



Asignatura: Teoría de Autómatas  
Profesor: D. Sc. Gerardo García Gil  
2022-B

Alumno: José Rafael Ruiz Gudiño Reg. 20110374  
Ingeniería en Desarrollo de Software  
Centro de Enseñanza Técnica Industrial (CETI)

## Índice

Objetivo.....	3
Introducción .....	3
¿Qué es un compilador? .....	3
Historia .....	3
Estructura de un compilador.....	4
Análisis léxico. ....	5
Análisis sintáctico.....	6
Análisis semántico.....	7
Generación de código intermedio.....	7
Optimización de código intermedio. ....	7
Generación y optimización de código objeto. ....	8
Compilador TINY .....	8
Componentes.....	8
Funcionamiento .....	9
Lenguaje TINY .....	9
Expresiones.....	10
Limitaciones .....	10
Tokens.....	10
Máquina TM .....	10
Desarrollo .....	11
Problema para resolver .....	11
Algoritmo.....	12
Implementación .....	12
DOSBox .....	13
Cargar el compilador TINY .....	14
Código del programa .....	15

<b>Resultados.....</b>	<b>17</b>
Montaje .....	17
Compilación.....	18
Ejecución .....	19
Prueba triángulo equilátero .....	19
Prueba triángulo isósceles.....	19
Prueba triángulo escaleno .....	20
<b>Conclusiones.....</b>	<b>20</b>
<b>Referencias.....</b>	<b>21</b>

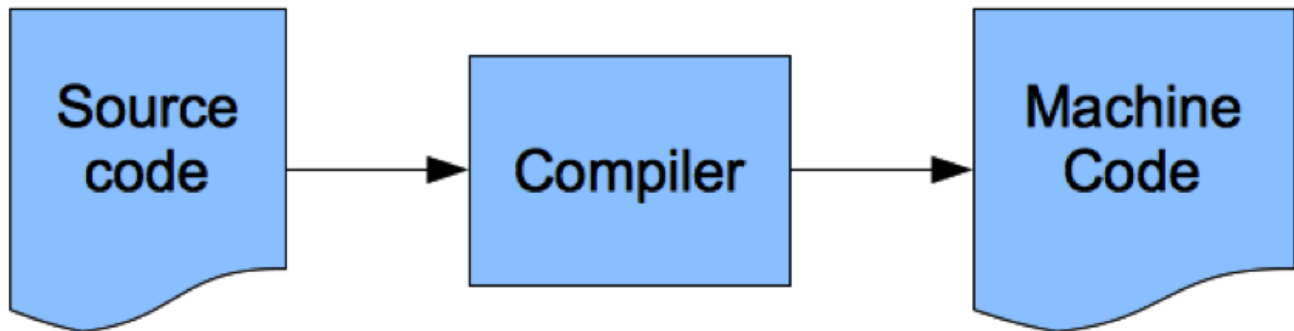
## Objetivo

Realizar el desarrollo de un programa implementado en el lenguaje y compilador TINY. El programa responderá a la problemática sugerida por el profesor, que en este caso el cual fue “Determinar si un triángulo es equilátero, isósceles o escaleno al ingresar la longitud de cada uno de sus tres lados”. A través del desarrollo de este programa, se pretende conocer de manera profunda el funcionamiento y partes de un compilador, para que de esta manera se reafirmen y apliquen todos los conocimientos que se han ido adquiriendo a lo largo del semestre. El siguiente documento presentará desde que es un compilador y que lo conforma, para seguir con la explicación del compilador, el programa a realizar, los resultados obtenidos y finalmente una conclusión de todo lo anterior.

## Introducción

### ¿Qué es un compilador?

Es un Software que traduce un programa escrito en un lenguaje de programación de alto nivel (C / C ++, COBOL, etc.) en lenguaje de máquina. Un compilador generalmente genera lenguaje ensamblador primero y luego traduce el lenguaje ensamblador al lenguaje máquina. Una utilidad conocida como «enlazador» combina todos los módulos de lenguaje de máquina necesarios en un programa ejecutable que se puede ejecutar en la computadora.



## Historia

El término «compilador» fue acuñado a principios de 1950 por Grace Murray Hopper. La traducción fue vista entonces como la «compilación» de una secuencia de rutinas seleccionadas.

