir la declaración de clases con sus propios atributos y métodos que siguen los requerimientos de la declaración de variables y funciones respectivamente.
<b>RF05</b>	El lenguaje debe manejar los niveles de accesibilidad "public" y "private" para los atributos y métodos de las clases.
<b>RF06</b>	El lenguaje debe permitir la herencia simple entre clases.
<b>RF07</b>	El lenguaje debe poder resolver expresiones aritméticas, relacionales y lógicas.
<b>RF08</b>	El lenguaje debe manejar los estatutos condicionales "if-elif-else" para cualquier expresión y "switch" para expresiones de tipo entero o caracter.
<b>RF09</b>	El lenguaje debe manejar el estatuto de repetición condicional "while" y el estatuto de repetición delimitado "from-to".
<b>RF10</b>	El lenguaje debe permitir y manejar estatutos para la entrada y salida de datos desde la terminal.
<b>RF11</b>	El lenguaje debe permitir la inclusión de comentarios de una sola línea.

## Casos de Uso

<b>ID:</b> CU01	<b>Nombre:</b> Realizar operación lógica binaria
<b>Descripción:</b> El código realiza una operación con un operador lógico binario.	
<b>Precondiciones:</b> <ul style="list-style-type: none"><li>• Ninguna</li></ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"><li>1. El usuario escribe el primer operando.</li><li>2. El usuario escribe un operador lógico (and, or).</li><li>3. El usuario escribe el segundo operando.</li><li>4. El usuario ejecuta el código.</li><li>5. El compilador verifica que los operandos sean compatibles con el operador.</li><li>6. La máquina virtual realiza la operación correspondiente al operador sobre los operandos especificados.</li><li>7. La máquina virtual guarda el resultado de la operación en el espacio temporal correspondiente.</li></ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"><li>• El espacio temporal resultante de la expresión tiene un valor de tipo entero.</li></ul>	
<b>Flujo Alternativo</b>	5a. El compilador detecta que los operandos no son compatibles con el operador y arroja un error.

<b>ID:</b> CU02	<b>Nombre:</b> Realizar operación relacional
<b>Descripción:</b> El código realiza una operación con un operador relacional.	
<b>Precondiciones:</b> <ul style="list-style-type: none"><li>• Ninguna</li></ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"><li>1. El usuario escribe el primer operando.</li><li>2. El usuario escribe un operador relacional (&gt;, &gt;=, &lt;, &lt;=, ==, !=).</li></ol>

	<ol style="list-style-type: none"> <li>3. El usuario escribe el segundo operando.</li> <li>4. El usuario ejecuta el código.</li> <li>5. El compilador verifica que los operandos sean compatibles con el operador.</li> <li>6. La máquina virtual realiza la operación correspondiente al operador sobre los operandos especificados.</li> <li>7. La máquina virtual guarda el resultado de la operación en el espacio temporal correspondiente.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• El espacio temporal resultante de la expresión tiene un valor de tipo entero.</li> </ul>	
<b>Flujo Alternativo</b>	5a. El compilador detecta que los operandos no son compatibles con el operador y arroja un error.

<b>ID: CU03</b>	<b>Nombre:</b> Realizar operación aritmética
<b>Descripción:</b> El código realiza una operación con un operador aritmético.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>• Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>1. El usuario escribe el primer operando.</li> <li>2. El usuario escribe un operador aritmético (+, -, *, /).</li> <li>3. El usuario escribe el segundo operando.</li> <li>4. El usuario ejecuta el código.</li> <li>5. El compilador verifica que los operandos sean compatibles con el operador.</li> <li>6. La máquina virtual realiza la operación correspondiente al operador sobre los operandos especificados.</li> <li>7. La máquina virtual guarda el resultado de la operación en el espacio temporal correspondiente.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• El espacio temporal resultante de la expresión tiene un valor del tipo apropiado con base en los tipos de los operandos y la operación realizada.</li> </ul>	

<b>Flujo Alternativo</b>	5a. El compilador detecta que los operandos no son compatibles con el operador y arroja un error.
--------------------------	---

<b>ID: CU04</b>	<b>Nombre:</b> Realizar operación unaria
<b>Descripción:</b> El código realiza una operación con un operador unario.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>El usuario escribe un operador unario (+, -, not).</li> <li>El usuario escribe el operando.</li> <li>El usuario ejecuta el código.</li> <li>El compilador verifica que el operando sea compatible con el operador.</li> <li>La máquina virtual realiza la operación correspondiente al operador sobre el operando especificado.</li> <li>La máquina virtual guarda el resultado de la operación en el espacio temporal correspondiente.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>El espacio temporal resultante de la expresión tiene un valor del tipo apropiado con base en el tipo del operando y la operación realizada.</li> </ul>	
<b>Flujo Alternativo</b>	5a. El compilador detecta que el operando no es compatible con el operador y arroja un error.

<b>ID: CU05</b>	<b>Nombre:</b> Realizar escritura (print)
<b>Descripción:</b> El código despliega un valor o letrero en pantalla.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	

<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>1. El usuario escribe el estatuto 'print' con uno o más expresiones o letreros separados por comas.</li> <li>2. El usuario ejecuta el código.</li> <li>3. El compilador verifica que los valores a imprimir sean compatibles con el estatuto.</li> <li>4. La máquina virtual evalúa los valores y los despliega en la pantalla.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• Los valores están desplegados en la pantalla.</li> </ul>	
<b>Flujo Alternativo</b>	3a. El compilador detecta un valor incompatible con el estatuto y arroja un error.

<b>ID: CU06</b>	<b>Nombre:</b> Realizar lectura (read)
<b>Descripción:</b> El código recibe y guarda un valor de entrada del usuario.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>• Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>1. El usuario escribe el estatuto 'read' con una o más variables separadas por comas.</li> <li>2. El usuario ejecuta el código.</li> <li>3. El compilador verifica que las variables a leer sean compatibles con el estatuto.</li> <li>4. El usuario escribe las entradas en la terminal.</li> <li>5. La máquina virtual evalúa los valores y los guarda en el espacio de memoria correspondiente.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• Los valores de entrada están almacenados en los espacios de memoria correspondientes.</li> </ul>	
<b>Flujo Alternativo</b>	3a. El compilador detecta una variable incompatible con el estatuto y arroja un error.



	5a. La máquina virtual detecta una entrada incompatible con su variable correspondiente y arroja un error.
--	--

<b>ID: CU07</b>	<b>Nombre:</b> Llamar a una función
<b>Descripción:</b> El código realiza la ejecución de la función que se manda a llamar.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>1. El usuario escribe el nombre de la función a llamar.</li> <li>2. El usuario escribe entre paréntesis los argumentos a mandar a la función.</li> <li>3. El usuario ejecuta el código.</li> <li>4. El compilador valida que la función llamada ya esté declarada.</li> <li>5. El compilador verifica que los argumentos especificados coincidan exactamente con los parámetros de la función.</li> <li>6. La máquina virtual ejecuta el código de la función y, si tiene valor de retorno, almacena su valor en un espacio temporal.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>Si la función llamada tiene un valor de retorno, este valor está asignado al espacio temporal resultante de la llamada.</li> </ul>	
<b>Flujo Alternativo</b>	<p>4a. El compilador detecta que la función llamada no está declarada y arroja un error.</p> <p>5a. El compilador encuentra un argumento que no coincide con el tipo de su parámetro correspondiente y arroja un error.</p> <p>5b. El compilador detecta un número diferente de argumentos que de parámetros y arroja un error.</p>

<b>ID: CU08</b>	<b>Nombre:</b> Asignar un valor a una variable
<b>Descripción:</b> El código asigna un valor a una variable.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>El usuario escribe el nombre de una variable.</li> <li>El usuario escribe el operador de asignación (=).</li> <li>El usuario escribe la expresión correspondiente al valor a asignar.</li> <li>El usuario ejecuta el código.</li> <li>El compilador verifica que la variable esté declarada.</li> <li>El compilador verifica que los tipos de la variable y la expresión sean idénticos.</li> <li>La máquina virtual evalúa la expresión y guarda el valor resultante en el espacio de memoria correspondiente a la variable.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>El valor de la expresión está asignado al espacio de memoria de la variable.</li> </ul>	
<b>Flujo Alternativo</b>	<p>5a. El compilador detecta que la variable no está declarada y arroja un error.</p> <p>6a. El compilador detecta que los tipos no son idénticos y arroja un error.</p>

<b>ID: CU09</b>	<b>Nombre:</b> Realizar estatuto if-elif-else
<b>Descripción:</b> El código ejecuta un estatuto condicional 'if-elif-else'	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>El usuario escribe un 'if' seguido de una expresión.</li> </ol>

	<ol style="list-style-type: none"> <li>2. El usuario escribe el bloque de código correspondiente al 'if'.</li> <li>3. El usuario escribe de cero a muchos 'elif's seguidos de expresiones.</li> <li>4. El usuario escribe un bloque de código correspondiente a cada 'elif'.</li> <li>5. El usuario escribe cero o un 'else'.</li> <li>6. El usuario escribe el bloque de código correspondiente al 'else', si hubo.</li> <li>7. El usuario ejecuta el código.</li> <li>8. El compilador verifica que todas las expresiones tengan como resultado un valor entero.</li> <li>9. La máquina virtual evalúa cada expresión y ejecuta solamente el bloque de la primera que resulte verdadera.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• El código ejecutó solamente uno o ninguno de los bloques de código que forman parte del estatuto.</li> </ul>	
<b>Flujo Alternativo</b>	8a. El compilador detecta una expresión cuyo resultado no es de tipo entero y arroja un error.

<b>ID: CU10</b>	<b>Nombre:</b> Realizar estatuto switch
<b>Descripción:</b> El código ejecuta un estatuto condicional 'switch'.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>• Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>1. El usuario escribe un 'switch' seguido de una expresión.</li> <li>2. El usuario escribe uno o muchos 'case' seguidos por una constante.</li> <li>3. El usuario escribe un bloque de código correspondiente a cada 'case'.</li> <li>4. El usuario escribe cero o un 'default'.</li> </ol>

	<ol style="list-style-type: none"> <li>5. El usuario escribe un bloque de código correspondiente al 'default'.</li> <li>6. El usuario ejecuta el código.</li> <li>7. El compilador verifica que el resultado de la expresión sea entero o caracter.</li> <li>8. El compilador verifica para cada 'case' que el tipo de la constante dada coincida con el resultado de la expresión.</li> <li>9. La máquina virtual evalúa la expresión del 'switch'.</li> <li>10. La máquina virtual compara el valor resultante de la expresión con cada constante dada y ejecuta solamente el bloque de la primera que resulte idéntica.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• El código ejecutó solamente uno o ninguno de los bloques de código que forman parte del estatuto.</li> </ul>	
<b>Flujo Alternativo</b>	<ol style="list-style-type: none"> <li>7a. El compilador detecta que el resultado de la expresión no es de tipo entero o caracter y arroja un error.</li> <li>8a. El compilador detecta que la constante dada en algún caso no coincide con el tipo del resultado de la expresión y arroja un error.</li> </ol>

<b>ID: CU11</b>	<b>Nombre:</b> Realizar estatuto while
<b>Descripción:</b> El código ejecuta un estatuto cíclico 'while'.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>• Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>1. El usuario escribe un 'while' seguido de una expresión.</li> <li>2. El usuario escribe el bloque de código correspondiente al 'while'.</li> <li>3. El usuario ejecuta el código.</li> <li>4. El compilador verifica que la expresión tenga un resultado de tipo entero.</li> </ol>

	<ol style="list-style-type: none"> <li>La máquina virtual evalúa la expresión. Si resulta verdadera, ejecuta el bloque de código. Si resulta falsa, termina el caso de uso.</li> <li>La máquina virtual regresa al paso 5 si el bloque de código fue ejecutado.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>El código ejecutó el bloque de código que forma parte del estatuto entre cero y múltiples veces.</li> </ul>	
<b>Flujo Alternativo</b>	4a. El compilador detecta que el resultado de la expresión no es de tipo entero y arroja un error.

<b>ID: CU12</b>	<b>Nombre:</b> Realizar estatuto from-to
<b>Descripción:</b> El código ejecuta un estatuto cíclico 'from-to'.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>El usuario escribe un 'from' seguido de una asignación a una variable.</li> <li>El usuario escribe un 'to' seguido de una expresión.</li> <li>El usuario escribe un bloque de código correspondiente al 'from-to'.</li> <li>El usuario ejecuta el código.</li> <li>El compilador verifica que la variable asignada sea de tipo entero.</li> <li>El compilador verifica que la expresión tenga un resultado de tipo entero.</li> <li>La máquina virtual evalúa la expresión. Si el valor actual de la variable es menor o igual a su resultado, ejecuta el bloque de código. Si no, termina el caso de uso.</li> <li>La máquina virtual le agrega 1 al valor de la variable y regresa al paso 5 si el bloque de código fue ejecutado.</li> </ol>

<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>El código ejecutó el bloque de código que forma parte del estatuto entre cero y múltiples veces.</li> </ul>	
<b>Flujo Alternativo</b>	5a. El compilador detecta que la variable asignada no es de tipo entero y arroja un error. 6a. El compilador detecta que el resultado de la expresión no es de tipo entero y arroja un error.

<b>ID: CU13</b>	<b>Nombre:</b> Declarar una función
<b>Descripción:</b> El usuario declara una función.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>El usuario escribe el tipo de retorno de la función (primitivo, void).</li> <li>El usuario escribe 'func' seguido del nombre de la función y su lista de parámetros (tipo y nombre).</li> <li>El usuario escribe el bloque de código correspondiente a la función.</li> <li>El usuario ejecuta el código.</li> <li>El compilador verifica que la función no haya sido declarada previamente.</li> <li>El compilador verifica que el bloque de código contenga un estatuto de retorno o no dependiendo del tipo de retorno.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>El código reconoce a la función declarada.</li> </ul>	
<b>Flujo Alternativo</b>	5a. El compilador encuentra otra función con el mismo nombre y arroja un error. 6a. El compilador detecta que hay un 'return' en una función de tipo void y arroja un error.

	6b. El compilador detecta que no hay un 'return' en una función de tipo primitivo y arroja un error.
--	--

<b>ID: CU14</b>	<b>Nombre:</b> Declarar una variable
<b>Descripción:</b> El usuario declara una variable en el contexto actual.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>El usuario escribe el tipo de dato, ya sea primitivo o estructurado.</li> <li>El usuario escribe uno o muchos nombres de variables separados por comas.</li> <li>El usuario ejecuta el código.</li> <li>El compilador verifica para cada variable que no haya sido declarada previamente.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>Se crean una o más variables en el programa.</li> </ul>	
<b>Flujo Alternativo</b>	4a. El compilador detecta que ya se ha declarado una variable con el mismo nombre y arroja un error.

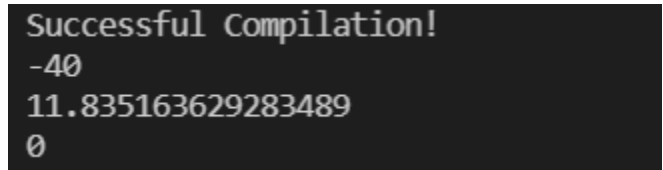
<b>ID: CU15</b>	<b>Nombre:</b> Declarar una clase
<b>Descripción:</b> El usuario declara una clase.	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>Ninguna</li> </ul>	
<b>Flujo de Eventos</b>	<ol style="list-style-type: none"> <li>El usuario escribe 'class' seguido del nombre de la clase.</li> <li>El usuario escribe cero o un 'inherits' seguido por un nombre de otra clase de la cual va a heredar esta clase.</li> </ol>

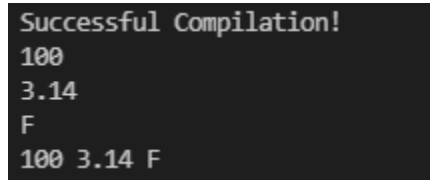
	<ol style="list-style-type: none"> <li>3. El usuario escribe 'attributes' seguido de cero o más declaraciones de variables [CU14].</li> <li>4. El usuario escribe 'methods' seguido de cero o más declaraciones de funciones [CU13].</li> <li>5. El usuario ejecuta el código.</li> <li>6. El compilador verifica que la clase no haya sido declarada previamente.</li> <li>7. El compilador verifica que la clase de la que se hereda, si la hay, haya sido declarada previamente.</li> </ol>
<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• La clase queda declarada para su uso en el programa.</li> </ul>	
<b>Flujo Alternativo</b>	<p>6a. El compilador detecta que ya existe una clase con el mismo nombre y arroja un error.</p> <p>7a. El compilador detecta que la clase de la que se hereda no ha sido declarada y arroja un error.</p>

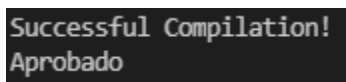
## Casos de Prueba

<b>ID</b>	CP01
<b>Objetivo</b>	Verificar funcionamiento de expresiones aritméticas, relacionales y lógicas.
<b>Código</b>	<pre> program expressions;  main() {     vars     int w = 2, x = 13, y = -3, z = 32, expr1;     float a = 3.21, b = 67.45, c = 22.5298, expr2;     int i, j, k, expr3;      expr1 = w + (-x - y) - +z;     expr2 = ((b / a) * -c) / expr1;     expr3 = not((a &gt;= w) or (b &lt; x) and c != z);      print(expr1, "\n");     print(expr2, "\n");     print(expr3, "\n"); } </pre>

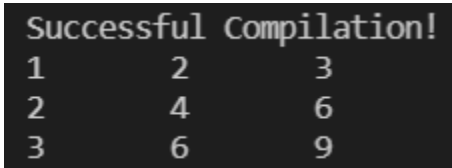


<b>Resultado Esperado</b>	-40 11.835164 0
<b>Resultado Real</b>	
<b>Resultado de la Prueba</b>	Aprobada

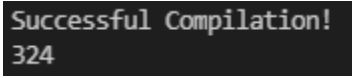
<b>ID</b>	CP02
<b>Objetivo</b>	Verificar funcionamiento de estatutos de entrada y salida.
<b>Código</b>	<pre> program io;  main() {     vars     int a;     float b;     char c;      read(a, b, c);     print(a, " ", b, " ", c, "\n"); } </pre>
<b>Entradas</b>	100 3.14 F
<b>Resultado Esperado</b>	100 3.14 F
<b>Resultado Real</b>	
<b>Resultado de la Prueba</b>	Aprobada

<b>ID</b>	CP03
<b>Objetivo</b>	Verificar funcionamiento de estatutos condicionales.
<b>Código</b>	<pre> program conditionals;  main() {     vars     float calificacion = 75.0;     char letra;      if (calificacion &gt;= 90) {         letra = 'A';     } elif (calificacion &gt;= 70) {         letra = 'C';     } else {         letra = 'F';     }      switch(letra) {         case 'A' {             print("Muy buena calificacion \n");         }         case 'C' {          }         case 'F' {          }     } } </pre>
<b>Resultado Esperado</b>	Aprobado
<b>Resultado Real</b>	
<b>Resultado de la Prueba</b>	<b>Aprobada</b>

<b>ID</b>	CP04
<b>Objetivo</b>	Verificar funcionamiento de estatutos cíclicos y arreglos.
<b>Código</b>	program cycles;

	<pre> vars     int arr[3,3];  main() {     vars     int i, j;      from i=0 to 2 {     from j=0 to 2 {         arr[i, j] = (i+1) * (j+1);     }     }      i = 0;     while(i &lt;= 2) {     j = 0;     while(j &lt;= 2) {         print(arr[i, j], "\t");         j = j+1;     }     print("\n");     i = i+1;     } } </pre>
<b>Resultado Esperado</b>	<pre> 1  2  3 2  4  6 3  6  9 </pre>
<b>Resultado Real</b>	
<b>Resultado de la Prueba</b>	<b>Aprobada</b>

<b>ID</b>	CP05
<b>Objetivo</b>	Verificar funcionamiento de funciones parametrizables.
<b>Código</b>	<pre> program functions;  func int multiply(int a, int b) {     return(a*b); } </pre>

	<pre>main() {     vars     int a = 3, b = 6;      print(multiply(multiply(a, a), multiply(b, b))); }</pre>
<b>Resultado Esperado</b>	324
<b>Resultado Real</b>	
<b>Resultado de la Prueba</b>	<b>Aprobada</b>

<b>ID</b>	CP06
<b>Objetivo</b>	Verificar funcionamiento de clases y objetos.
<b>Código</b>	<pre>program objects;  class Cat {     attributes     private char eyeColor;     methods     public func void meow() {         print("Meow... \n");     }      public func void setEyeColor(char color) {         eyeColor = color;     }     public func char getEyeColor() {         return(eyeColor);     } };  vars     Cat tom;  main() {     vars     char color = 'r';      tom.meow();</pre>

	<pre>tom.setEyeColor(color); print(tom.getEyeColor()); }</pre>
<b>Resultado Esperado</b>	Meow... r
<b>Resultado Real</b>	
<b>Resultado de la Prueba</b>	<b>Aprobada</b>

## Proceso de Desarrollo

Para cada avance semanal de este proyecto, se realizó primero una discusión a través de una llamada por **Discord** para planear y diseñar las tareas que se realizarán el resto de la semana. En estas discusiones se definieron aspectos como la gramática del lenguaje, la posición y funcionalidad de los puntos neurálgicos para nuestro análisis semántico y cualquier cambio necesario a elementos correspondientes a avances previos. Para la implementación del proyecto, se utilizó la dinámica de **pair-programming**, de manera que ambos autores del lenguaje trabajaron en cada segmento del código simultáneamente a través de la herramienta de **Visual Studio Code**, que cuenta con una extensión llamada **LiveShare**, que permite el desarrollo de código colaborativamente.

Debido a que en la actualidad por causas de fuerza mayor se tiene que trabajar de manera remota, estas herramientas fueron cruciales para completar el proyecto en tiempo y forma.

### Bitácoras de Avance

Fecha	Avance
29/03/2021	Se realiza la planeación del proyecto y se definen los cambios iniciales a la descripción del proyecto. Luego, se genera la definición regular del lenguaje con todas las tokens a utilizar. Comienza la generación de diagramas de sintaxis.

	<b>Commits:</b> Ninguno
31/03/2021	<p>Termina la generación de diagramas de sintaxis y se implementa el código en ANTLR para el analizador léxico y sintáctico del lenguaje.</p> <p><b>Commits:</b> <a href="#">#1</a>, <a href="#">#2</a>, <a href="#">#3</a></p>
05/04/2021	<p>Se realizan correcciones a la gramática de la declaración de funciones.</p> <p><b>Commits:</b> <a href="#">#1</a></p>
06/04/2021	<p>Se realizan modificaciones adicionales a nuestra gramática para reflejar los cambios en el alcance del proyecto</p> <p><b>Commits:</b> <a href="#">#1</a></p>
<b>09/04/2021</b>	<b>Entrega del avance 1</b>
11/04/2021	<p>Se diseñan los puntos neurálgicos para el análisis de semántica básica de variables y se ubican en los diagramas de sintaxis. Se investiga cómo funciona la implementación del análisis semántico en ANTLR utilizando Python 3.</p> <p><b>Commits:</b> Ninguno</p>
13/04/2021	<p>Se modifica la implementación de nuestra gramática en ANTLR para acomodar los nuevos puntos neurálgicos. Se implementa el directorio de funciones, clases y variables, así como el análisis semántico que los llena.</p> <p><b>Commits:</b> <a href="#">#1</a>, <a href="#">#2</a>, <a href="#">#3</a></p>
14/04/2021	<p>Se refactoriza el código para el análisis semántico de clases, funciones y variables. Se realiza el diseño de la tabla de consideraciones semánticas.</p> <p><b>Commits:</b> <a href="#">#1</a>, <a href="#">#2</a>, <a href="#">#3</a>, <a href="#">#4</a></p>
<b>16/04/2021</b>	<b>Entrega del avance 2</b>

17/04/2021	<p>Se diseñan los puntos neurálgicos correspondientes al manejo de expresiones con cuádruplos y se ubican en los diagramas de sintaxis. Se diseñan correcciones a la gramática con respecto a la jerarquía de operadores.</p> <p><b>Commits:</b> Ninguno</p>
18/04/2021	<p>Se hacen correcciones a la gramática relacionadas a los operadores unarios y la jerarquía de otros operadores. Se implementa el cubo semántico, así como los módulos 'CustomListener' y 'QuadrupleList' para la creación y manejo de cuádruplos.</p> <p><b>Commits:</b> <a href="#">#1</a>, <a href="#">#2</a>, <a href="#">#3</a></p>
19/04/2021	<p>Se implementa la semántica de expresiones.</p> <p><b>Commits:</b> <a href="#">#1</a></p>
20/04/2021	<p>Se diseñan los puntos neurálgicos para la semántica de estatutos lineales y se ubican en los diagramas de sintaxis. Se modifica la implementación de la gramática en ANTLR para acomodar estos puntos.</p> <p><b>Commits:</b> <a href="#">#1</a></p>
21/04/2021	<p>Se implementa la semántica de estatutos condicionales.</p> <p><b>Commits:</b> <a href="#">#1</a></p>
<b>23/04/2021</b>	<b>Entrega del avance 3</b>
24/04/2021	<p>Se diseñan los puntos neurálgicos para estatutos condicionales y ciclos y se ubican en los diagramas de sintaxis.</p> <p><b>Commits:</b> Ninguno</p>
25/04/2021	<p>Se modifica la implementación de la gramática en ANTLR para acomodar los puntos de estatutos condicionales y ciclos. Posteriormente, se realiza la implementación de estos puntos.</p> <p><b>Commits:</b> <a href="#">#1</a>, <a href="#">#2</a>, <a href="#">#3</a></p>

<b>30/04/2021</b>	<b>Entrega del avance 4</b>
01/05/2021	Se modifica la gramática y se diseñan los puntos neurálgicos para la generación de código de funciones.  <b>Commits:</b> <a href="#">#1</a>
02/05/2021	Se realiza la implementación de los puntos neurálgicos para la semántica de funciones.  <b>Commits:</b> <a href="#">#1</a>
03/05/2021	Se realiza la refactorización para incluir direcciones virtuales.  <b>Commits:</b> <a href="#">#1</a> , <a href="#">#2</a>
<b>08/05/2021</b>	<b>Entrega del avance 5</b>
10/05/2021	Se agrega el cuádruplo de GOTO a main al inicio de la lista.  <b>Commits:</b> <a href="#">#1</a>
12/05/2021	Se agrega el manejador de direcciones para valores constantes. Se agregan las variables globales para returns de funciones.  <b>Commits:</b> <a href="#">#1</a> , <a href="#">#2</a>
13/05/2021	Se agrega la implementación del mapeo de memoria para ejecución, así como la primera implementación de la máquina virtual, para expresiones y estatutos lineales.  <b>Commits:</b> <a href="#">#1</a>
14/05/2021	Se agrega la excepción para división entre 0.  <b>Commits:</b> <a href="#">#1</a>
<b>16/05/2021</b>	<b>Entrega del avance 6</b>
17/05/2021	Se implementa la semántica para la declaración de arreglos. Se inicia la semántica para indexación de arreglos.



	<b>Commits:</b> <a href="#">#1</a> , <a href="#">#2</a>
18/05/2021	Se termina la indexación de arreglos.  <b>Commits:</b> <a href="#">#1</a> ,
21/05/2021	Se modifica la gramática y se diseñan los puntos neurálgicos para la declaración de instancias. Se implementa la semántica y generación de código para la declaración de instancias. Se realiza el mapeo de memoria virtual para instancias.  <b>Commits:</b> <a href="#">#1</a> , <a href="#">#2</a>
22/05/2021	Se modifica la gramática y se diseñan los puntos neurálgicos para el uso de instancias. Se implementa la semántica y generación de código para el uso de instancias.  <b>Commits:</b> <a href="#">#1</a>
<b>25/05/2021</b>	<b>Entrega del avance 7</b>

## Reflexión

### Héctor Gibrán González Leal

Para mí este proyecto fue la culminación de todo lo que he aprendido durante la carrera. A lo largo de los avances, tuve que ir incorporando conocimientos de las diferentes clases que había tomado, ya sea sobre estructuras de datos y algoritmos, hasta ingeniería de software para la documentación y control de versiones. El proyecto que desarrollamos me gustó mucho, creo que tuvo un muy buen equilibrio entre aplicación de conocimientos previos, aplicación de temas vistos en clase y retos. Es claro que sigue siendo algo muy pequeño si lo comparamos con los lenguajes que hoy dominan la industria, pero aún así creo que se vieron muy bien los fundamentos que construyen dichos lenguajes, al punto de que ahora puedo darme una idea sobre el proceso que se tuvo que hacer para construir un lenguaje como C, Java, incluso Python.



**Pedro Fernando Villezca Garza**

Desde que vimos la descripción y los requerimientos, el proyecto me pareció muy retador. El aspecto principal que me hizo verlo de esta manera fue la cantidad de componentes sobre los cuales no teníamos conocimiento alguno al momento de empezar a trabajar. Lo que habíamos aprendido en clase hasta ese momento solamente cubría al primer avance del proyecto, y todo lo demás lo tuvimos que aprender sobre la marcha. Por lo tanto, en las etapas iniciales fue cuando más dificultades tuve, pues intentaba pensar no solo en lo que teníamos que hacer en ese momento, sino también en cómo afectarían nuestras decisiones en ese momento a las etapas futuras del desarrollo. Sin embargo, conforme pasó el tiempo y formé una imagen más clara de los avances a futuro, pude enfocarme mejor en cada avance individualmente. Al mismo tiempo, siento que varias de las metas adicionales que definimos al inicio del proyecto fueron buenas decisiones, porque gracias a ellas nos retamos a extender la implementación vista en clase a elementos que no cubrimos en la misma.



# Descripción del Lenguaje

## Nombre del Lenguaje

TeamPlusPlus

## Principales Características del Lenguaje

TeamPlusPlus es un lenguaje del paradigma imperativo procedural, orientado a objetos. Cuenta con los tipos primitivos enteros (int), flotantes (float) y caracteres (char). Los tipos int y float pueden ser utilizados en expresiones lógicas (and, or, not), relacionales (mayor, menor, igual, etc.) y aritméticas (suma y resta, multiplicación y división. El tipo char solamente funciona en operaciones relacionales. Adicionalmente, el lenguaje permite la definición de clases, o tipos estructurados, con atributos y métodos de tipo primitivo. Las clases incluyen funcionalidad como niveles de accesibilidad (público y privado) y herencia simple. Además, es posible declarar arreglos de una o dos dimensiones de tanto tipos primitivos como estructurados.

El lenguaje soporta la entrada y salida de datos en pantalla, así como la asignación de valores a variables. Se incluyen además dos estatutos multicondicionales (if-elif-else y switch), así como dos estatutos cíclicos pre-condicionados (while y from-to).

Finalmente, el lenguaje es modular, permitiendo la declaración de tanto funciones como métodos con tipos de retorno primitivos o void, así como una cantidad variable de parámetros de tipo primitivo.

## Listado de Errores

### Compilación

Mensaje	Descripción
[Error] Invalid operand types <type> and <type> for operator <operator>	Ocorre cuando se intenta realizar una operación donde la combinación de tipos y operador no son aceptados por el cubo semántico.
[Error] Cannot evaluate value of type <type> for conditions.	Ocorre cuando se usa un tipo de dato diferente de INT para la expresión evaluada por un estatuto condicional.
[Error] Variable <name> does	Ocorre cuando se llama una variable que no se encuentra

not exist.	en la tabla de variables.
[Error] Attribute <name> is private to class <name>.	Ocurre cuando desde una clase se llama un atributo privado de su clase padre.
[Error] Cannot use array without indexing.	Ocurre cuando no se da un índice para una variable que tiene dimensiones.
[Error] Primitive types do not have methods.	Ocurre cuando se intenta llamar un método utilizando una variable de tipo primitivo.
[Error] Function <name> does not exist.	Ocurre cuando se llama una función que no se encuentra en el directorio de funciones.
[Error] Function <name> is private to class <name>.	Ocurre cuando desde una clase se llama un método privado de su clase padre.
[Error] Functions must return primitive types when used in expressions.	Ocurre cuando se realiza una llamada a función sin valor de retorno dentro de una expresión.
[Error] Invalid operand type <type> for operator <operator>.	Ocurre cuando se usa un tipo de dato no compatible con un operador unario.
[Error] Cannot assign values of structured type.	Ocurre cuando se intenta asignar un valor de un tipo estructurado a una variable.
[Error] Type mismatch. Cannot assign value of type <type> to variable of type <type>.	Ocurre cuando no coincide el tipo de dato de una variable con el tipo de dato del valor a asignar.
[Error] Array initialization is not allowed.	Ocurre cuando se intenta asignar un valor inicial en la declaración de una variable dimensionada.
[Error] Type mismatch. Cannot return value of type <type> from function with return type <type>.	Ocurre cuando un estatuto de retorno intenta retornar un tipo de dato distinto al de la función.
[Error] Cannot read data for structured types.	Ocurre cuando se intenta utilizar una variable de tipo estructurado en un estatuto de lectura.
[Error] Cannot print value of	Ocurre cuando se intenta imprimir una variable de tipo

a structured type.	estructurado.
[Error] Cannot assign values of structured type.	Ocorre cuando se intenta asignar un valor de tipo estructurado a una variable.
[Error] Cannot create switch statement with type <type>.	Ocorre cuando se utiliza un tipo de dato incompatible en la expresión inicial de un estatuto switch.
[Error] Invalid return statement in void function.	Ocorre cuando se encuentra un estatuto de retorno en una función void.
[Error] Missing return statement outside of non-linear statements in non-void function <name>.	Ocorre cuando una función con valor de retorno no contiene un estatuto de retorno fuera de estatutos condicionales.
[Error] Too many arguments given for function <name>.	Ocorre cuando una función recibe más argumentos que los esperados.
[Error] Expected argument of type <type> but received type <type>.	Ocorre cuando el tipo de dato de un argumento recibido no coincide con el tipo de dato del parámetro correspondiente.
[Error] Not enough arguments given for function <name>. Expected <amount> arguments but received <amount>.	Ocorre cuando una función recibe menos argumentos que los esperados.
[Error] Cannot index array with value of type <type>.	Ocorre cuando se intenta un valor de tipo diferente a INT para indexar una variable dimensionada.
[Error] Cannot index a variable with no dimensions.	Ocorre cuando se intenta indexar una variable no dimensionada.
[Error] Mismatched dimensions for variable <name>	Ocorre cuando se indexa una variable dimensionada con la cantidad incorrecta de dimensiones.
[Error] Function <name> already declared.	Ocorre cuando se intenta declarar una función que ya existe.
[Error] Variable <name> already declared.	Ocorre cuando se intenta declarar una variable que ya existe.

[Error] Class <name> already declared.	Ocorre cuando se intenta declarar una clase que ya existe.
[Error] Class <name> inherits from undeclared class <name>.	Ocorre cuando se intenta heredar de una clase que no existe.
[Error] Class <name> is undefined.	Ocorre cuando se intenta instanciar un objeto de una clase que no existe.
[Error] Dimensions must be greater than zero.	Ocorre cuando se intenta declarar una variable dimensionada donde una dimensión tiene 0 o menos espacios.
[Error] Cannot get address for variable of type <type>.	Ocorre cuando se intenta instanciar una variable de un tipo que no existe.
[Error] Address limit for variables of type <type> exceeded in current context.	Ocorre cuando se intenta instanciar una variable de un tipo cuyo límite de memoria ha sido excedido.
[Error] Address limit for pointers exceeded in current context.	Ocorre cuando se llega al límite de pointers que el lenguaje permite declarar en un contexto.
[Error] Address limit for instances exceeded in current context.	Ocorre cuando se llega al límite de instancias que el lenguaje permite declarar en un contexto.