Grace Brewster Murray Hopper fue una científica informática estadounidense y contraalmirante de la Marina de los Estados Unidos. Una de las primeras programadoras de la computadora Harvard Mark I, fue una pionera en programación que inventó una de las primeras herramientas relacionadas con el compilador. Ella popularizó la idea de los lenguajes de programación independientes de la máquina, lo que condujo al desarrollo de COBOL, un lenguaje de programación de alto nivel que todavía se usa en la actualidad.

El primer compilador del lenguaje de alto nivel FORTRAN se desarrolló entre 1954 y 1957 en IBM por un grupo dirigido por John Backus.



Un compilador es uno de los pilares de la programación y de cómo entender la comunicación entre un lenguaje de alto nivel y una máquina. Al poder conocer el funcionamiento de este paso intermedio nos permitirá desarrollar y programar de una forma más precisa los lenguajes de alto nivel.

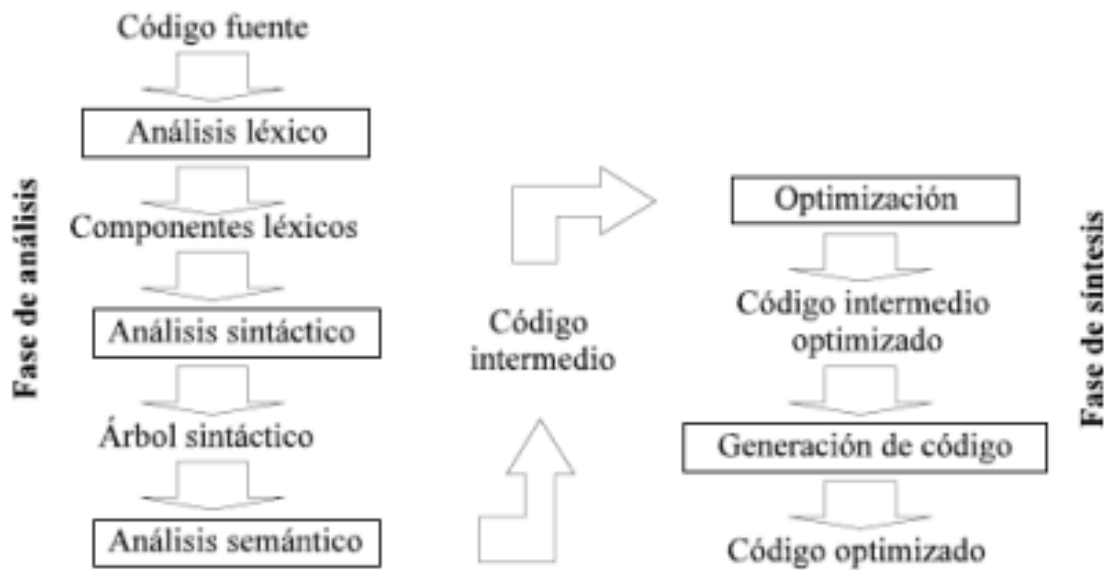
Tradicionalmente los compiladores generaban código máquina de inferior calidad que el que podían escribir programadores humanos, pero actualmente los compiladores proporcionan hoy en día un código máquina de alta calidad pequeño y rápido, haciendo poco atractiva la programación en ensamblador.

Los programadores de ensamblador siguen teniendo ventaja en cuanto a que disponen de un mayor conocimiento global del programa que les permite realizar determinadas optimizaciones del código que resultan muy difíciles para los compiladores.

### **Estructura de un compilador**

Un compilador identifica los significados de las diferentes construcciones presentes en la definición del propio lenguaje.