## Ejecución

Mensaje	Descripción
[Error] Variable not initialized.	Ocorre cuando se intenta utilizar una variable que todavía no tiene valor en una expresión.
[Error] Cannot perform division by zero.	Ocorre cuando se intenta realizar una división por 0.
[Error] Wrong input type. Expected input of type <type>.	Ocorre cuando se lee una entrada de diferente tipo a la variable que se está leyendo.

[Error] Index out of bounds,	Ocorre cuando el índice de una variable dimensionada excede el tamaño de la dimensión correspondiente.
------------------------------	--

# Descripción del Compilador

## Equipo de Cómputo, Lenguaje y Utilerías

Archivo	Lenguaje	Librerías utilizadas
TeamPlusPlus.py	Python 3.7	antlr4 sys
CustomListener.py	Python 3.7	antlr4 sys copy
DirGen.py	Python 3.7	-
QuadrupleList.py	Python 3.7	-
VirtualMemory.py	Python 3.7	-
Utils	Python 3.7	Enum copy

Equipo de Cómputo		
Equipo	Equipo 1	Equipo 2
<b>Sistema operativo</b>	Windows 10 x64-bit	Windows 10 x64-bit
<b>CPU</b>	i7-8750H @ 2.2 GHz	AMD Ryzen 5 3600 6-Core 3.60GHz
<b>Memoria</b>	16 GB RAM	16 GB RAM



# Análisis Léxico

## Patrones de Construcción

<b>cteInt</b>	[0-9]+
<b>cteFloat</b>	cteInt (\. cteInt)?
<b>cteChar</b>	'.'
<b>cteString</b>	".*"
<b>id</b>	[a-zA-Z]+[a-zA-Z0-9_]*
<b>comment</b>	#. * \n

## Tokens

<b>program</b>	program	<b>default</b>	default	<b>or</b>	or
<b>int</b>	int	<b>func</b>	func	<b>not</b>	not
<b>float</b>	float	<b>return</b>	return	<b>void</b>	void
<b>char</b>	char	<b>read</b>	read	<b>cteInt</b>	<a href="#">regex</a>
<b>vars</b>	vars	<b>print</b>	print	<b>cteFloat</b>	<a href="#">regex</a>
<b>class</b>	class	<b>while</b>	while	<b>cteChar</b>	<a href="#">regex</a>
<b>inherits</b>	inherits	<b>from</b>	from	<b>cteString</b>	<a href="#">regex</a>
<b>main</b>	main	<b>to</b>	to	<b>id</b>	<a href="#">regex</a>
<b>if</b>	if	<b>attributes</b>	attributes	<b>comment</b>	<a href="#">regex</a>
<b>else</b>	else	<b>methods</b>	methods		
<b>elif</b>	elif	<b>public</b>	public		
<b>switch</b>	switch	<b>private</b>	private		
<b>case</b>	case	<b>and</b>	and		

<b>colon</b>	:	<b>rightParenthesis</b>	)	<b>div</b>	∕
<b>comma</b>	,	<b>leftBracket</b>	\[	<b>equals</b>	==
<b>dot</b>	\.	<b>rightBracket</b>	]	<b>greaterThan</b>	>
<b>assign</b>	=	<b>semiColon</b>	;	<b>lessThan</b>	<
<b>leftBrace</b>	\{	<b>plus</b>	+	<b>greaterEquals</b>	>=
<b>rightBrace</b>	}	<b>minus</b>	-	<b>lessEquals</b>	<=
<b>leftParenthesis</b>	\(	<b>mult</b>	\*	<b>different</b>	!=

## Análisis Sintáctico

`PROGRAM`  $\rightarrow$  `program id semiColon PROGRAM_A PROGRAM_B PROGRAM_C MAIN`

`PROGRAM_A`  $\rightarrow$  `CLASSES` |  $\epsilon$

`PROGRAM_B`  $\rightarrow$  `VARS` |  $\epsilon$

`PROGRAM_C`  $\rightarrow$  `FUNCTIONS` |  $\epsilon$

`CLASSES`  $\rightarrow$  `class id CLASSES_A leftBrace CLASSES_B CLASSES_C rightBrace semiColon CLASSES_D`

`CLASSES_A`  $\rightarrow$  `inherits id` |  $\epsilon$

`CLASSES_B`  $\rightarrow$  `CVARS` |  $\epsilon$

`CLASSES_C`  $\rightarrow$  `CFUNCS` |  $\epsilon$

`CLASSES_D`  $\rightarrow$  `CLASSES` |  $\epsilon$

`VARS`  $\rightarrow$  `vars VARS_A`

`VARS_A`  $\rightarrow$  `VARS_B INIT VARS_C semiColon VARS_D`

$\text{VARS\_B} \rightarrow \text{id} \mid \text{TYPE}$

$\text{VARS\_C} \rightarrow \text{comma INIT VARS\_C} \mid \varepsilon$

$\text{VARS\_D} \rightarrow \text{VARS\_A} \mid \varepsilon$

$\text{CVARS} \rightarrow \text{attributes CVARS\_A}$

$\text{CVARS\_A} \rightarrow \text{LEVEL TYPE CINIT CVARS\_B semiColon CVARS\_C}$

$\text{CVARS\_B} \rightarrow \text{comma CINIT CVARS\_B} \mid \varepsilon$

$\text{CVARS\_C} \rightarrow \text{CVARS\_A} \mid \varepsilon$

$\text{INIT} \rightarrow \text{id INIT\_A INIT\_C}$

$\text{INIT\_A} \rightarrow \text{leftBracket cteInt INIT\_B rightBracket} \mid \varepsilon$

$\text{INIT\_B} \rightarrow \text{comma cteInt} \mid \varepsilon$

$\text{INIT\_C} \rightarrow \text{assign EXPRESSION} \mid \varepsilon$

$\text{CINIT} \rightarrow \text{id INIT\_A}$

$\text{VAR} \rightarrow \text{id VAR\_A VAR\_C}$

$\text{VAR\_A} \rightarrow \text{leftBracket EXPRESSION VAR\_B rightBracket} \mid \varepsilon$

$\text{VAR\_B} \rightarrow \text{comma EXPRESSION} \mid \varepsilon$

$\text{VAR\_C} \rightarrow \text{dot ATTR} \mid \varepsilon$

$\text{ATTR} \rightarrow \text{id VAR\_A}$

$\text{FUNCTIONS} \rightarrow \text{FUNCTION FUNCTIONS\_A}$

$\text{FUNCTIONS\_A} \rightarrow \text{FUNCTION FUNCTIONS\_A} \mid \varepsilon$

CFUNCTIONS  $\rightarrow$  methods LEVEL FUNCTION CFUNCTIONS\_A

CFUNCTIONS\_A  $\rightarrow$  LEVEL FUNCTION CFUNCTIONS\_A  $| \epsilon$

FUNCTION  $\rightarrow$  func FUNCTION\_A id leftParenthesis FUNCTION\_B  
rightParenthesis FUNBLOCK

FUNCTION\_A  $\rightarrow$  TYPE  $|$  void

FUNCTION\_B  $\rightarrow$  TYPE id FUNCTION\_C  $| \epsilon$

FUNCTION\_C  $\rightarrow$  comma TYPE id FUNCTION\_C  $| \epsilon$

MAIN  $\rightarrow$  main leftParenthesis rightParenthesis FUNBLOCK

FUNBLOCK  $\rightarrow$  leftBrace FUNBLOCK\_A FUNBLOCK\_B rightBrace

FUNBLOCK\_A  $\rightarrow$  VARS  $| \epsilon$

FUNBLOCK\_B  $\rightarrow$  STATEMENT FUNBLOCK\_B  $| \epsilon$

TYPE  $\rightarrow$  int  $|$  float  $|$  char

LEVEL  $\rightarrow$  public  $|$  private

STATEMENT  $\rightarrow$  ASSIGNMENT  $|$  FUNCALL semiColon  $|$  RETURN  $|$  READ  $|$  PRINT  $|$   
CONDITION  $|$  LOOP

ASSIGNMENT  $\rightarrow$  VAR assign EXPRESSION semiColon

FUNCALL  $\rightarrow$  FUNCALL\_A id leftParenthesis FUNCALL\_B rightParenthesis

FUNCALL\_A  $\rightarrow$  VAR dot  $| \epsilon$

FUNCALL\_B  $\rightarrow$  EXPRESSION FUNCALL\_C  $| \epsilon$

FUNCALL\_C  $\rightarrow$  comma EXPRESSION FUNCALL\_C  $| \epsilon$

RETURN → return leftParenthesis EXPRESSION rightParenthesis semiColon

READ → read leftParenthesis VAR READ\_A rightParenthesis semiColon

READ\_A → comma VAR READ\_A |  $\epsilon$

PRINT → print leftParenthesis PRINT\_A PRINT\_B rightParenthesis  
semiColon

PRINT\_A → EXPRESSION | cteString

PRINT\_B → comma PRINT\_A PRINT\_B |  $\epsilon$

BLOCK → leftBrace BLOCK\_A rightBrace

BLOCK\_A → STATEMENT BLOCK\_A |  $\epsilon$

CONDITION → IFELSE | SWITCH

IFELSE → if leftParenthesis EXPRESSION rightParenthesis BLOCK IFELSE\_A  
IFELSE\_B

IFELSE\_A → ELIF |  $\epsilon$

IFELSE\_B → else BLOCK |  $\epsilon$

ELIF → elif leftParenthesis EXPRESSION rightParenthesis BLOCK ELIF\_A

ELIF\_A → elif leftParenthesis EXPRESSION rightParenthesis BLOCK ELIF\_A  
|  $\epsilon$

SWITCH → switch leftParenthesis EXPRESSION rightParenthesis leftBrace  
SWITCH\_A SWITCH\_D rightBrace

SWITCH\_A → case SWITCH\_C BLOCK SWITCH\_B

SWITCH\_B → case SWITCH\_C BLOCK SWITCH\_B |  $\epsilon$

SWITCH\_C → cteChar | cteInt

$\text{SWITCH\_D} \rightarrow \text{default BLOCK} \mid \varepsilon$

$\text{LOOP} \rightarrow \text{WLOOP} \mid \text{FLOOP}$

$\text{WLOOP} \rightarrow \text{while leftParenthesis EXPRESSION rightParenthesis BLOCK}$

$\text{FLOOP} \rightarrow \text{from VAR assign EXP to EXP BLOCK}$

$\text{EXPRESSION} \rightarrow \text{EXPRESSIO EXPRESSION\_A}$

$\text{EXPRESSION\_A} \rightarrow \text{or EXPRESSIO EXPRESSION\_A} \mid \varepsilon$

$\text{EXPRESSIO} \rightarrow \text{EXPRESS EXPRESSIO\_A}$

$\text{EXPRESSIO\_A} \rightarrow \text{and EXPRESS EXPRESSIO\_A} \mid \varepsilon$

$\text{EXPRESS} \rightarrow \text{EXP EXPRESS\_A}$

$\text{EXPRESS\_A} \rightarrow \text{RELOP EXP} \mid \varepsilon$

$\text{EXP} \rightarrow \text{TERM EXP\_A}$

$\text{EXP\_A} \rightarrow \text{SUMOP TERM EXP\_A} \mid \varepsilon$

$\text{TERM} \rightarrow \text{FACTOR TERM\_A}$

$\text{TERM\_A} \rightarrow \text{MULOP FACTOR TERM\_A} \mid \varepsilon$

$\text{FACTOR} \rightarrow \text{FACTOR\_A FACTOR\_B}$

$\text{FACTOR\_A} \rightarrow \text{UNOP} \mid \varepsilon$

$\text{FACTOR\_B} \rightarrow \text{leftParenthesis EXPRESSION rightParenthesis} \mid \text{VALUE}$

$\text{UNOP} \rightarrow \text{not} \mid \text{plus} \mid \text{minus}$

RELOP → equals | greaterThan | lessThan | greaterEquals | lessEquals | different

SUMOP → plus | minus

MULOP → mult | div

VALUE → VAR | cteInt | cteFloat | cteChar | FUNCALL

## Análisis Semántico y Generación de Código Intermedio

### Códigos de Operación

<b>OR</b>	0	<b>SUB</b>	11	<b>GOSUB</b>	22
<b>AND</b>	1	<b>POS</b>	12	<b>PARAMETER</b>	23
<b>EQ</b>	2	<b>NEG</b>	13	<b>ERA</b>	24
<b>GT</b>	3	<b>NOT</b>	14	<b>ENDFUNC</b>	25
<b>LT</b>	4	<b>ASSIGN</b>	15	<b>GOMAIN</b>	26
<b>GTE</b>	5	<b>FF</b>	16	<b>VERIFY</b>	27
<b>LTE</b>	6	<b>RETURN</b>	17	<b>POINT</b>	28
<b>DIFF</b>	7	<b>READ</b>	18	<b>INST</b>	29
<b>MULT</b>	8	<b>PRINT</b>	19	<b>METHOD</b>	30
<b>DIV</b>	9	<b>GOTO</b>	20		
<b>SUM</b>	10	<b>GOTOF</b>	21		

## Direcciones Virtuales

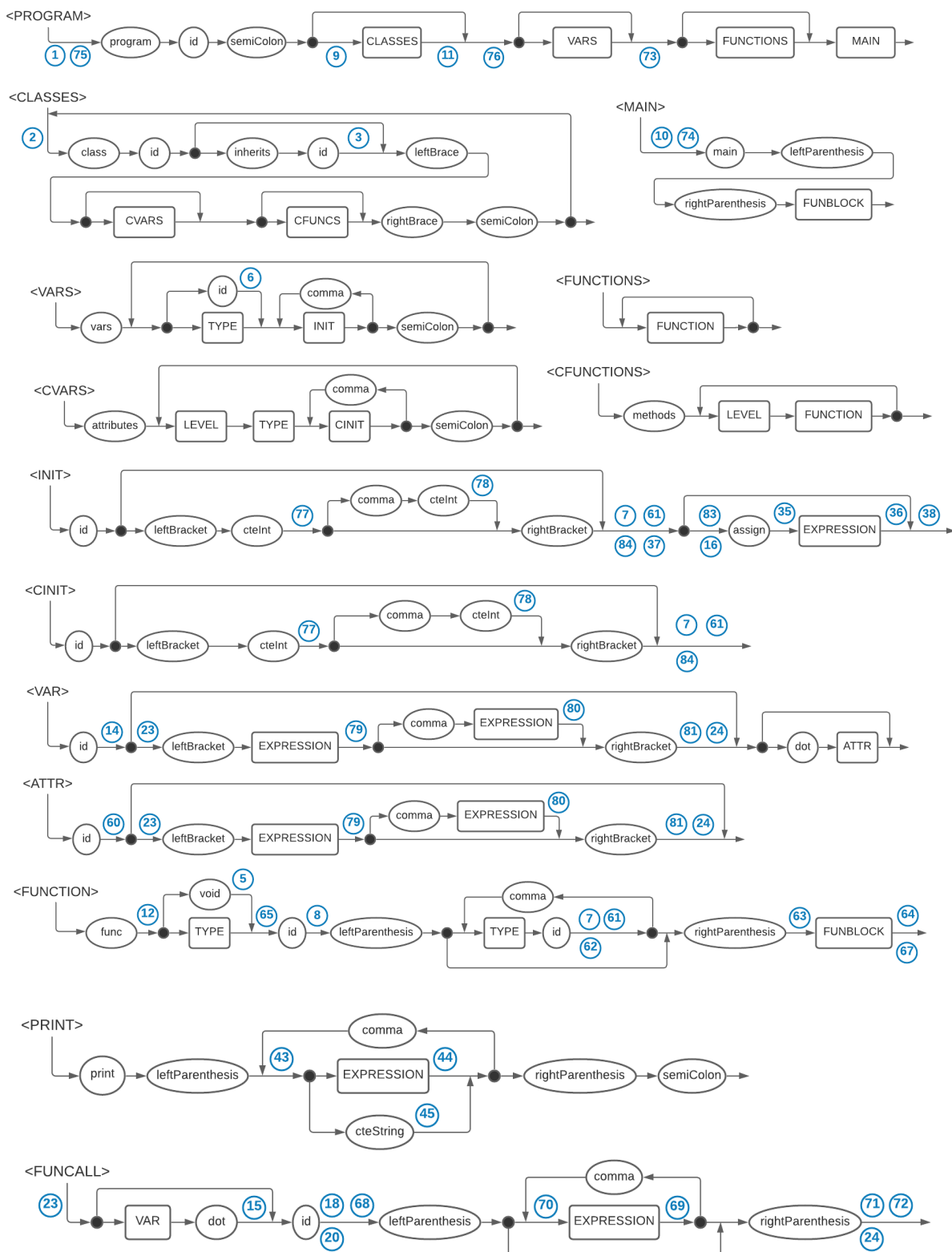
<b>Contexto</b>	<b>Inicio</b>	<b>Fin</b>
Global*	0	999
Local*	1000	1999
Temporal*	2000	2999
Constante*	3000	3999
Pointer	4000	4999
Instancias Locales	5000	5999
Instancias Globales	6000	6999
Atributos*	7000	7999

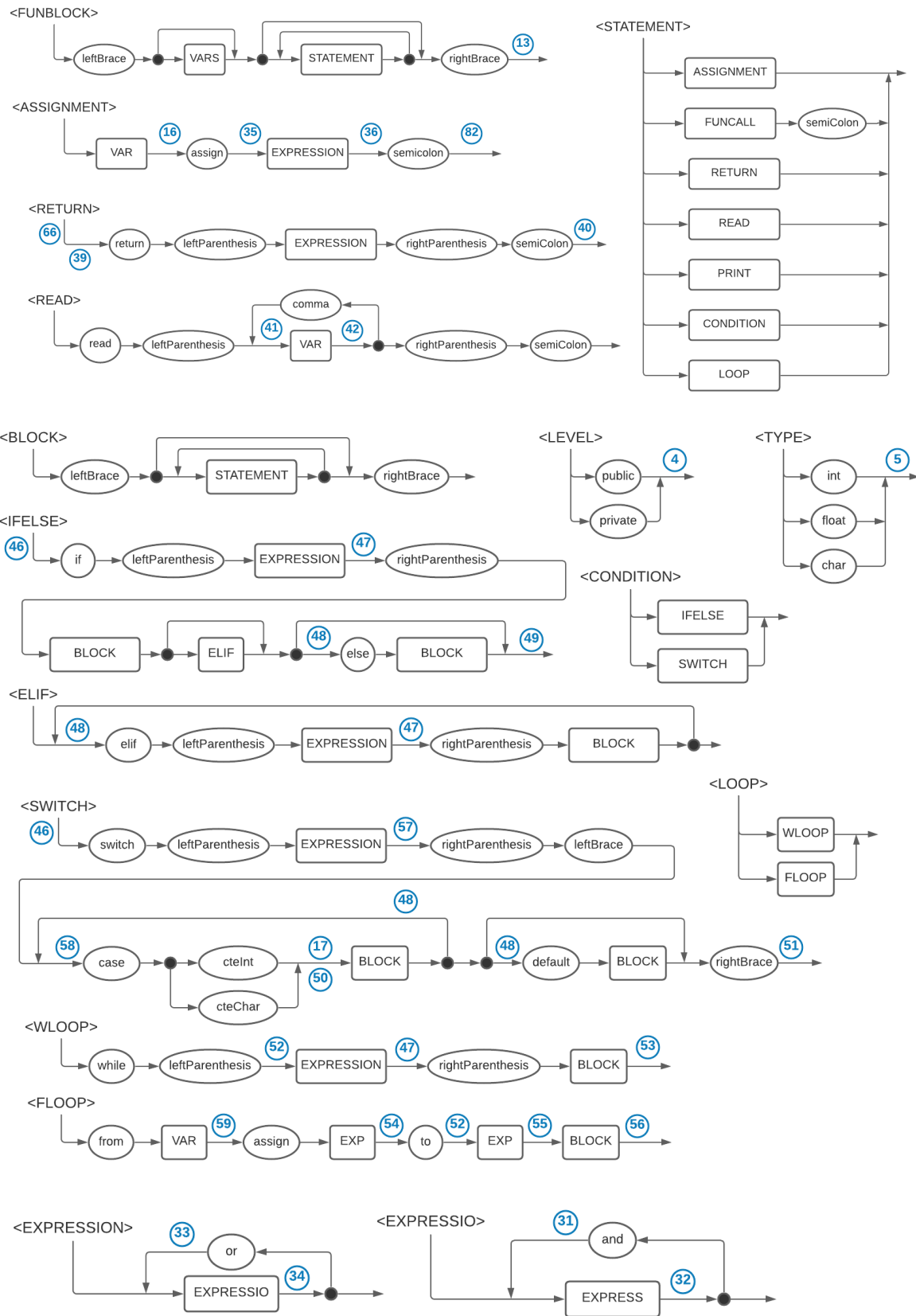
\* Los contextos marcados están divididos por tipo de la siguiente manera:

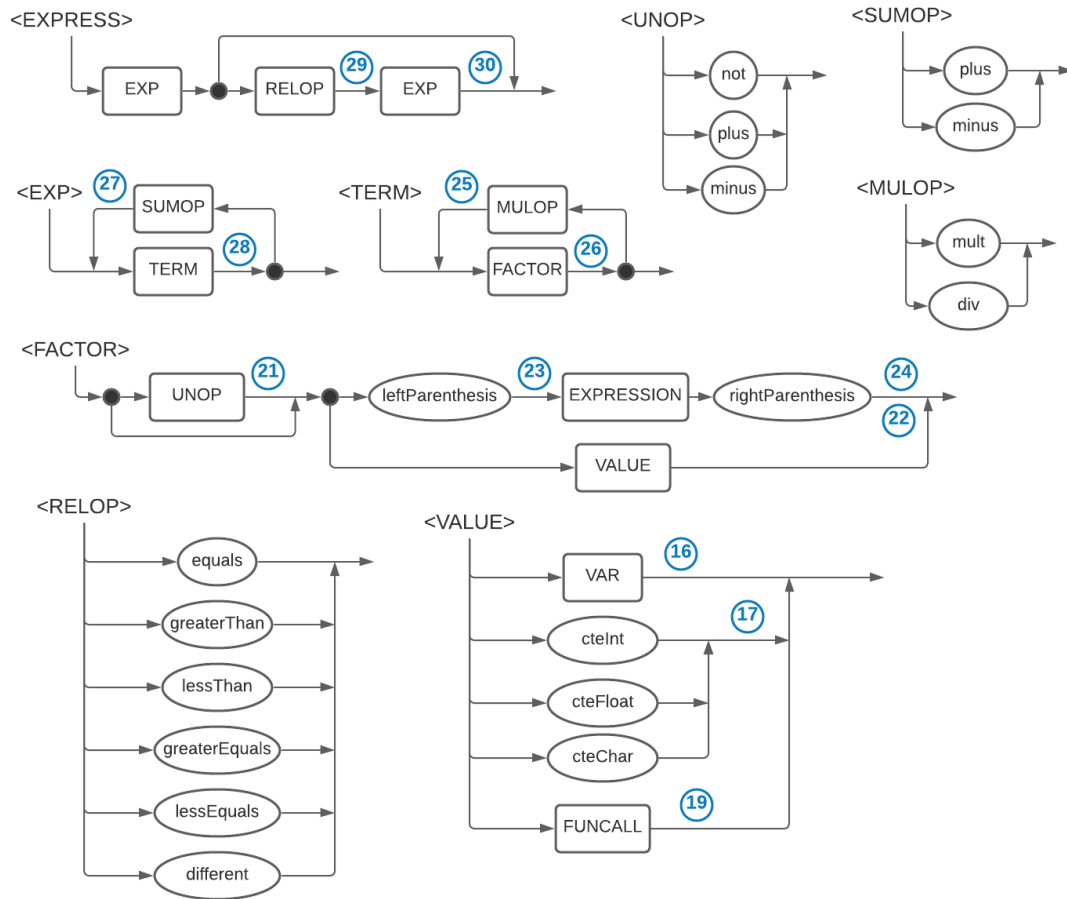
<b>Tipo de dato</b>	<b>Inicio</b>	<b>Fin</b>
Int	0	99
Float	100	199
Char	200	299



# Diagramas de Sintaxis y Puntos Neurálgicos







Número	Descripción
1	Crea el directorio general de clases y funciones, mete la función 'global' al directorio de funciones.
2	Verifica que la clase no exista y la agrega al directorio de clases.
3	Verifica que la clase que hereda exista. Registra la clase padre en la clase actual.
4	Cambia el nivel de accesibilidad actual.
5	Cambia el tipo actual.
6	Verifica que la clase exista.
7	Verifica que la variable no exista en su contexto la agrega a la tabla de variables del contexto actual.

8	Verifica que la función no exista y la agrega al directorio de funciones correspondiente. Crea una variable global para la función.
9	Actualiza flag que indica que el contexto actual es una clase.
10	Agrega 'main' al directorio de funciones.
11	Actualiza flag que indica que el contexto actual es una función. Reinicia los valores que se utilizan en la declaración de clases.
12	Prende la flag que dice si estamos dentro de una función.
13	Apaga la flag que dice si estamos dentro de una función.
14	Verifica que la variable exista y sea accesible en el contexto actual. Mete la variable a la pila de variables.
15	Prende la flag que dice si estamos dentro de un método.
16	Agrega la dirección de la variable actual y su tipo a la pila de operandos.
17	Agrega para el valor leído una dirección constante y su tipo a la pila de operandos.
18	Verifica que la función exista y sea accesible en el contexto actual.
19	Genera un cuádruplo de asignación de la dirección de retorno de la función a una dirección temporal.
20	Apaga la flag que dice si estamos dentro de un método.
21	Mete el operador unario a la pila de operadores.
22	Si en el tope de pila de operadores hay un operador unario, genera un cuádruplo para esa operación.
23	Mete un fondo falso a la pila de operadores.
24	Saca un fondo falso de la pila de operadores.
25	Mete el operador de '*' o '/' a la pila de operadores.
26	Si en el tope de la pila de operadores hay un '*' o '/', genera un cuádruplo para esa operación.
27	Mete el operador de '+' o '-' a la pila de la pila de operadores.

28	Si en el tope de la pila de operadores hay un '+' o '-', genera un cuádruplo para esa operación.
29	Mete el operador relacional a la pila de operadores.
30	Si en el tope de la pila de operadores hay un operador relacional, genera un cuádruplo para esa operación.
31	Mete el operador 'and' a la pila de operadores.
32	Si en el tope de la pila de operadores hay un 'and', genera un cuádruplo para esa operación.
33	Mete el operador 'or' a la pila de operadores.
34	Si en el tope de la pila de operadores hay un 'or', genera un cuádruplo para esa operación.
35	Mete el operador '=' a la pila de operadores.
36	Si en el tope de la pila de operadores hay un '=', genera un cuádruplo para esa operación.
37	Almacena la variable en la pila de variables.
38	Saca la variable de la pila de variables.
39	Mete el operador 'return' a la pila de operadores.
40	Si en el tope de la pila de operadores hay un 'return', genera un cuádruplo para esa operación.
41	Mete el operador 'read' a la pila de operadores.
42	Si en el tope de la pila de operadores hay un 'read', genera un cuádruplo para esa operación.
43	Mete el operador 'print' a la pila de operadores.
44	Si en el tope de la pila de operadores hay un 'print', genera un cuádruplo para esa operación, tomando el resultado de la expresión.
45	Si en el tope de la pila de operadores hay un 'print', genera el cuádruplo de esa operación, tomando el letrero.
46	Mete un fondo falso a la pila de saltos.

47	Mete el índice del cuádruplo actual en la pila de saltos y genera un cuádruplo de GOTOF con el tope de la pila de operandos.
48	Genera un cuádruplo de GOTO. Saca de la pila de saltos el tope y actualiza el cuádruplo correspondiente. Mete el índice anterior a la pila de saltos.
49	Mientras no vea un fondo falso, saca de la pila de saltos el tope y actualiza el cuádruplo correspondiente. Saca el fondo falso cuando lo encuentre.
50	Genera un cuádruplo de comparación entre la expresión del switch y el caso actual. Guarda el índice actual en la pila de saltos y crea un GOTOF.
51	Hace lo mismo que el punto 49. Hace pop a la pila de variables recurrentes.
52	Guarda el índice actual en la pila de saltos.
53	Genera un GOTO. Saca el tope de la pila de saltos y actualiza el cuádruplo correspondiente. Actualiza el GOTO nuevo con el tope de la pila de saltos.
54	Genera un cuádruplo de asignación con la variable y el tope de la pila de operandos. Guarda la variable nuevamente en la pila de operandos.
55	Genera un cuádruplo de ' $\leq$ ' con la variable y la segunda expresión. Mete el índice de cuádruplo actual en la pila de saltos y genera un GOTOF.
56	Genera cuádruplos de '+', '=' y GOTO. Saca el tope de la pila de saltos y actualiza ese cuádruplo. Actualiza GOTO con el tope de la misma pila.
57	Saca el tope de la pila de operandos y lo guarda en la pila de variables recurrentes.
58	Mete un operando equivalente al del tope de la pila de variables recurrentes a la pila de operandos.
59	Guarda la variable en la pila de variables recurrentes. Mete un operando con esa variable a la pila de operandos.
60	Verifica que el atributo exista y sea accesible en el contexto actual. Mete la variable a la pila de variables.
61	Actualiza el contador de variables del contexto actual con base en el tipo de la variable. Asigna una dirección virtual a la variable.
62	Mete el parámetro y su tipo a la tabla de parámetros de la función.

63	Guarda en la función el contador de cuádruplos actual.
64	Genera un cuádruplo de ENDFUNC.
65	Dependiendo del tipo de retorno, cambia la flag que verifica la presencia de 'return's en la función.
66	Con base en su valor anterior, actualiza la flag que verifica la presencia de 'return's en la función.
67	Hace la verificación del return de la función con base en el valor de la flag.
68	Genera un cuádruplo de ERA con el tamaño de memoria que requiere la función. Inicializa el contador de parámetros a 0.
69	Verifica si el tipo del argumento coincide con el del parámetro. Genera un cuádruplo de PARAMETER.
70	Incrementa el contador de parámetros en 1.
71	Verifica que la cantidad de argumentos y parámetros coincidan. Actualiza variables globales para la dirección de retorno de la función y su tipo.
72	Genera un cuádruplo de GOSUB al cuádruplo inicial de la función.
73	Genera un cuádruplo de GOMAIN y guarda el índice actual en la pila de saltos.
74	Saca el tope de la pila de saltos y actualiza el GOMAIN.
75	Agrega un cuádruplo de GOTO al inicio de la declaración de las variables globales.
76	Saca de la pila de saltos el índice del cuádruplo de GOTO y lo actualiza con el contador actual.
77	Guarda la primera dimensión de la variable.
78	Guarda la segunda dimensión de la variable.
79	Genera el cuádruplo de VERIFY con el resultado de la expresión y la primera dimensión del arreglo.
80	Genera el cuádruplo de VERIFY con el resultado de la expresión y la segunda dimensión del arreglo.

81	Genera los cuádruplos para obtener la dirección resultante de la casilla del arreglo (pointer).
82	Hace pop de la pila de variables y también de la pila de pointers.
83	Verifica que la variable en la asignación inicial no tenga dimensiones.
84	Crea un cuádruplo de instanciación para variables de tipo estructurado.

## Cubo Semántico

**ASSIGN:** =

**LOGOP:** and, or, not

**RELOP:** ==, >, <, >=, <=, !=

**SUMOP:** +, -

**MULOP:** \*, /

	INT	INT
INT	ASSIGN, LOGOP, RELOP, SUMOP, MULOP	

	INT	FLOAT
INT	LOGOP, RELOP	
FLOAT	SUMOP, MULOP	

	FLOAT	INT
FLOAT	SUMOP, MULOP	
INT	LOGOP, RELOP	



	<b>FLOAT</b>	<b>FLOAT</b>
<b>FLOAT</b>	ASSIGN, SUMOP, MULOP	
<b>INT</b>	LOGOP, RELOP	

	<b>CHAR</b>	<b>CHAR</b>
<b>CHAR</b>	ASSIGN	
<b>INT</b>	RELOP	

## Administración de Memoria en Compilación

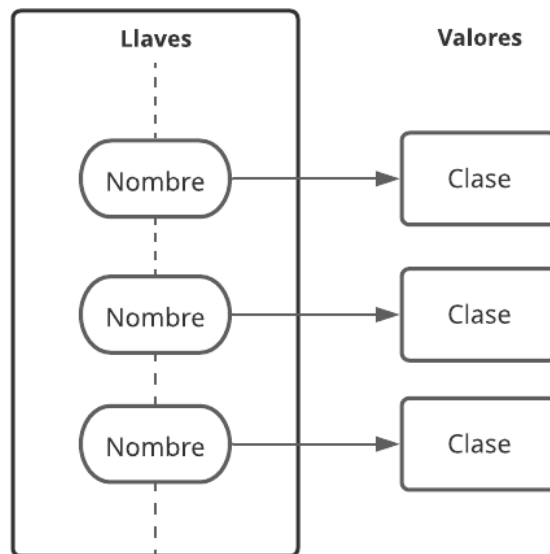
### Directorio General

El Directorio General es el objeto principal que se utiliza para el manejo de declaraciones de funciones y variables. Internamente, contiene a una instancia del Directorio de Funciones y otra del Directorio de Clases.

<b>Directorio General</b>	Directorio de Funciones
	Directorio de Clases

### Directorio de Clases

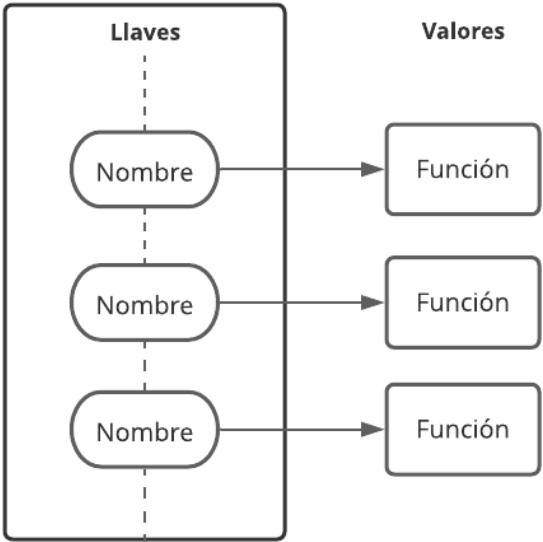
El Directorio de Clases es un diccionario que almacena toda la información sobre las clases que se definen en un programa. Tiene como llaves los nombres de las clases declaradas, y como valores, un objeto de Clase que contiene toda la información relevante a la clase.



<b>Clase</b>	Nombre de la clase
	Nombre de la clase padre
	Métodos (Directorio de Funciones)
	Atributos (Tabla de Variables)
	Address Manager

## Directorio de Funciones

El Directorio de Funciones es un diccionario que almacena toda la información sobre funciones que se definen en el mismo contexto del programa. Esto puede ser el contexto global, para las funciones globales en el programa, o un contexto local a una clase, para los métodos de la misma. Tiene como llaves los nombres de las funciones declaradas, y como valor un objeto de Función que contiene toda la información relevante a la función. Para propósitos de este lenguaje, internamente se representa al contexto global como una función de nombre "global".