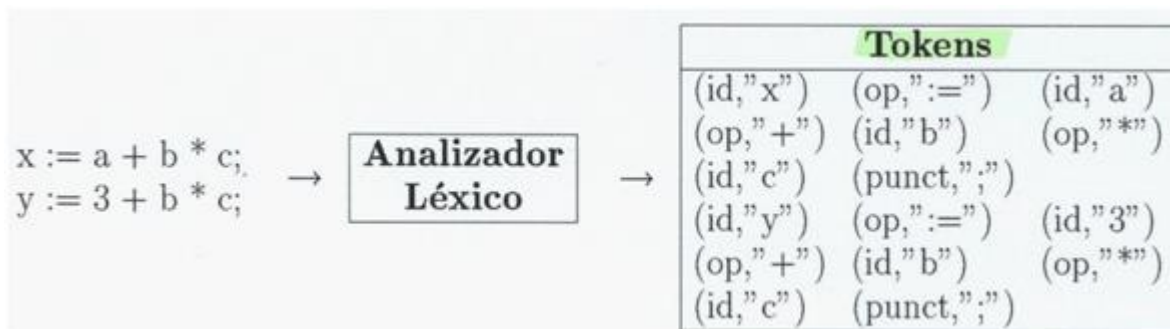
En un compilador pueden distinguirse dos fases principales: una fase de análisis, en la que la estructura y el significado del código fuente se analiza; y otra fase de síntesis, en la que se genera el programa objeto.



*Etapas de traducción de un compilador: análisis y síntesis*

## Análisis léxico.

Es la etapa en la que se realiza un análisis a nivel de caracteres. El objetivo de esta fase es reconocer los componentes léxicos presentes en el código fuente, enviándolos después, junto con sus atributos, al analizador sintáctico.



En el estudio de esta fase de análisis léxico es importante que distinguir entre:

- Token, el nombre del token es un símbolo abstracto que representa un tipo de unidad léxica. Estos tokens representan palabras reservadas, identificadores, operadores, símbolos especiales, constantes numéricas y de caracteres.
- Patrón, es una regla que genera la secuencia de caracteres que puede representar a un determinado token.
- Lexema, cadena de caracteres que concuerda con un patrón que describe un token. Un token puede tener uno o infinitos lexemas.

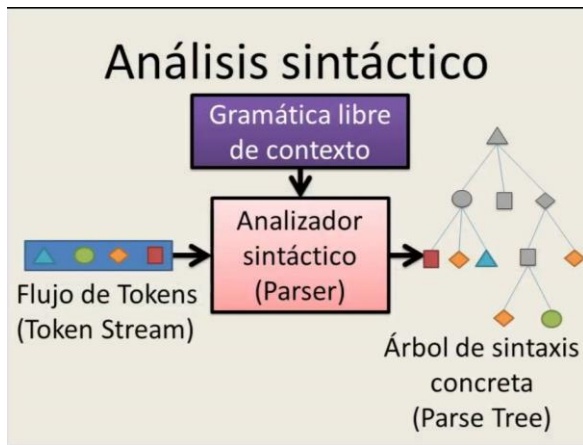
La detección de tokens llevada a cabo en esta fase de análisis léxico de un compilador se realiza con gramáticas y lenguajes regulares.

## Análisis sintáctico.

Un analizador sintáctico toma los tokens que le envíe el analizador léxico y creará un árbol sintáctico que refleje la estructura del programa fuente.

En esta fase se comprobará si con dichos tokens se puede formar alguna sentencia válida dentro del lenguaje.

La sintaxis de la mayoría de los lenguajes de programación se define habitualmente por medio de gramáticas libres de contexto. El término libre de contexto se refiere al hecho de que un no terminal puede siempre ser sustituido sin tener en cuenta el contexto en el que aparece.



A partir de estas gramáticas independientes de contexto se diseñan algoritmos de análisis sintácticos capaces de determinar si una determinada cadena de tokens pertenece al lenguaje definido por una gramática dada.

El proceso de comprobación de si una secuencia de tokens pertenece o no a un determinado lenguaje independiente de contexto se lleva a cabo por medio de autómatas a pila.

Los errores detectados en la fase de análisis sintáctico se refieren al hecho de que la estructura que se ha seguido en la construcción de una secuencia de tokens no es la correcta según la gramática que define el lenguaje.

El manejador de errores de un analizador sintáctico debe tener como objetivos:

- Indicar y localizar cada uno de los errores encontrados.
- Recuperarse del error para poder seguir examinando errores sin necesidad de cortar el proceso de compilación.
- No ralentizar en exceso el propio proceso de compilación.