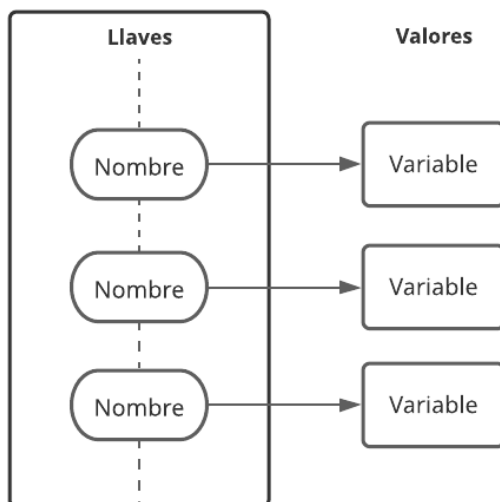


<b>Función</b>	Nombre de la función	
	Nivel de accesibilidad	
	Tipo de retorno	
	Clase a la que pertenece originalmente	
	Dirección de su variable global de retorno	
	Tabla de Variables	
	Lista de Parámetros	Nombre
		Tipo
	Dirección de su primer cuádruplo	
	Function Address Manager	

Tabla de Variables

La Tabla de Variables es un diccionario que almacena toda la información sobre funciones que se definen en el mismo contexto del programa. Esto puede ser

dentro de una función o método para sus variables locales, dentro de una clase para sus atributos.



<b>Variable</b>	Nombre de la variable
	Tipo de la variable
	Nombre del tipo de la variable (si es estructurado)
	Dirección de la variable
	Nivel de accesibilidad
	Clase a la que pertenece originalmente
	Tamaño de su primera dimensión
	Tamaño de su segunda dimensión

## Address Managers

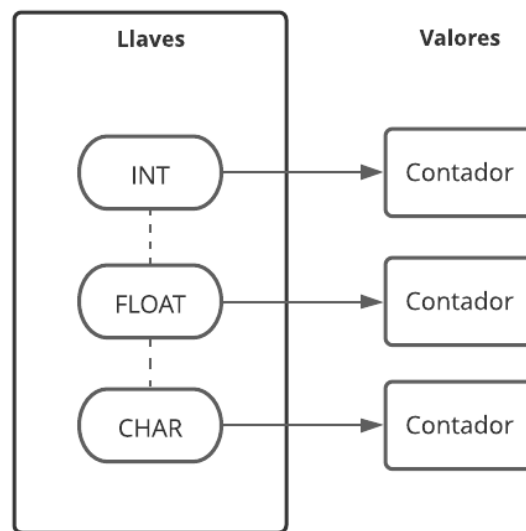
### Function Address Manager

Este Address Manager se utiliza para el manejo de direcciones dentro de todas las funciones. Sirve como una interfaz, conteniendo todos los Address Managers necesarios para el manejo de contextos locales, que incluyen variables, temporales, pointers e instancias locales.

<b>Function Address Manager</b>	Address Manager (Local)
	Address Manager (Temporal)
	Pointer Manager
	Address Manager (Instancia Local)

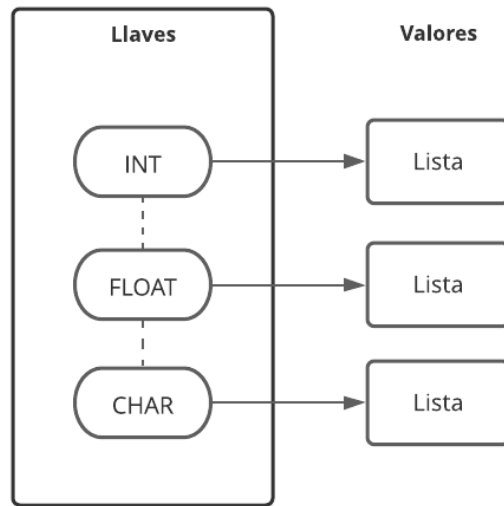
### Address Manager para Contextos Divididos por Tipo

Este tipo de Address Manager se utiliza para los contextos Global, Local, Constante y Atributo. Cuando se solicita una dirección, dependiendo del tipo se accede al contador correspondiente y se regresa su siguiente valor.



### Address Manager para Direcciones Temporales

Adicionalmente al diccionario de contadores anterior, el Address Manager para Direcciones Temporales contiene un diccionario de listas de direcciones temporales liberadas, que se utiliza para el reciclaje de direcciones temporales. Cuando este Address Manager recibe una solicitud de una dirección, primero revisa si existe una dirección liberada para el tipo correspondiente antes de generar una nueva.



### Address Manager para Pointers

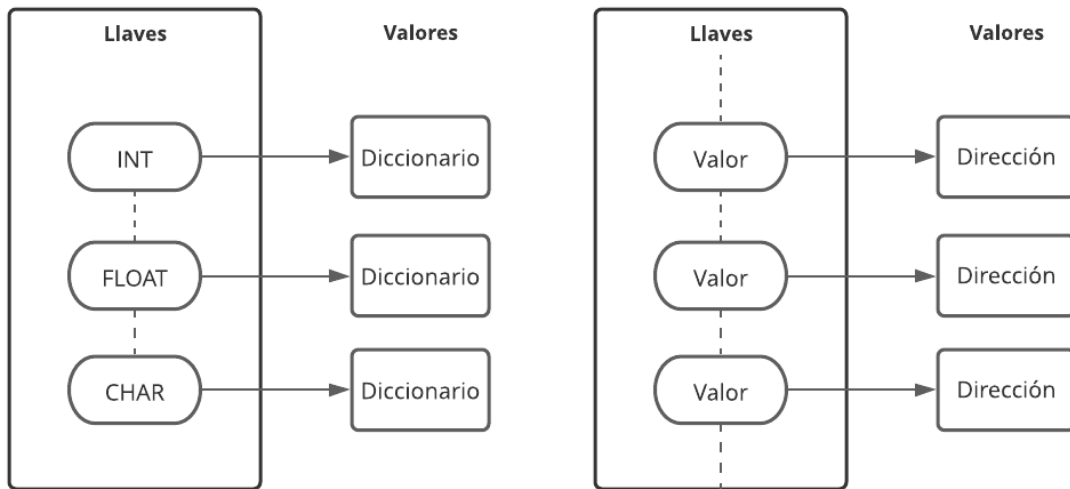
El Address Manager para Pointers se distingue de los anteriores debido a que las direcciones no se dividen por tipo. Se tiene un solo contador, a partir del cual se generan las direcciones solicitadas.

### Address Manager para Instancias

El Address Manager para Instancias es muy similar al que se diseñó para pointers. Se usa para obtener direcciones para instancias en el contexto global y local. La única diferencia que tiene con el de pointers es que aquí se permite solicitarle más de una dirección a la vez.

### Tabla de Constantes

La Tabla de Constantes es un diccionario utilizado para el manejo de constantes en un programa y es contenida por un Address Manager del contexto Constante. Tiene como llave cada uno de los tipos primitivos del lenguaje, y como valor un diccionario que a su vez tiene como llaves el valor de una constante del tipo correspondiente al diccionario, y como valor la dirección correspondiente a ese valor.



## Lista de Cuádruplos

La Lista de Cuádruplos es la estructura donde se almacena el código intermedio generado por el compilador. La Lista de Cuádruplos está implementada como una fila para asegurar que se mantenga el orden de los cuádruplos.

<b>Cuádruplo</b>	Índice
	Operador
	Operando Izquierdo
	Operando Derecho
	Resultado

# Descripción de la Máquina Virtual

## Equipo de Cómputo, Lenguaje y Utilerías

Archivo	Lenguaje	Librerías utilizadas
VirtualMachine.py	Python 3.7	sys operator
MachineMemory.py	Python 3.7	-

Equipo de Cómputo		
Equipo	Equipo 1	Equipo 2
<b>Sistema operativo</b>	Windows 10 x64-bit	Windows 10 x64-bit
<b>CPU</b>	i7-8750H @ 2.2 GHz	AMD Ryzen 5 3600 6-Core 3.60GHz
<b>Memoria</b>	16 GB RAM	16 GB RAM

## Administración de Memoria en Ejecución

### Máquina Virtual

La Máquina Virtual se encarga de realizar todo el proceso de ejecución para un programa que ya terminó la etapa de compilación. Desde el punto de vista de la administración de memoria, la Máquina Virtual recibe directamente del compilador el Directorio General, la Lista de Cuádruplos y la Tabla de Constantes generada por el compilador. Además, contiene otras estructuras que se utilizan para el manejo de direcciones simuladas.



<b>Máquina Virtual</b>	Directorio General
	Lista de Cuádruplos
	Tabla de Constantes para Ejecución
	Memoria de Máquina (Global)
	Lista de Instancias (Global)
	Pila de Ejecución
	Pila de Ejecución Temporal
	Pila de Instancias Activas

### Pila de Ejecución

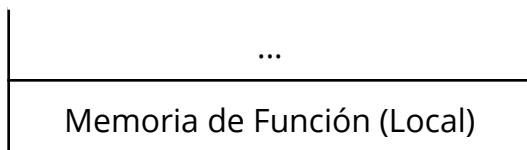
La Pila de Ejecución es la estructura que permite manejar contextos locales en la memoria de ejecución. Cada elemento de la pila es una instancia de Memoria de Función específica a la llamada de función con la que se creó. Cada vez que una función termina su ejecución, su memoria correspondiente, que debe estar al tope de la pila, sale de la pila. Al inicio de la ejecución, se genera una Memoria de Función correspondiente al contexto global. Esta instancia se queda en la pila de ejecución hasta que todo el programa termine.

...
Memoria de Función (Local)
Memoria de Función (Local)
Memoria de Función (Global)

### Pila de Ejecución Temporal

La Pila de Ejecución Temporal es la estructura que permite la transferencia de información de un contexto a otro. Cuando el programa realiza una llamada a una función, una instancia de Memoria de Función correspondiente a la llamada es agregada a esta pila. Mientras está en ella, recibe información sobre sus parámetros desde la Memoria de Función que está al tope de la Pila de Ejecución.

Una vez que ha recibido toda la información necesaria, y está lista para comenzar su ejecución, la instancia de Memoria de Función sale de la Pila de Ejecución Temporal y entra a la Pila de Ejecución.



## Memoria de Función

La Memoria de Función es la estructura que representa a la memoria de un contexto local en ejecución. Se genera con base en la entrada de la función correspondiente en el Directorio General. En ella se generan las estructuras de memoria relevantes únicamente a la función en cuestión.

<b>Memoria de Función</b>	Memoria de Máquina (Local)
	Memoria de Máquina (Temporal)
	Memoria de Pointers
	Dirección de su variable global de retorno
	Índice del siguiente cuádruplo a ejecutar
	Lista de Instancias (Local)

## Lista de Instancias

La Lista de Instancias es la estructura que maneja la memoria específica a instancias de tipos estructurados en un contexto. A diferencia de otras estructuras de memoria en ejecución, la Lista de Instancias comienza estando totalmente vacía y se le agregan elementos cada vez que se realiza una declaración de una instancia en el programa.

Para referenciar valores específicos dentro de una Lista de Instancias, la Máquina Virtual recibe direcciones virtuales compuestas con el formato 50007000 o 60007000, donde la primera mitad de la dirección (5000 o 6000) se refiere a su posición dentro de la Lista de Instancias (local o global respectivamente) y la

segunda mitad de la dirección se refiere a la dirección del valor dentro de esa instancia. Usando el ejemplo de la dirección compuesta 50007000, esta se refiere al valor con dirección 000 dentro de la Memoria de Instancia que se encuentra en el índice 0 de la Lista de Instancias del contexto local actual.

Memoria de Instancia	Memoria de Instancia	Memoria de Instancia	Memoria de Instancia	...
----------------------	----------------------	----------------------	----------------------	-----

## Pila de Instancias Activas

La Pila de Instancias Activas es una estructura que permite que durante la ejecución de métodos se pueda hacer una referencia directa a un atributo de la instancia que llamó al método. Al igual que la Lista de Instancias, sus elementos son Memorias de Instancia. Cuando se llama un método, se toma la instancia que lo llama de la Lista de Instancias apropiada y se mete a la Pila de Instancias Activas.

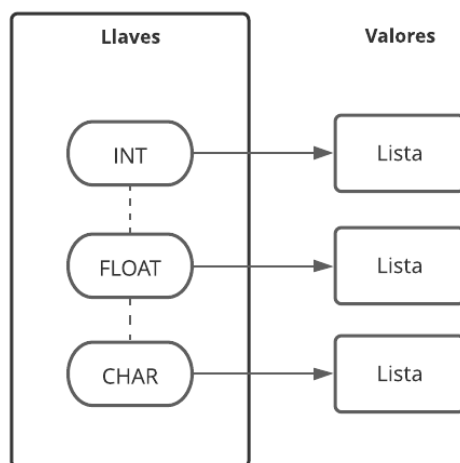
...
Memoria de Instancia
Memoria de Instancia
Memoria de Instancia

## Memoria de Máquina

La Memoria de Máquina es la estructura donde se almacenan durante ejecución todos los valores que se definen en el programa. Esta se genera a partir de una instancia de Address Manager. Consiste de un diccionario, donde las llaves son tipos de datos, y cada valor es una lista del tamaño exacto necesario con base en la cantidad de direcciones asignadas por el Address Manager durante compilación.

La Máquina Virtual preprocesa las direcciones virtuales para que las direcciones de ejecución que maneja la Memoria de Máquina sean independientes del contexto. Por ejemplo, si la Máquina Virtual recibe las direcciones 1102, 2221 y 7024, primero utiliza el primer dígito de cada dirección (1, 2 y 7) para identificar la instancia de Memoria de Máquina adecuada, y esta solamente necesita manejar el resto de la dirección (102, 221 y 024) internamente. A su vez, el primer dígito de estas

direcciones reducidas (1, 2 y 0) representan la llave del diccionario de listas, y los dígitos restantes (02, 21, 24) representan el índice dentro de la lista donde se encuentra el valor que se busca.



## Memoria de Instancia

La Memoria de Instancia es una Memoria de Máquina específicamente para representar instancias en las estructuras de datos que las manejan. Su estructura y funcionamiento es idéntico al de una Memoria de Máquina.

## Memoria de Pointers

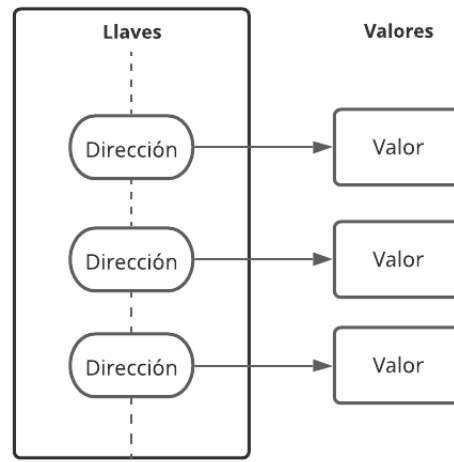
La Memoria de Pointers es una simplificación de la Memoria de Máquina. En vez de un diccionario, se utiliza una lista creada a partir de una instancia de Address Manager para Pointers. La diferencia principal con la Memoria de Máquina es que los valores que almacena son direcciones a algún otro valor.

Dirección	Dirección	Dirección	Dirección	...
-----------	-----------	-----------	-----------	-----

## Tabla de Constantes para Ejecución

La Tabla de Constantes para Ejecución es la estructura que relaciona las direcciones de valores constantes a sus valores correspondientes. Se genera a partir de la Tabla de Constantes utilizada durante el proceso de compilación, invirtiendo los valores y

las llaves de cada diccionario y después los combina para formar un solo diccionario.



### Fórmulas para Indexación de Arreglos

El lenguaje solamente permite la declaración de arreglos de una o dos dimensiones, cada una de las cuales tiene como límite inferior fijo el índice cero. Por lo tanto, nuestra implementación de las fórmulas para indexar arreglos fue simplificada de la siguiente manera:

Para una dimensión:

$$\text{Address}(\text{id}[s]) = \text{baseAddressId} + s$$

Para dos dimensiones:

$$\text{Address}(\text{id}[s_1, s_2]) = \text{baseAddressId} + s_1 * d_2 + s_2$$

# Pruebas del Funcionamiento del Lenguaje

Programa	Resultados
<pre> program search;  vars     int arr[10], mid;  func int binarySearch(int value) {     vars         int start = 0;         int end = 9;          while(start &lt;= end) {             mid = (end + start) / 2;              if (arr[mid] == value) {                 return(mid);             } elif (arr[mid] &gt; value) {                 end = mid - 1;             } else {                 start = mid + 1;             }         }          return(-1);     }  func int recursiveSearch(int value, int start, int end) {     if (start &lt;= end) {         mid = (end + start) / 2;          if (arr[mid] == value) {             return(mid);         } elif (arr[mid] &gt; value) {             return(recursiveSearch(value, start, mid-1));         } else {             return(recursiveSearch(value, mid+1, end));         }     }      return(-1); }  func int linearSearch(int value) {     vars         int i;          from i=0 to 9 {             if (arr[i] == value) {                 return(i);             }         }          return (-1);     }  main() {     vars </pre>	<p>Código Intermedio:</p> <ol style="list-style-type: none"> <li>0. GOTO, None, None, 1,</li> <li>1. GOMAIN, None, None, 77,</li> <li>2. ASSIGN, 3000, None, 1001,</li> <li>3. ASSIGN, 3001, None, 1002,</li> <li>4. LTE, 1001, 1002, 2000,</li> <li>5. GOTOF, 2000, None, 25,</li> <li>6. SUM, 1002, 1001, 2000,</li> <li>7. DIV, 2000, 3002, 2001,</li> <li>8. ASSIGN, 2001, None, 10,</li> <li>9. VERIFY, 10, 10, None,</li> <li>10. POINT, 0, 10, 4000,</li> <li>11. EQ, 4000, 1000, 2000,</li> <li>12. GOTOF, 2000, None, 15,</li> <li>13. RETURN, None, None, 10,</li> <li>14. GOTO, None, None, 24,</li> <li>15. VERIFY, 10, 10, None,</li> <li>16. POINT, 0, 10, 4001,</li> <li>17. GT, 4001, 1000, 2001,</li> <li>18. GOTOF, 2001, None, 22,</li> <li>19. SUB, 10, 3004, 2000,</li> <li>20. ASSIGN, 2000, None, 1002,</li> <li>21. GOTO, None, None, 24,</li> <li>22. SUM, 10, 3004, 2001,</li> <li>23. ASSIGN, 2001, None, 1001,</li> <li>24. GOTO, None, None, 4,</li> <li>25. NEG, 3004, None, 2000,</li> <li>26. RETURN, None, None, 2000,</li> <li>27. ENDFUNC, None, None, None,</li> <li>28. LTE, 1001, 1002, 2000,</li> <li>29. GOTOF, 2000, None, 60,</li> <li>30. SUM, 1002, 1001, 2000,</li> <li>31. DIV, 2000, 3002, 2001,</li> <li>32. ASSIGN, 2001, None, 10,</li> <li>33. VERIFY, 10, 10, None,</li> <li>34. POINT, 0, 10, 4000,</li> <li>35. EQ, 4000, 1000, 2000,</li> <li>36. GOTOF, 2000, None, 39,</li> <li>37. RETURN, None, None, 10,</li> <li>38. GOTO, None, None, 60,</li> <li>39. VERIFY, 10, 10, None,</li> <li>40. POINT, 0, 10, 4001,</li> <li>41. GT, 4001, 1000, 2001,</li> <li>42. GOTOF, 2001, None, 52,</li> <li>43. ERA, None, None, recursiveSearch,</li> <li>44. PARAMETER, 1000, None, 1000,</li> <li>45. PARAMETER, 1001, None, 1001,</li> <li>46. SUB, 10, 3004, 2000,</li> <li>47. PARAMETER, 2000, None, 1002,</li> <li>48. GOSUB, recursiveSearch, None, 28,</li> <li>49. ASSIGN, 12, None, 2001,</li> <li>50. RETURN, None, None, 2001,</li> <li>51. GOTO, None, None, 60,</li> <li>52. ERA, None, None, recursiveSearch,</li> <li>53. PARAMETER, 1000, None, 1000,</li> <li>54. SUM, 10, 3004, 2000,</li> <li>55. PARAMETER, 2000, None, 1001,</li> <li>56. PARAMETER, 1002, None, 1002,</li> </ol>

```

int i, value, result;
char option;

from i=0 to 9 {
  read(arr[i]);
}
print("Iterative Binary Search, Recursive
Binary Search or Linear Search? ('i', 'r', or
'l'): ");
read(option);
print("Value to search: ");
read(value);

switch(option) {
case 'i' {
  print(binarySearch(value));
}
case 'r' {
  print(recursiveSearch(value, 0, 9));
}
case 'l' {
  print(linearSearch(value));
}
default {
  print("What? \n");
}
}

```

```

57. GOSUB, recursiveSearch, None, 28,
58. ASSIGN, 12, None, 2000,
59. RETURN, None, None, 2000,
60. NEG, 3004, None, 2002,
61. RETURN, None, None, 2002,
62. ENDFUNC, None, None, None,
63. ASSIGN, 3000, None, 1001,
64. LTE, 1001, 3001, 2000,
65. GOTO, 2000, None, 74,
66. VERIFY, 1001, 10, None,
67. POINT, 0, 1001, 4000,
68. EQ, 4000, 1000, 2000,
69. GOTO, 2000, None, 71,
70. RETURN, None, None, 1001,
71. SUM, 1001, 3004, 2000,
72. ASSIGN, 2000, None, 1001,
73. GOTO, None, None, 64,
74. NEG, 3004, None, 2000,
75. RETURN, None, None, 2000,
76. ENDFUNC, None, None, None,
77. ASSIGN, 3000, None, 1000,
78. LTE, 1000, 3001, 2000,
79. GOTO, 2000, None, 86,
80. VERIFY, 1000, 10, None,
81. POINT, 0, 1000, 4000,
82. READ, None, None, 4000,
83. SUM, 1000, 3004, 2000,
84. ASSIGN, 2000, None, 1000,
85. GOTO, None, None, 78,
86. PRINT, None, None, "Iterative Binary Search,
Recursive Binary Search or Linear Search? ('i', 'r',
or 'l'): ",
87. READ, None, None, 1200,
88. PRINT, None, None, "Value to search: ",
89. READ, None, None, 1001,
90. EQ, 1200, 3200, 2000,
91. GOTO, 2000, None, 98,
92. ERA, None, None, binarySearch,
93. PARAMETER, 1001, None, 1000,
94. GOSUB, binarySearch, None, 2,
95. ASSIGN, 11, None, 2000,
96. PRINT, None, None, 2000,
97. GOTO, None, None, 117,
98. EQ, 1200, 3201, 2001,
99. GOTO, 2001, None, 108,
100. ERA, None, None, recursiveSearch,
101. PARAMETER, 1001, None, 1000,
102. PARAMETER, 3000, None, 1001,
103. PARAMETER, 3001, None, 1002,
104. GOSUB, recursiveSearch, None, 28,
105. ASSIGN, 12, None, 2001,
106. PRINT, None, None, 2001,
107. GOTO, None, None, 117,
108. EQ, 1200, 3202, 2002,
109. GOTO, 2002, None, 116,
110. ERA, None, None, linearSearch,
111. PARAMETER, 1001, None, 1000,
112. GOSUB, linearSearch, None, 63,
113. ASSIGN, 13, None, 2002,
114. PRINT, None, None, 2002,
115. GOTO, None, None, 117,
116. PRINT, None, None, "What? \n"

```

	<p>Entrada:</p> <pre> 1 3 5 9 11 14 16 19 21 23 Iterative Binary Search, Recursive Binary Search or Linear Search? ('i', 'r', or 'l'): i Value to search: 21 </pre> <p>Resultado:</p> <pre> 8 </pre>
<pre> program sort;  vars     int arr[10];  func void swap(int i, int j) {     vars         int temp;      temp = arr[i];     arr[i] = arr[j];     arr[j] = temp; }  func void bubbleSort(int lim) {     vars         int temp, i, j;      from i=0 to lim-1 {         from j=0 to lim - i - 2 {             if (arr[j] &gt; arr[j+1]) {                 swap(j, j+1);             }         }     } }  func int partition(int low, int high) {     vars         int pivot = arr[high];         int i = low - 1, j;      from j=low to high-1 {         if (arr[j] &lt; pivot) {             i = i + 1;             swap(i, j);         }     }     swap(i+1, high);     return(i + 1); }  func void quickSort(int low, int high) {     vars         int pivot;         if (low &lt; high) { </pre>	<p>Código Intermedio:</p> <pre> 0. GOTO, None, None, 1, 1. GOMAIN, None, None, 91, 2. VERIFY, 1000, 10, None, 3. POINT, 0, 1000, 4000, 4. ASSIGN, 4000, None, 1002, 5. VERIFY, 1000, 10, None, 6. POINT, 0, 1000, 4001, 7. VERIFY, 1001, 10, None, 8. POINT, 0, 1001, 4002, 9. ASSIGN, 4002, None, 4001, 10. VERIFY, 1001, 10, None, 11. POINT, 0, 1001, 4003, 12. ASSIGN, 1002, None, 4003, 13. ENDFUNC, None, None, None, 14. ASSIGN, 3001, None, 1002, 15. SUB, 1000, 3002, 2000, 16. LTE, 1002, 2000, 2001, 17. GOTOF, 2001, None, 41, 18. ASSIGN, 3001, None, 1003, 19. SUB, 1000, 1002, 2000, 20. SUB, 2000, 3003, 2001, 21. LTE, 1003, 2001, 2000, 22. GOTOF, 2000, None, 38, 23. VERIFY, 1003, 10, None, 24. POINT, 0, 1003, 4000, 25. SUM, 1003, 3002, 2001, 26. VERIFY, 2001, 10, None, 27. POINT, 0, 2001, 4001, 28. GT, 4000, 4001, 2000, 29. GOTOF, 2000, None, 35, 30. ERA, None, None, swap, 31. PARAMETER, 1003, None, 1000, 32. SUM, 1003, 3002, 2000, 33. PARAMETER, 2000, None, 1001, 34. GOSUB, swap, None, 2, 35. SUM, 1003, 3002, 2000, 36. ASSIGN, 2000, None, 1003, 37. GOTO, None, None, 19, 38. SUM, 1002, 3002, 2000, 39. ASSIGN, 2000, None, 1002, 40. GOTO, None, None, 15, 41. ENDFUNC, None, None, None, 42. VERIFY, 1001, 10, None, 43. POINT, 0, 1001, 4000, </pre>



<pre>         pivot = partition(low, high);          quickSort(low, pivot - 1);         quickSort(pivot + 1, high);     }  main() {     vars     int i;     char c;     from i = 0 to 9 {         read(arr[i]);     }      print("Bubble or quick? (b or q): ");     read(c);      switch(c) {         case 'b' {             bubbleSort(10);         }         case 'q' {             quickSort(0, 9);         }         default {             print("What?");         }     }      print("Result: ");     from i = 0 to 9 {         print(arr[i], " ");     } } </pre>	<pre> 44. ASSIGN, 4000, None, 1002, 45. SUB, 1000, 3002, 2000, 46. ASSIGN, 2000, None, 1003, 47. ASSIGN, 1000, None, 1004, 48. SUB, 1001, 3002, 2000, 49. LTE, 1004, 2000, 2001, 50. GOTO, 2001, None, 64, 51. VERIFY, 1004, 10, None, 52. POINT, 0, 1004, 4001, 53. LT, 4001, 1002, 2000, 54. GOTO, 2000, None, 61, 55. SUM, 1003, 3002, 2001, 56. ASSIGN, 2001, None, 1003, 57. ERA, None, None, swap, 58. PARAMETER, 1003, None, 1000, 59. PARAMETER, 1004, None, 1001, 60. GOSUB, swap, None, 2, 61. SUM, 1004, 3002, 2000, 62. ASSIGN, 2000, None, 1004, 63. GOTO, None, None, 48, 64. ERA, None, None, swap, 65. SUM, 1003, 3002, 2001, 66. PARAMETER, 2001, None, 1000, 67. PARAMETER, 1001, None, 1001, 68. GOSUB, swap, None, 2, 69. SUM, 1003, 3002, 2000, 70. RETURN, None, None, 2000, 71. ENDFUNC, None, None, None, 72. LT, 1000, 1001, 2000, 73. GOTO, 2000, None, 90, 74. ERA, None, None, partition, 75. PARAMETER, 1000, None, 1000, 76. PARAMETER, 1001, None, 1001, 77. GOSUB, partition, None, 42, 78. ASSIGN, 10, None, 2000, 79. ASSIGN, 2000, None, 1002, 80. ERA, None, None, quickSort, 81. PARAMETER, 1000, None, 1000, 82. SUB, 1002, 3002, 2000, 83. PARAMETER, 2000, None, 1001, 84. GOSUB, quickSort, None, 72, 85. ERA, None, None, quickSort, 86. SUM, 1002, 3002, 2000, 87. PARAMETER, 2000, None, 1000, 88. PARAMETER, 1001, None, 1001, 89. GOSUB, quickSort, None, 72, 90. ENDFUNC, None, None, None, 91. ASSIGN, 3001, None, 1000, 92. LTE, 1000, 3004, 2000, 93. GOTO, 2000, None, 100, 94. VERIFY, 1000, 10, None, 95. POINT, 0, 1000, 4000, 96. READ, None, None, 4000, 97. SUM, 1000, 3002, 2000, 98. ASSIGN, 2000, None, 1000, 99. GOTO, None, None, 92, 100. PRINT, None, None, "Bubble or quick? (b or q): ", 101. READ, None, None, 1200, 102. EQ, 1200, 3200, 2000, 103. GOTO, 2000, None, 108, 104. ERA, None, None, bubbleSort, 105. PARAMETER, 3005, None, 1000, 106. GOSUB, bubbleSort, None, 14, 107. GOTO, None, None, 116, 108. EQ, 1200, 3201, 2000, 109. GOTO, 2000, None, 115, </pre>
--	--

	<pre> 110. ERA, None, None, quickSort, 111. PARAMETER, 3001, None, 1000, 112. PARAMETER, 3004, None, 1001, 113. GOSUB, quickSort, None, 72, 114. GOTO, None, None, 116, 115. PRINT, None, None, "What?", 116. PRINT, None, None, "Result: ", 117. ASSIGN, 3001, None, 1000, 118. LTE, 1000, 3004, 2000, 119. GOTOF, 2000, None, 127, 120. VERIFY, 1000, 10, None, 121. POINT, 0, 1000, 4001, 122. PRINT, None, None, 4001, 123. PRINT, None, None, " ", 124. SUM, 1000, 3002, 2000, 125. ASSIGN, 2000, None, 1000, 126. GOTO, None, None, 118] </pre> <p>Entrada:</p> <pre> 10 9 8 4 5 2 6 1 3 8 Bubble or quick? (b or q): q </pre> <p>Resultado:</p> <p>Result: 1 2 3 4 5 6 8 8 9 10</p>
<pre> program factorial;  vars     int arr[91];  func int cyclicFactorial(int n) {     vars         int result = 1, i;      from i=1 to n {         result = result * i;     }      return(result); }  func int recursiveFactorial(int n) {     if (n == 0 or n == 1) {         return(1);     }     return(n * recursiveFactorial(n - 1)); }  func int dynamicFactorial(int n) {     if (n == 0 or n == 1) {         return(1);     }      if (arr[n] != -1) {         return(arr[n]);     } </pre>	<p>Código Intermedio:</p> <pre> 0. GOTO, None, None, 1, 1. GOMAIN, None, None, 52, 2. ASSIGN, 3000, None, 1001, 3. ASSIGN, 3000, None, 1002, 4. LTE, 1002, 1000, 2000, 5. GOTOF, 2000, None, 11, 6. MULT, 1001, 1002, 2000, 7. ASSIGN, 2000, None, 1001, 8. SUM, 1002, 3000, 2000, 9. ASSIGN, 2000, None, 1002, 10. GOTO, None, None, 4, 11. RETURN, None, None, 1001, 12. ENDFUNC, None, None, None, 13. EQ, 1000, 3001, 2000, 14. EQ, 1000, 3000, 2001, 15. OR, 2000, 2001, 2002, 16. GOTOF, 2002, None, 18, 17. RETURN, None, None, 3000, 18. ERA, None, None, recursiveFactorial, 19. SUB, 1000, 3000, 2000, 20. PARAMETER, 2000, None, 1000, 21. GOSUB, recursiveFactorial, None, 13, 22. ASSIGN, 92, None, 2001, 23. MULT, 1000, 2001, 2002, 24. RETURN, None, None, 2002, 25. ENDFUNC, None, None, None, 26. EQ, 1000, 3001, 2000, 27. EQ, 1000, 3000, 2001, 28. OR, 2000, 2001, 2002, </pre>

```

        arr[n] = dynamicFactorial(n-1) * n;
        return(arr[n]);
    }

    main() {
        vars
        char option;
        int value, i;

        from i=0 to 90 {
            arr[i] = -1;
        }

        print("Cyclic, Recursive, or Dynamic?:
('c', 'r' or 'd'): ");
        read(option);
        print("Value: ");
        read(value);

        switch(option) {
            case 'c' {
                print(cyclicFactorial(value));
            }
            case 'r' {
                print(recursiveFactorial(value));
            }
            case 'd' {
                print(dynamicFactorial(value));
            }
            default {
                print("What?");
            }
        }
    }
}

```

```

29. GOTO, 2002, None, 31,
30. RETURN, None, None, 3000,
31. VERIFY, 1000, 91, None,
32. POINT, 0, 1000, 4000,
33. NEG, 3000, None, 2000,
34. DIFF, 4000, 2000, 2001,
35. GOTO, 2001, None, 39,
36. VERIFY, 1000, 91, None,
37. POINT, 0, 1000, 4001,
38. RETURN, None, None, 4001,
39. VERIFY, 1000, 91, None,
40. POINT, 0, 1000, 4002,
41. ERA, None, None, dynamicFactorial,
42. SUB, 1000, 3000, 2002,
43. PARAMETER, 2002, None, 1000,
44. GOSUB, dynamicFactorial, None, 26,
45. ASSIGN, 93, None, 2000,
46. MULT, 2000, 1000, 2001,
47. ASSIGN, 2001, None, 4002,
48. VERIFY, 1000, 91, None,
49. POINT, 0, 1000, 4003,
50. RETURN, None, None, 4003,
51. ENDFUNC, None, None, None,
52. ASSIGN, 3001, None, 1001,
53. LTE, 1001, 3003, 2000,
54. GOTO, 2000, None, 62,
55. VERIFY, 1001, 91, None,
56. POINT, 0, 1001, 4000,
57. NEG, 3000, None, 2000,
58. ASSIGN, 2000, None, 4000,
59. SUM, 1001, 3000, 2000,
60. ASSIGN, 2000, None, 1001,
61. GOTO, None, None, 53,
62. PRINT, None, None, "Cyclic, Recursive, or
Dynamic?: ('c', 'r' or 'd'): ",
63. READ, None, None, 1200,
64. PRINT, None, None, "Value: ",
65. READ, None, None, 1000,
66. EQ, 1200, 3200, 2000,
67. GOTO, 2000, None, 74,
68. ERA, None, None, cyclicFactorial,
69. PARAMETER, 1000, None, 1000,
70. GOSUB, cyclicFactorial, None, 2,
71. ASSIGN, 91, None, 2000,
72. PRINT, None, None, 2000,
73. GOTO, None, None, 91,
74. EQ, 1200, 3201, 2001,
75. GOTO, 2001, None, 82,
76. ERA, None, None, recursiveFactorial,
77. PARAMETER, 1000, None, 1000,
78. GOSUB, recursiveFactorial, None, 13,
79. ASSIGN, 92, None, 2001,
80. PRINT, None, None, 2001,
81. GOTO, None, None, 91,
82. EQ, 1200, 3202, 2002,
83. GOTO, 2002, None, 90,
84. ERA, None, None, dynamicFactorial,
85. PARAMETER, 1000, None, 1000,
86. GOSUB, dynamicFactorial, None, 26,
87. ASSIGN, 93, None, 2002,
88. PRINT, None, None, 2002,
89. GOTO, None, None, 91,
90. PRINT, None, None, "What?"