Existen varias estrategias para corregirlos:

- En primer lugar, se puede ignorar el problema. Esta estrategia consiste en ignorar el resto de la entrada a analizar hasta encontrar un token especial
- Otra opción es tratar de realizar una recuperación a nivel de frase, es decir, intentar recuperar el error una vez ha sido detectado.
- Otra estrategia en el tratamiento de errores sintácticos es el considerar reglas de producción adicionales para el control de errores.

- Se puede realizar una corrección global, se trata de encontrar la construcción sintáctica más parecida a la dada que sí pueda ser reconocida.

## **Análisis semántico**

La semántica se encarga de describir el significado de los símbolos, palabras y frases de un lenguaje, ya sea un lenguaje natural o de programación.

Hay que dotar de significado a lo que se ha realizado en la fase anterior de análisis sintáctico.

**Gramáticas atribuidas:** Con este tipo de gramáticas se asocia a cada símbolo de la gramática un conjunto de atributos y un conjunto de reglas semánticas. Se conoce como gramática atribuida, o gramática de atributos, a una gramática independiente del contexto cuyos símbolos terminales y no terminales tienen asociados un conjunto de atributos que representa una propiedad del símbolo.

**Tabla de símbolos:** La tabla de símbolos es una estructura de datos que contiene información por cada identificador detectado en la fase de análisis léxico. Por medio de la tabla de símbolos, el compilador registra los identificadores utilizados en el programa fuente reuniendo información sobre las características de dicho identificador. Estas características pueden proporcionar información necesaria para realizar el análisis semántico.

**Tratamiento de errores:** Durante el análisis semántico el compilador intenta detectar construcciones que tengan estructura sintáctica correcta pero no tengan significado para la operación implicada.

## **Generación de código intermedio.**

En un modelo en el que se realice una separación de fases en análisis y síntesis dentro de un compilador, la etapa inicial traduce un programa fuente a una representación intermedia a partir de la cual se genera después el código objeto.

Entre las ventajas de usar un código intermedio destaca el hecho de que se pueda crear un compilador para una arquitectura distinta sin tocar el front-end de un compilador ya existente para otra arquitectura. De este modo se permite después aplicar un optimizador de código independiente de la máquina a la representación intermedia.

## **Optimización de código intermedio.**

La segunda etapa del proceso de síntesis trata de optimizar el código intermedio, para posteriormente generar código máquina más rápido de ejecutar.

Unos de los tipos de optimización de código más habituales son la eliminación de variables no usadas y el desenredado de bucles.

También resulta muy habitual traducir las expresiones lógicas para que tenga que calcularse simplemente el valor de aquellos operandos necesarios para poder evaluar la expresión (evaluación en corto circuito).

Otra optimización de código típica es la traducción con precálculo de expresiones constantes.

## Generación y optimización de código objeto.

La fase final de un compilador es la generación de código objeto. Cada una de las instrucciones presentes en el código intermedio se debe traducir a una secuencia de instrucciones máquina, donde un aspecto decisivo es la asignación de variables a registros físicos del procesador.

## Compilador TINY

El compilador TINY es un compilador creado con fines didácticos para una fácil comprensión práctica acerca de los conceptos y partes de un compilador. Se utiliza como ejemplo ejecutable para las técnicas estudiadas capítulo a capítulo del libro Construcción de compiladores principios y práctica del autor estadounidense Kenneth C. Louden presentando un compilador no tan complicado como los usados en el día a día de los programadores, pero tampoco tan sencillo cuya ejecución pueda verse de manera muy reducida.

Este compilador tendrá como lenguaje fuente un lenguaje pequeño, el cual se llama lenguaje TINY. Además, el lenguaje máquina a utilizar como el lenguaje objetivo es una simplificación para que sea el lenguaje ensamblador para un procesador hipotético simple, llamado TINY Machine.

## Componentes

La estructura de este compilador se compone de los siguientes archivos

Archivos de cabecera	Archivos de código
globals.h	analyze.c
util.h	cgen.c
scan.h	code.c
parse.h	main.c
symtab.h	parse.c
analyze.h	scan.c
code.h	symtab.c
cgen.h	tm.c
	util.c

Todos los archivos a excepción de globals.h y main.c se componen de pares de archivos de cabecera/código con los prototipos de función disponibles externamente dados en el archivo de cabecera e implementados (posiblemente con funciones locales estáticas adicionales) con el archivo de código asociado. A continuación, se explica cada archivo:

**analyse.h/analyze.c:** Fase del analizador semántico del compilador (reconocimiento de la coherencia de entrada).

**cgen.h/cgen.c:** Generador de código (transformación de la entrada en una representación de código intermedio y posteriormente en código objetivo).