```

### Entrada:

Cyclic, Recursive, or Dynamic?: ('c', 'r' or 'd'): r

	Value: 10  <b>Resultado:</b> 3628800
<pre> program fibonacci;  vars     int arr[50];  func int cyclicFibonacci(int n) {     vars         int prev = 0, curr = 1, next, i;      if (n == 0 or n == 1) {         return(n);     }      from i=2 to n {         next = curr + prev;         prev = curr;         curr = next;     }      return(next); }  func int recursiveFibonacci(int n) {     if (n == 0 or n == 1) {         return(n);     }      return(recursiveFibonacci(n - 2) + recursiveFibonacci(n - 1)); }  func int dynamicFibonacci(int n) {     if (n == 0 or n == 1) {         return(n);     }      if (arr[n] != -1) {         return(arr[n]);     }      arr[n] = dynamicFibonacci(n-2) + dynamicFibonacci(n-1);     return(arr[n]); }  main() {     vars         char option;         int value, i;      from i=0 to 49 {         arr[i] = -1;     }      print("Cyclic, Recursive, or Dynamic?: ('c', 'r' or 'd'): ");     read(option);     print("Value: ");     read(value);      switch(option) { </pre>	<b>Generación de Código:</b> <ol style="list-style-type: none"> <li>0. GOTO, None, None, 1,</li> <li>1. GOMAIN, None, None, 70,</li> <li>2. ASSIGN, 3000, None, 1001,</li> <li>3. ASSIGN, 3001, None, 1002,</li> <li>4. EQ, 1000, 3000, 2000,</li> <li>5. EQ, 1000, 3001, 2001,</li> <li>6. OR, 2000, 2001, 2002,</li> <li>7. GOTO, 2002, None, 9,</li> <li>8. RETURN, None, None, 1000,</li> <li>9. ASSIGN, 3002, None, 1004,</li> <li>10. LTE, 1004, 1000, 2000,</li> <li>11. GOTO, 2000, None, 19,</li> <li>12. SUM, 1002, 1001, 2001,</li> <li>13. ASSIGN, 2001, None, 1003,</li> <li>14. ASSIGN, 1002, None, 1001,</li> <li>15. ASSIGN, 1003, None, 1002,</li> <li>16. SUM, 1004, 3001, 2002,</li> <li>17. ASSIGN, 2002, None, 1004,</li> <li>18. GOTO, None, None, 10,</li> <li>19. RETURN, None, None, 1003,</li> <li>20. ENDFUNC, None, None, None,</li> <li>21. EQ, 1000, 3000, 2000,</li> <li>22. EQ, 1000, 3001, 2001,</li> <li>23. OR, 2000, 2001, 2002,</li> <li>24. GOTO, 2002, None, 26,</li> <li>25. RETURN, None, None, 1000,</li> <li>26. ERA, None, None, recursiveFibonacci,</li> <li>27. SUB, 1000, 3002, 2000,</li> <li>28. PARAMETER, 2000, None, 1000,</li> <li>29. GOSUB, recursiveFibonacci, None, 21,</li> <li>30. ASSIGN, 51, None, 2001,</li> <li>31. ERA, None, None, recursiveFibonacci,</li> <li>32. SUB, 1000, 3001, 2002,</li> <li>33. PARAMETER, 2002, None, 1000,</li> <li>34. GOSUB, recursiveFibonacci, None, 21,</li> <li>35. ASSIGN, 51, None, 2000,</li> <li>36. SUM, 2001, 2000, 2002,</li> <li>37. RETURN, None, None, 2002,</li> <li>38. ENDFUNC, None, None, None,</li> <li>39. EQ, 1000, 3000, 2000,</li> <li>40. EQ, 1000, 3001, 2001,</li> <li>41. OR, 2000, 2001, 2002,</li> <li>42. GOTO, 2002, None, 44,</li> <li>43. RETURN, None, None, 1000,</li> <li>44. VERIFY, 1000, 50, None,</li> <li>45. POINT, 0, 1000, 4000,</li> <li>46. NEG, 3001, None, 2000,</li> <li>47. DIFF, 4000, 2000, 2001,</li> <li>48. GOTO, 2001, None, 52,</li> <li>49. VERIFY, 1000, 50, None,</li> <li>50. POINT, 0, 1000, 4001,</li> <li>51. RETURN, None, None, 4001,</li> <li>52. VERIFY, 1000, 50, None,</li> <li>53. POINT, 0, 1000, 4002,</li> <li>54. ERA, None, None, dynamicFibonacci,</li> <li>55. SUB, 1000, 3002, 2002,</li> <li>56. PARAMETER, 2002, None, 1000,</li> <li>57. GOSUB, dynamicFibonacci, None, 39,</li> <li>58. ASSIGN, 52, None, 2000,</li> </ol>

<pre> case 'c' { print(cyclicFibonacci(value)); } case 'r' { print(recursiveFibonacci(value)); } case 'd' { print(dynamicFibonacci(value)); } default { print("What?"); } } </pre>	<pre> 59. ERA, None, None, dynamicFibonacci, 60. SUB, 1000, 3001, 2001, 61. PARAMETER, 2001, None, 1000, 62. GOSUB, dynamicFibonacci, None, 39, 63. ASSIGN, 52, None, 2002, 64. SUM, 2000, 2002, 2001, 65. ASSIGN, 2001, None, 4002, 66. VERIFY, 1000, 50, None, 67. POINT, 0, 1000, 4003, 68. RETURN, None, None, 4003, 69. ENDFUNC, None, None, None, 70. ASSIGN, 3000, None, 1001, 71. LTE, 1001, 3004, 2000, 72. GOTOF, 2000, None, 80, 73. VERIFY, 1001, 50, None, 74. POINT, 0, 1001, 4000, 75. NEG, 3001, None, 2000, 76. ASSIGN, 2000, None, 4000, 77. SUM, 1001, 3001, 2000, 78. ASSIGN, 2000, None, 1001, 79. GOTO, None, None, 71, 80. PRINT, None, None, "Cyclic, Recursive, or Dynamic?: ('c', 'r' or 'd'):", 81. READ, None, None, 1200, 82. PRINT, None, None, "Value: ", 83. READ, None, None, 1000, 84. EQ, 1200, 3200, 2000, 85. GOTOF, 2000, None, 92, 86. ERA, None, None, cyclicFibonacci, 87. PARAMETER, 1000, None, 1000, 88. GOSUB, cyclicFibonacci, None, 2, 89. ASSIGN, 50, None, 2000, 90. PRINT, None, None, 2000, 91. GOTO, None, None, 109, 92. EQ, 1200, 3201, 2001, 93. GOTOF, 2001, None, 100, 94. ERA, None, None, recursiveFibonacci, 95. PARAMETER, 1000, None, 1000, 96. GOSUB, recursiveFibonacci, None, 21, 97. ASSIGN, 51, None, 2001, 98. PRINT, None, None, 2001, 99. GOTO, None, None, 109, 100. EQ, 1200, 3202, 2002, 101. GOTOF, 2002, None, 108, 102. ERA, None, None, dynamicFibonacci, 103. PARAMETER, 1000, None, 1000, 104. GOSUB, dynamicFibonacci, None, 39, 105. ASSIGN, 52, None, 2002, 106. PRINT, None, None, 2002, 107. GOTO, None, None, 109, 108. PRINT, None, None, "What?" </pre> <p><b>Entrada:</b>  Cyclic, Recursive, or Dynamic?: ('c', 'r' or 'd'): d  Value: 25</p> <p><b>Resultado:</b>  75025</p>
<pre> program matrix;  vars     float m1[3,3], m2[3,3], m3[3,3]; </pre>	<p><b>Código Intermedio:</b></p> <pre> 0. GOTO, None, None, 1, 1. GOMAIN, None, None, 44, 2. ASSIGN, 3000, None, 1000, </pre>

```

func void multiply() {
    vars
    int i, j, k;

    from i=0 to 2 {
        from j=0 to 2 {
            from k=0 to 2 {
                m3[i,j] = m1[i, k] * m2[k, j] +
m3[i, j];
            }
        }
    }
}

main() {
    vars
    int i, j;

    from i=0 to 2 {
        from j=0 to 2 {
            read(m1[i, j]);
        }
    }

    from i=0 to 2 {
        from j=0 to 2 {
            print(m1[i,j], "\t");
        }
        print("\n");
    }

    from i=0 to 2 {
        from j=0 to 2 {
            read(m2[i, j]);
        }
    }

    from i=0 to 2 {
        from j=0 to 2 {
            print(m2[i,j], "\t");
        }
        print("\n");
    }

    from i=0 to 2 {
        from j=0 to 2 {
            m3[i, j] = 0.0;
        }
    }

    multiply();

    from i=0 to 2 {
        from j=0 to 2 {
            print(m3[i,j], "\t");
        }
        print("\n");
    }
}

```

```

3. LTE, 1000, 3001, 2000,
4. GOTO, 2000, None, 43,
5. ASSIGN, 3000, None, 1001,
6. LTE, 1001, 3001, 2000,
7. GOTO, 2000, None, 40,
8. ASSIGN, 3000, None, 1002,
9. LTE, 1002, 3001, 2000,
10. GOTO, 2000, None, 37,
11. VERIFY, 1000, 3, None,
12. VERIFY, 1001, 3, None,
13. MULT, 1000, 3002, 2000,
14. SUM, 2000, 1001, 2001,
15. POINT, 118, 2001, 4000,
16. VERIFY, 1000, 3, None,
17. VERIFY, 1002, 3, None,
18. MULT, 1000, 3002, 2002,
19. SUM, 2002, 1002, 2003,
20. POINT, 100, 2003, 4001,
21. VERIFY, 1002, 3, None,
22. VERIFY, 1001, 3, None,
23. MULT, 1002, 3002, 2004,
24. SUM, 2004, 1001, 2005,
25. POINT, 109, 2005, 4002,
26. MULT, 4001, 4002, 2100,
27. VERIFY, 1000, 3, None,
28. VERIFY, 1001, 3, None,
29. MULT, 1000, 3002, 2006,
30. SUM, 2006, 1001, 2007,
31. POINT, 118, 2007, 4003,
32. SUM, 2100, 4003, 2101,
33. ASSIGN, 2101, None, 4000,
34. SUM, 1002, 3003, 2008,
35. ASSIGN, 2008, None, 1002,
36. GOTO, None, None, 9,
37. SUM, 1001, 3003, 2008,
38. ASSIGN, 2008, None, 1001,
39. GOTO, None, None, 6,
40. SUM, 1000, 3003, 2008,
41. ASSIGN, 2008, None, 1000,
42. GOTO, None, None, 3,
43. ENDFUNC, None, None, None,
44. ASSIGN, 3000, None, 1000,
45. LTE, 1000, 3001, 2000,
46. GOTO, 2000, None, 62,
47. ASSIGN, 3000, None, 1001,
48. LTE, 1001, 3001, 2000,
49. GOTO, 2000, None, 59,
50. VERIFY, 1000, 3, None,
51. VERIFY, 1001, 3, None,
52. MULT, 1000, 3002, 2000,
53. SUM, 2000, 1001, 2001,
54. POINT, 100, 2001, 4000,
55. READ, None, None, 4000,
56. SUM, 1001, 3003, 2002,
57. ASSIGN, 2002, None, 1001,
58. GOTO, None, None, 48,
59. SUM, 1000, 3003, 2002,
60. ASSIGN, 2002, None, 1000,
61. GOTO, None, None, 45,
62. ASSIGN, 3000, None, 1000,
63. LTE, 1000, 3001, 2002,
64. GOTO, 2002, None, 82,
65. ASSIGN, 3000, None, 1001,
66. LTE, 1001, 3001, 2002,
67. GOTO, 2002, None, 78,
68. VERIFY, 1000, 3, None,
69. VERIFY, 1001, 3, None,

```

	70. MULT, 1000, 3002, 2002, 71. SUM, 2002, 1001, 2003, 72. POINT, 100, 2003, 4001, 73. PRINT, None, None, 4001, 74. PRINT, None, None, "\t", 75. SUM, 1001, 3003, 2004, 76. ASSIGN, 2004, None, 1001, 77. GOTO, None, None, 66, 78. PRINT, None, None, "\n", 79. SUM, 1000, 3003, 2004, 80. ASSIGN, 2004, None, 1000, 81. GOTO, None, None, 63, 82. ASSIGN, 3000, None, 1000, 83. LTE, 1000, 3001, 2004, 84. GOTOF, 2004, None, 100, 85. ASSIGN, 3000, None, 1001, 86. LTE, 1001, 3001, 2004, 87. GOTOF, 2004, None, 97, 88. VERIFY, 1000, 3, None, 89. VERIFY, 1001, 3, None, 90. MULT, 1000, 3002, 2004, 91. SUM, 2004, 1001, 2005, 92. POINT, 109, 2005, 4002, 93. READ, None, None, 4002, 94. SUM, 1001, 3003, 2006, 95. ASSIGN, 2006, None, 1001, 96. GOTO, None, None, 86, 97. SUM, 1000, 3003, 2006, 98. ASSIGN, 2006, None, 1000, 99. GOTO, None, None, 83, 100. ASSIGN, 3000, None, 1000, 101. LTE, 1000, 3001, 2006, 102. GOTOF, 2006, None, 120, 103. ASSIGN, 3000, None, 1001, 104. LTE, 1001, 3001, 2006, 105. GOTOF, 2006, None, 116, 106. VERIFY, 1000, 3, None, 107. VERIFY, 1001, 3, None, 108. MULT, 1000, 3002, 2006, 109. SUM, 2006, 1001, 2007, 110. POINT, 109, 2007, 4003, 111. PRINT, None, None, 4003, 112. PRINT, None, None, "\t", 113. SUM, 1001, 3003, 2008, 114. ASSIGN, 2008, None, 1001, 115. GOTO, None, None, 104, 116. PRINT, None, None, "\n", 117. SUM, 1000, 3003, 2008, 118. ASSIGN, 2008, None, 1000, 119. GOTO, None, None, 101, 120. ASSIGN, 3000, None, 1000, 121. LTE, 1000, 3001, 2008, 122. GOTOF, 2008, None, 138, 123. ASSIGN, 3000, None, 1001, 124. LTE, 1001, 3001, 2008, 125. GOTOF, 2008, None, 135, 126. VERIFY, 1000, 3, None, 127. VERIFY, 1001, 3, None, 128. MULT, 1000, 3002, 2008, 129. SUM, 2008, 1001, 2009, 130. POINT, 118, 2009, 4004, 131. ASSIGN, 3100, None, 4004, 132. SUM, 1001, 3003, 2010, 133. ASSIGN, 2010, None, 1001, 134. GOTO, None, None, 124, 135. SUM, 1000, 3003, 2010, 136. ASSIGN, 2010, None, 1000,
--	--

	<pre> 137. GOTO, None, None, 121, 138. ERA, None, None, multiply, 139. GOSUB, multiply, None, 2, 140. ASSIGN, 3000, None, 1000, 141. LTE, 1000, 3001, 2010, 142. GOTOF, 2010, None, 160, 143. ASSIGN, 3000, None, 1001, 144. LTE, 1001, 3001, 2010, 145. GOTOF, 2010, None, 156, 146. VERIFY, 1000, 3, None, 147. VERIFY, 1001, 3, None, 148. MULT, 1000, 3002, 2010, 149. SUM, 2010, 1001, 2011, 150. POINT, 118, 2011, 4005, 151. PRINT, None, None, 4005, 152. PRINT, None, None, "\t", 153. SUM, 1001, 3003, 2012, 154. ASSIGN, 2012, None, 1001, 155. GOTO, None, None, 144, 156. PRINT, None, None, "\n", 157. SUM, 1000, 3003, 2012, 158. ASSIGN, 2012, None, 1000, 159. GOTO, None, None, 141 </pre> <p>Entrada:</p> <pre> 1 2 3 4 5 6 7 8 9 1.0      2.0      3.0 4.0      5.0      6.0 7.0      8.0      9.0 9 8 7 6 5 4 3 2 1 9.0      8.0      7.0 6.0      5.0      4.0 3.0      2.0      1.0 </pre> <p>Resultado:</p> <pre> 30.0     24.0     18.0 84.0     69.0     54.0 138.0    114.0    90.0 </pre>
<pre> program oop;  class Person {     attributes         public int age;         private float weight, height;      methods         public func int getAge() { </pre>	<p>Código Intermedio:</p> <pre> 0. GOTO, None, None, 65, 1. RETURN, None, None, 7000, 2. ENDFUNC, None, None, None, 3. RETURN, None, None, 7100, 4. ENDFUNC, None, None, None, 5. ASSIGN, 1100, None, 7100, 6. ENDFUNC, None, None, None, </pre>