**code.h/code.c:** Contienen las utilerías para la generación del código que son dependientes de la máquina objetivo.

**globals.h:** Está incluido en todos los archivos del código y contiene las definiciones de los tipos de datos y variables globales utilizadas a lo largo del compilador.



**main.c:** Contiene el programa principal que controla el compilador, además de asignar e inicializar las variables globales.

**parse.h/parse.c:** Fase del analizador sintáctico (reconocimiento de la estructura del lenguaje).

**scan.h/scan.c:** Fase del analizador léxico (reconocimiento de los elementos del lenguaje).

**symtab.h/symtab.c:** Contiene una implementación de tabla de cálculo de dirección de una tabla de símbolos adecuada para usarse con el lenguaje TINY.

**tm.c:** Contiene la implementación para traducir el código TINY a ensamblador.

**util.h/util.c:** Contienen funciones de utilerías necesarias para generar la representación interna del código fuente (el árbol sintáctico) y exhibir la información de error y listado.

## Funcionamiento

A pesar de que el compilador TINY pueda verse poco realista, como se ha visto, sus archivos corresponden aproximadamente a las fases y se pueden analizar (así como también compilar y ejecutar) de manera individual. Sin embargo, a comparación de los componentes comunes que tiene un compilador, este carece de algunos componentes como la tabla de literales, el manejador de error y distintas fases de optimización.

Además, hay que agregar que no hay código intermedio separado del árbol sintáctico y la tabla de símbolos interactúa solamente con el analizador semántico y el generador de código.

Para reducir la interacción entre los archivos, el compilador TINY es capaz de dar cuatro pasadas, a continuación, se presentan en orden:

Primera pasada (se compone por el compilador léxico y analizador sintáctico): Construye el árbol sintáctico.

Segunda pasada (se compone de el análisis semántico): Construye la tabla de símbolos.

Tercera pasada (compuesta por el análisis semántico): Realiza la verificación de tipos.

Cuarta pasada: Genera el código.

## Lenguaje TINY

El lenguaje TINY es simple y poco robusto. Un programa en TINY tiene una estructura muy simple: es simplemente una secuencia de sentencias separadas mediante signos de punto y coma en una sintaxis semejante a la de Ada o Pascal. No hay procedimientos ni declaraciones. Todas las variables son variables enteras, y las variables son declaradas simplemente al asignar valores a las mismas (de modo parecido a FORTRAN o BASIC). Existen solamente dos sentencias de control: una sentencia "if" y una sentencia "repeat". Ambas sentencias de control pueden ellas mismas contener secuencias de sentencias. Una sentencia "if" tiene una parte opcional "else" y debe terminarse mediante la palabra clave "end". También existen sentencias de lectura y escritura que realizan entrada/salida. Los comentarios, que no deben estar anidados, se permiten dentro de llaves tipográficas "{}".

## Expresiones

Las expresiones en TINY también se encuentran limitadas a expresiones aritméticas enteras y booleanas. Una expresión booleana se compone de una comparación de dos expresiones aritméticas que utilizan cualesquiera de los dos operadores de comparación  $<$  y  $=$ . Una expresión aritmética puede involucrar constantes enteras, variables, paréntesis y cualquiera de los cuatro operadores enteros  $+$ ,  $-$ ,  $*$  y  $/$ . (división entera), con las propiedades matemáticas habituales. Las expresiones booleanas pueden aparecer solamente como pruebas en sentencias de control: no hay variables booleanas, asignación o E/S (entrada/salida).

Aunque TINY carece de muchas características necesarias para los lenguajes de programación reales (procedimientos, arreglos y valores de punto flotante son algunas de las omisiones más serias), todavía es lo suficientemente extenso para ejemplificar la mayoría de las características esenciales de un compilador.

## Limitaciones

El lenguaje TINY carece de características necesarias para los lenguajes de programación, por ejemplo, algunas de las omisiones más importantes son los procedimientos, arreglos y valores de punto flotante. Por otro lado, no puede trabajar con paradigmas como recursividad, y carece de estructuras de control como el `for`, `while`, `do while`, etc.

## Tokens

TINY posee tokens que caen dentro de tres categorías típicas: las palabras reservadas (de las cuales posee ocho), símbolos especiales (que permiten las cuatro operaciones aritméticas básicas, dos operaciones de comparación, la asignación de valores, el salto de línea y la separación de elementos) y tokens adicionales. En la siguiente tabla se describen los tokens y su respectiva clase:

Palabras reservadas	Símbolos especiales
If	+
then	-
else	*
end	/
repeat	=
Write	<
read	:=
Until	(
	)
	;

## Máquina TM

Empleamos el lenguaje ensamblador para esta máquina como el lenguaje objetivo para el compilador TINY. La máquina TM tiene instrucciones suficientes para ser un objetivo adecuado para un lenguaje pequeño como TINY. De hecho, TM tiene algunas de las propiedades de las computadoras con conjunto de instrucciones reducido (o RISC, de Reduced Instruction Set Computers), en que toda la aritmética y las pruebas deben tener lugar en registros y los modos de direccionamiento son muy limitados.

Nuestro simulador para la máquina TM lee el código ensamblador directamente de un archivo y lo ejecuta. De este modo, evitamos la complejidad agregada de traducir el lenguaje ensamblador a código de máquina. Sin embargo, nuestro simulador no es un verdadero ensamblador, en el sentido de que no hay etiquetas o direcciones simbólicas. Así, el compilador TINY todavía debe calcular direcciones absolutas para los saltos. También, para evitar la complejidad extra de vincularse con rutinas externas de entrada/salida, la máquina TM contiene facilidades integradas de E/S para enteros; éstas son leídas desde, y escritas hacia, los dispositivos estándar durante la simulación. El simulador TM se puede compilar desde el código fuente `tm.c` utilizando cualquier compilador C ANSI.

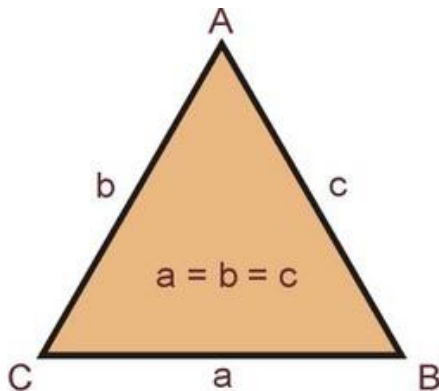
## Desarrollo

### Problema para resolver

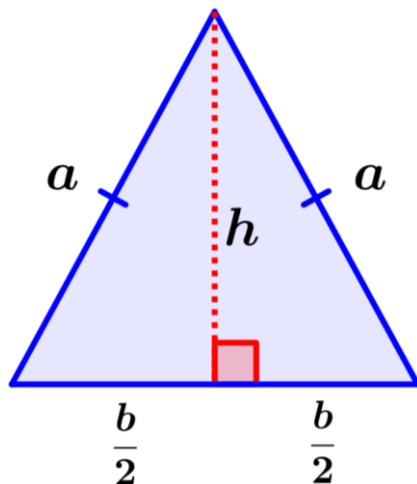
Determinar el tipo de triángulo a partir de la longitud de sus lados.

Antes que nada, se debe de definir que hace un triángulo de cierto tipo:

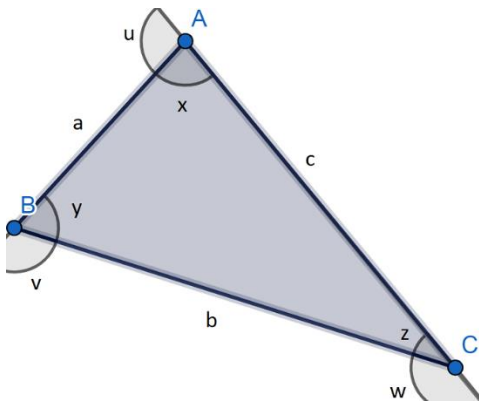
**Triangulo equilátero:** Un triángulo equilátero se define como un polígono regular, es decir, tiene cada uno de sus tres lados iguales.