<pre> return(age); }  public func float getWeight() { return(weight); }  public func void setWeight(float w) { weight = w; }  public func float getHeight() { return(height); }  public func void setHeight(float h) { height = h; }  private func float calculateBMI() { vars     float BMI;  BMI = weight / (height * height); return(BMI); }  public func float getBMI() { return(calculateBMI()); } };  class Student inherits Person { attributes private float grades[3]; methods public func void setGrades() { vars     int i; print("Ingresa calificaciones: "); from i=0 to 2 {     read(grades[i]); } }  private func float calculateAverage(){ vars     int i=0;     float sum = 0.0;      while (i &lt; 3) {         sum = sum + grades[i];         i = i+1;     }     return(sum / 3); }  public func char getGrade() { vars     float avg = calculateAverage();  if (avg &gt;= 90) {     return('A'); } elif (avg &gt;= 80) {     return('B'); } elif (avg &gt;= 70) {     return('C'); } } </pre>	<pre> 7. RETURN, None, None, 7101, 8. ENDFUNC, None, None, None, 9. ASSIGN, 1100, None, 7101, 10. ENDFUNC, None, None, None, 11. MULT, 7101, 7101, 2100, 12. DIV, 7100, 2100, 2101, 13. ASSIGN, 2101, None, 1100, 14. RETURN, None, None, 1100, 15. ENDFUNC, None, None, None, 16. METHOD, None, None, -1, 17. ERA, Person, None, calculateBMI, 18. GOSUB, calculateBMI, None, 11, 19. ASSIGN, 102, None, 2100, 20. RETURN, None, None, 2100, 21. ENDFUNC, None, None, None, 22. PRINT, None, None, "Ingresa calificaciones: ", 23. ASSIGN, 3000, None, 1000, 24. LTE, 1000, 3001, 2000, 25. GOTO, 2000, None, 32, 26. VERIFY, 1000, 3, None, 27. POINT, 7102, 1000, 4000, 28. READ, None, None, 4000, 29. SUM, 1000, 3003, 2000, 30. ASSIGN, 2000, None, 1000, 31. GOTO, None, None, 24, 32. ENDFUNC, None, None, None, 33. ASSIGN, 3000, None, 1000, 34. ASSIGN, 3100, None, 1100, 35. LT, 1000, 3004, 2000, 36. GOTO, 2000, None, 44, 37. VERIFY, 1000, 3, None, 38. POINT, 7102, 1000, 4000, 39. SUM, 1100, 4000, 2100, 40. ASSIGN, 2100, None, 1100, 41. SUM, 1000, 3003, 2000, 42. ASSIGN, 2000, None, 1000, 43. GOTO, None, None, 35, 44. DIV, 1100, 3004, 2100, 45. RETURN, None, None, 2100, 46. ENDFUNC, None, None, None, 47. METHOD, None, None, -1, 48. ERA, Student, None, calculateAverage, 49. GOSUB, calculateAverage, None, 33, 50. ASSIGN, 104, None, 2100, 51. ASSIGN, 2100, None, 1100, 52. GTE, 1100, 3005, 2000, 53. GOTO, 2000, None, 56, 54. RETURN, None, None, 3200, 55. GOTO, None, None, 63, 56. GTE, 1100, 3006, 2000, 57. GOTO, 2000, None, 60, 58. RETURN, None, None, 3201, 59. GOTO, None, None, 63, 60. GTE, 1100, 3007, 2000, 61. GOTO, 2000, None, 63, 62. RETURN, None, None, 3202, 63. RETURN, None, None, 3203, 64. ENDFUNC, None, None, None, 65. INST, 1, True, Person, 66. INST, 1, True, Student, 67. GOMAIN, None, None, 68, 68. PRINT, None, None, "Ingresa edad: ", 69. READ, None, None, 1000, 70. ASSIGN, 1000, None, 60007000, 71. PRINT, None, None, "Ingresa peso: ", 72. READ, None, None, 1100, 73. METHOD, None, None, 6000, </pre>
---	--

```

    }
    return('D');
}
};

vars
    Person enrique;
    Student beto;

main() {
    vars
        int age;
        float weight, height;

    print("Ingresa edad: ");
    read(age);
    enrique.age = age;

    print("Ingresa peso: ");
    read(weight);
    enrique.setWeight(weight);

    print("Ingresa altura: ");
    read(height);
    enrique.setHeight(height);

    print("BMI is ", enrique.getBMI(), " at
age ", enrique.age, "\n");

    print("Ingresa edad: ");
    read(age);
    beto.age = age;

    print("Ingresa peso: ");
    read(weight);
    beto.setWeight(weight);

    print("Ingresa altura: ");
    read(height);
    beto.setHeight(height);

    beto.setGrades();

    print("BMI is ", beto.getBMI(), " at age
", beto.age, "\n");
    print("Grade letter is: ",
beto.getGrade(), "\n");
}

74. ERA, Person, None, setWeight,
75. PARAMETER, 1100, None, 1100,
76. GOSUB, setWeight, None, 5,
77. PRINT, None, None, "Ingresa altura: ",
78. READ, None, None, 1101,
79. METHOD, None, None, 6000,
80. ERA, Person, None, setHeight,
81. PARAMETER, 1101, None, 1100,
82. GOSUB, setHeight, None, 9,
83. PRINT, None, None, "BMI is ",
84. METHOD, None, None, 6000,
85. ERA, Person, None, getBMI,
86. GOSUB, getBMI, None, 16,
87. ASSIGN, 103, None, 2100,
88. PRINT, None, None, 2100,
89. PRINT, None, None, " at age ",
90. PRINT, None, None, 60007000,
91. PRINT, None, None, "\n",
92. PRINT, None, None, "Ingresa edad: ",
93. READ, None, None, 1000,
94. ASSIGN, 1000, None, 60017000,
95. PRINT, None, None, "Ingresa peso: ",
96. READ, None, None, 1100,
97. METHOD, None, None, 6001,
98. ERA, Person, None, setWeight,
99. PARAMETER, 1100, None, 1100,
100. GOSUB, setWeight, None, 5,
101. PRINT, None, None, "Ingresa altura: ",
102. READ, None, None, 1101,
103. METHOD, None, None, 6001,
104. ERA, Person, None, setHeight,
105. PARAMETER, 1101, None, 1100,
106. GOSUB, setHeight, None, 9,
107. METHOD, None, None, 6001,
108. ERA, Student, None, setGrades,
109. GOSUB, setGrades, None, 22,
110. PRINT, None, None, "BMI is ",
111. METHOD, None, None, 6001,
112. ERA, Person, None, getBMI,
113. GOSUB, getBMI, None, 16,
114. ASSIGN, 103, None, 2101,
115. PRINT, None, None, 2101,
116. PRINT, None, None, " at age ",
117. PRINT, None, None, 60017000,
118. PRINT, None, None, "\n",
119. PRINT, None, None, "Grade letter is: ",
120. METHOD, None, None, 6001,
121. ERA, Student, None, getGrade,
122. GOSUB, getGrade, None, 47,
123. ASSIGN, 200, None, 2200,
124. PRINT, None, None, 2200,
125. PRINT, None, None, "\n"

```

### Entrada:

Ingresa edad: 13  
 Ingresa peso: 71  
 Ingresa altura: 1.67

Ingresa edad: 22  
 Ingresa peso: 78  
 Ingresa altura: 1.56  
 Ingresa calificaciones: 90  
 75  
 100

	<p>Resultado:</p> <p>BMI is 25.458065904119906 at age 13</p> <p>BMI is 32.05128205128205 at age 22</p> <p>Grade letter is: B</p>
--	--

# Documentación del Código del Proyecto

Esta sección explica algunos segmentos de código que se quieren resaltar y explicar con más detalle.

## DirGen

Primero se quiere resaltar la implementación que se hizo para agregar nuevas variables y funciones en sus directorios correspondientes. Dado que se está trabajando con un directorio de clases en conjunto con un directorio de funciones general, existen varias posibilidades que se tienen que considerar para saber exactamente en dónde guardar la variable o función.

```
def add_function(self, new_function, class_name = None, function_level = None):
    self.current_scope = new_function.name

    if self.function_search(new_function.name, class_name) is not None:
        print(f'[Error] Function \'{new_function.name}\'' already declared.')
        sys.exit()

    if self.in_class == 0:
        # New function is global
        self.dir_func[new_function.name] = new_function
    else:
        # New function is a method
        new_function.set_level(function_level)
        new_function.set_original_class(class_name)
        self.dir_class[class_name].methods[new_function.name] = new_function

def add_variable(self, variable_name, class_name, variable_level = None):
    if self.variable_check(variable_name, class_name) is not None:
        print(f'[Error] Variable \'{variable_name}\'' already declared.')
        sys.exit()

    if (self.current_scope == "global"):
        # Variable is global
        if self.current_type != Type.ID:
            # Get address for a primitive global variable
            address = self.global_address_manager.get_address(self.current_type, self.d1, self.d2)
        else:
            # Get address for a global instance
            address = self.global_instance_manager.get_address(self.d1, self.d2)
        self.dir_func["global"].variables[variable_name] = Variable(variable_name, self.current_type,
self.current_type_id, address, d1=self.d1, d2=self.d2)
    elif self.in_class == 0:
```

```

        # Variable is local to a function
        address = self.dir_func[self.current_scope].get_address(self.current_type, self.d1, self.d2)
        self.dir_func[self.current_scope].variables[variable_name] = Variable(variable_name,
self.current_type, self.current_type_id, address, d1=self.d1, d2=self.d2)
    elif self.in_function == 0:
        # Variable is an attribute
        address = self.dir_class[class_name].get_address(self.current_type, self.d1, self.d2)
        self.dir_class[class_name].attributes[variable_name] = Variable(variable_name, self.current_type,
self.current_type_id, address, variable_level, class_name, d1=self.d1, d2=self.d2)
    else:
        # Variable is local to a method
        address = self.dir_class[class_name].methods[self.current_scope].get_address(self.current_type,
self.d1, self.d2)
        self.dir_class[class_name].methods[self.current_scope].variables[variable_name] =
Variable(variable_name, self.current_type, self.current_type_id, address, d1=self.d1, d2=self.d2)

    return address

```

## CustomListener

Del archivo CustomListener, primero se quiere resaltar una de las partes más importantes de todo el código, que es la función que genera un cuádruplo a partir de los datos en la Pila de Operadores y Pila de Operandos. Aunque no necesariamente fue la implementación más difícil, sí fue una parte a la que se le prestó mucha atención, debido a que esta función es crucial para que el compilador funcione de manera adecuada.

```

# Generates a quadruple using the top two operands in the operand stack and
# the top operator in the operator stack
def generate_quadruple(self, top_operator):
    right_operand = self.quadruple_list.pop_operand()
    left_operand = self.quadruple_list.pop_operand()
    result_type = semantic_cube[left_operand.variable_type][right_operand.variable_type][top_operator]

    # Checks whether the operation between both operands is valid
    if result_type is None:
        print(f"[Error] Invalid operand types \'{Type(left_operand.variable_type).name}\' and
\''{Type(right_operand.variable_type).name}\' for operator
{Operator(self.quadruple_list.top_operator()).name}.")
        sys.exit()

    result_address = self.get_temp(result_type)

    # Creates a quadruple for the received operation and adds its resulting temporary variable to the
    operand stack
    self.push_quadruple(self.quadruple_list.pop_operator(), left_operand.address, right_operand.address,
result_address)
    self.quadruple_list.push_operand(result_address, result_type)

```

Segundo, se quiere incluir la implementación de la función que genera los cuádruplos para la traslación que obtiene la dirección que se está indexando. Como se puede ver en el código, se generan diferentes cuádruplos dependiendo de si la variable es de una o dos dimensiones, lo cual se tuvo que tomar en cuenta. Otro chequeo muy importante que se tuvo considerar fue que el número de dimensiones con las que se está indexando la variable sea la misma al número de dimensiones que tiene la variable. Por ejemplo, si la variable es una matriz, revisar que no se este indexando con un sólo índice.

```
# Point 24, Point 81
def exitIndexing(self, ctx):
    # Point 81
    var = self.caller_vars.top()
    d2 = self.dir_gen.const_address_manager.get_address(Type.INT, var.d2)
    dim_count = var.dim_count
    result_address = self.dir_gen.get_pointer()
    base_address = var.address

    if dim_count == 0:
        print("[Error] Cannot index a variable with no dimensions.")
        sys.exit()

    elif dim_count == 1:
        # Creates the quadruples to calculate the formula for 1-D array pointers
        # s + base_address
        if dim_count != self.global_dims:
            print(f"[Error] Mismatched dimensions for variable {var.name}.")
            sys.exit()

        s = self.quadruple_list.pop_operand().address

        self.quadruple_list.push_quadruple(Operator.POINT, base_address, s, result_address)

    else:
        if dim_count != self.global_dims:
            print(f"[Error] Mismatched dimensions for variable {var.name}.")
            sys.exit()

        # Creates the quadruples to calculate the formula for 1-D array pointers
        # s1*m1 + s2 + base_address

        s2 = self.quadruple_list.pop_operand().address
        s1 = self.quadruple_list.pop_operand().address

        temp_addresses = [self.get_temp(Type.INT), self.get_temp(Type.INT)]
        self.quadruple_list.push_quadruple(Operator.MULT, s1, d2, temp_addresses[0])
```

```

        self.quadruple_list.push_quadruple(Operator.SUM, temp_addresses[0], s2, temp_addresses[1])
        self.quadruple_list.push_quadruple(Operator.POINT, base_address, temp_addresses[1],
result_address)

    self.pointer_stack.push(result_address)
    self.global_dims = 0

    # Point 24
    self.quadruple_list.pop_operator()

```

## Virtual Machine

El segmento que se quiere resaltar de la implementación de la Máquina Virtual es la manera en que se procesan las direcciones para, ya sea leer o escribir un valor en memoria. Esto fue complicado debido a que se tienen muchos contextos distintos en el proyecto, además de la posibilidad de que puedan venir direcciones compuestas en el caso de instancias.

```

def read_address(self, address):
    value = None
    if address >= 10000:
        # Compound address
        inst = address // 10000
        attr = address % 10000
        context = inst // 1000
        reduced_index = inst % 1000
        reduced_address = attr % 1000

        if context == 6:
            # Global instance
            value = self.global_instance_memory[reduced_index].read_address(reduced_address)
        elif context == 5:
            # Local instance
            value = self.exec_stack.top().instance_list[reduced_index].read_address(reduced_address)
        elif context == 4:
            # Pointer to instance
            pointed_address = self.exec_stack.top().pointer_memory.read_pointer(reduced_index)
            pointed_address = pointed_address * 10000 + attr
            value = self.read_address(pointed_address)

    else:
        # Simple address
        context = address // 1000
        reduced_address = address % 1000
        if context == 7:
            # Attribute local to a method
            value = self.active_instances.top().read_address(reduced_address)

```

```

elif context == 4:
    # Pointer to array cell
    pointed_address = self.exec_stack.top().pointer_memory.read_pointer(reduced_address)
    value = self.read_address(pointed_address)
elif context == 3:
    # Constant
    value = self.const_table[address]
elif context == 2:
    # Temp
    value = self.exec_stack.top().temp_memory.read_address(reduced_address)
elif context == 1:
    # Local
    value = self.exec_stack.top().local_memory.read_address(reduced_address)
elif context == 0:
    # Global
    value = self.global_memory.read_address(reduced_address)

if value is None:
    print("[Error] Variable not initialized.")
    sys.exit()

return value

def write_address(self, address, value):

    if address >= 10000:
        # Compound address
        inst = address // 10000
        attr = address % 10000
        context = inst // 1000
        reduced_index = inst % 1000
        reduced_address = attr % 1000

        if context == 6:
            # Global instance
            self.global_instance_memory[reduced_index].write_address(reduced_address, value)
        elif context == 5:
            # Local instance
            self.exec_stack.top().instance_list[reduced_index].write_address(reduced_address, value)
        elif context == 4:
            # Pointer to instance
            pointed_address = self.exec_stack.top().pointer_memory.read_pointer(reduced_index)
            pointed_address = pointed_address * 10000 + attr
            self.write_address(pointed_address, value)

    else:
        context = address // 1000
        reduced_address = address % 1000

```



```

if context == 7:
    # Attribute local to a method
    self.active_instances.top().write_address(reduced_address, value)
elif context == 4:
    self.write_address(self.exec_stack.top().pointer_memory.read_pointer(reduced_address), value)
elif context == 2:
    # Temp
    self.exec_stack.top().temp_memory.write_address(reduced_address, value)
elif context == 1:
    # Local
    self.exec_stack.top().local_memory.write_address(reduced_address, value)
else:
    # Global
    self.global_memory.write_address(reduced_address, value)

```

## Machine Memory

De la Memoria de Máquina, se quiere resaltar la implementación en sí que se diseñó. Algo muy importante que se consideró fue que no hubiera desperdicio de memoria al hacer el mapeo de las direcciones virtuales a la memoria de ejecución. Y el acercamiento que se terminó implementando fue que el objeto de Memoria de Máquina recibiera como parámetro el contador de direcciones virtuales correspondiente y sobre ese hiciera una lista del tamaño necesario.

```

class FunctionMemory:
    """
    Defines the structure that will be used to create the memory required for a function call.
    This includes a local, and temporal Machine Memory, as well as a pointer memory and an instance list.
    """

    def __init__(self, local_dir, temp_dir, pointer_dir, return_addr, next_quad, is_method = False):
        self.local_memory = MachineMemory(local_dir)
        self.temp_memory = MachineMemory(temp_dir)
        self.pointer_memory = PointerMemory(pointer_dir)
        self.return_addr = return_addr
        self.next_quad = next_quad
        self.instance_list = []
        self.is_method = is_method

    def advance(self):
        self.next_quad += 1

class MachineMemory:
    """
    Defined the structure that will store all values that will be used during program execution.
    Consists mainly of a dictionary where data types are the keys and the values are lists, where

```

the size is given by the Address Manager recieved as a paramter.

"""

```
def __init__(self, address_dir):
    # Create list based on the amount of virtual directions handed out
    self.memory = {
        Type.INT: [None for i in range(address_dir.addresses[Type.INT])],
        Type.FLOAT: [None for i in range(address_dir.addresses[Type.FLOAT])],
        Type.CHAR: [None for i in range(address_dir.addresses[Type.CHAR])],
    }

def read_address(self, address):
    # Return value inside the given address
    type = address // 100
    index = address % 100
    return self.memory[type][index]

def write_address(self, address, value):
    # Write a value to the given address
    type = address // 100
    index = address % 100
    self.memory[type][index] = value
```