**Triangulo isósceles:** Un triángulo isósceles se define como aquel que tiene dos lados con la misma longitud y uno de distinta longitud. Asimismo, los dos ángulos que están frente a los lados iguales también miden lo mismo.



**Triángulo escaleno:** Un triángulo escaleno se define como aquella figura geométrica de tres lados, cada uno de los cuales mide una longitud distinta.



## Algoritmo

Como el compilador solo maneja los comparadores menores que “<” e igual que “=”, se tiene que replantear la forma de implementar el algoritmo que en un lenguaje de programación común sería muy sencillo.

En primeras instancias se debe tener tres variables las cuales representan los tres tipos de triángulo y que van a funcionar como bandera las cuales van a estar inicializadas en cero. Después se deben tener otras tres variables para de esta manera leer cada uno de los lados del triángulo. Y a partir de aquí comienzan las comparaciones.

El caso más sencillo es el del equilátero ya que simplemente solo se tiene que comparar que sean iguales los tres lados del triángulo a través de un if anidado y en caso de cumplir las condiciones se reasigna la variable del equilátero con valor de 1.

El segundo caso sería el del isósceles que sería tener dos lados iguales y uno diferente. Como el lenguaje TINY sólo admite dos tipos de comparadores, aquí se tiene que jugar con los menores que, por lo que un caso sería tener en el if anidado del equilátero otra condición en caso de que un lado sea menor a otro o viceversa y así sucesivamente realizar otros dos if anidados para cubrir todos los casos posibles de ingreso de longitud de lados. En caso de que se cumplan las condiciones se reasigna la variable de isósceles con el valor de 2.

El tercer caso sería el del triángulo escaleno en el cual todos sus lados son diferentes. Aquí se tiene que comparar las tres variables con el operador menor que considerando las diversas formas de ingresar las variables. Si se cumplen las condiciones se reasigna la variable de isósceles con el valor de 3.

Finalmente se vuelven a poner tres condiciones en las que si alguna de las tres banderas tiene su valor esperado se imprima la variable en consola.

## Implementación

Debido a que sólo es posible ejecutar TINY en máquinas con arquitectura de 16 bits, para poder ejecutarlo en máquinas con una arquitectura diferente, es necesario instalar una máquina virtual que haga uso de esta arquitectura o hacer uso de un emulador. Por recomendación del profesor, se ha optado por instalar el emulador DOSBox.

## DOSBox

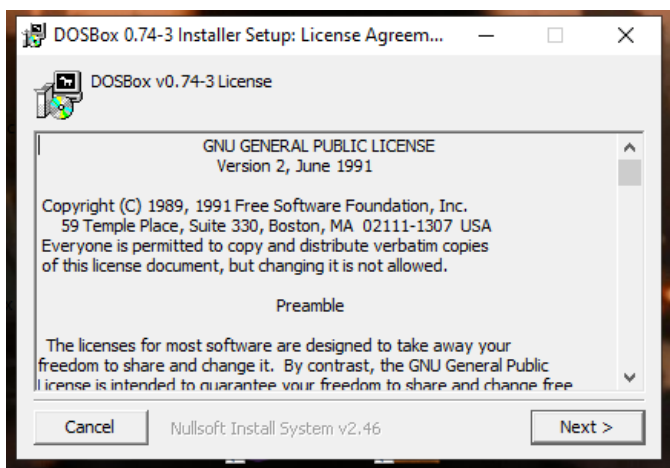
DOSBox es un emulador gratuito y de código abierto de DOS que utiliza la biblioteca SDL, lo que hace que DOSBox sea muy fácil de portar a diferentes plataformas. DOSBox ya ha sido portado a muchas plataformas diferentes, como Windows, BeOS, Linux, MacOS X, etc.

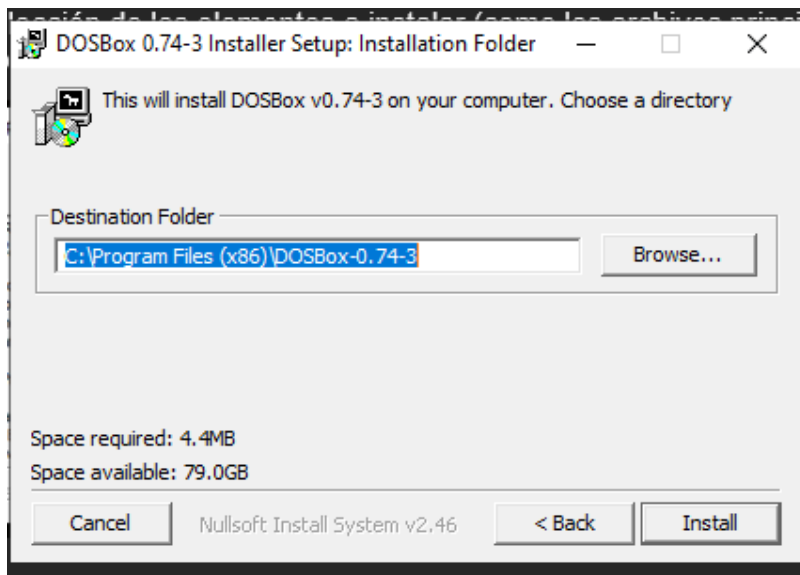
DOSBox también emula CPU: 286/386 modo real/modo protegido, sistema de archivos de directorio/XMS/EMS, gráficos Tandy/Hercules/CGA/EGA/VGA/VESA y una tarjeta SoundBlaster/Gravis Ultra Sound.

La instalación de este programa se realiza a través de su página oficial de descarga, seleccionando la versión del sistema operativo deseado



En mi caso opte usar la versión de Windows debido a que es el sistema operativo que uso actualmente. Su instalación inicia con la descarga del archivo ejecutable, la aceptación de la licencia de uso, la elección de los elementos a instalar (como los archivos principales y el acceso directo) y por último la ubicación del programa en el equipo.

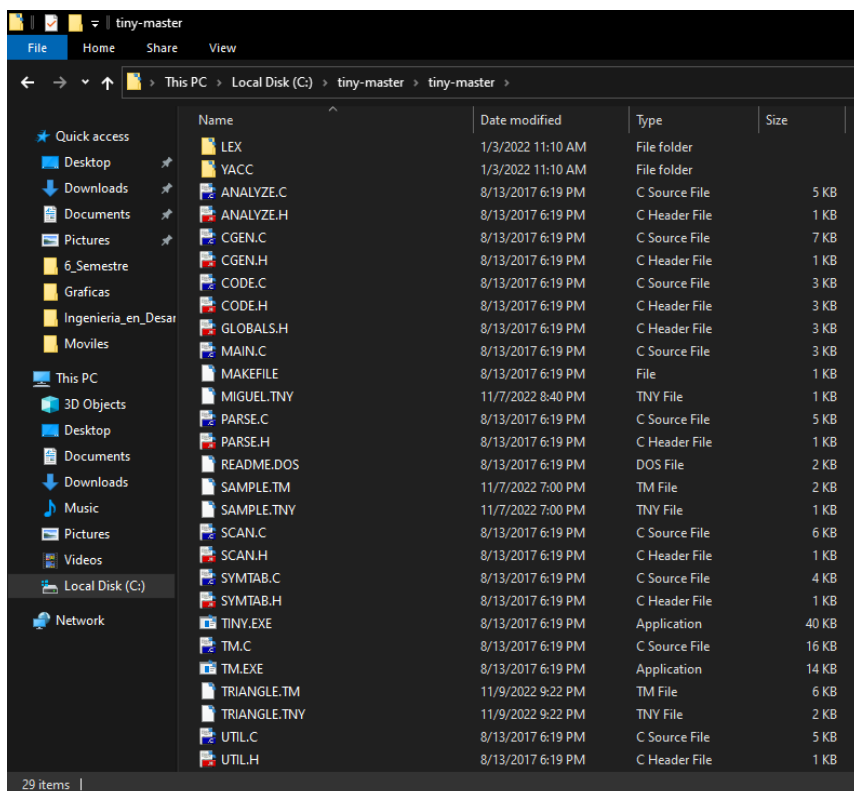




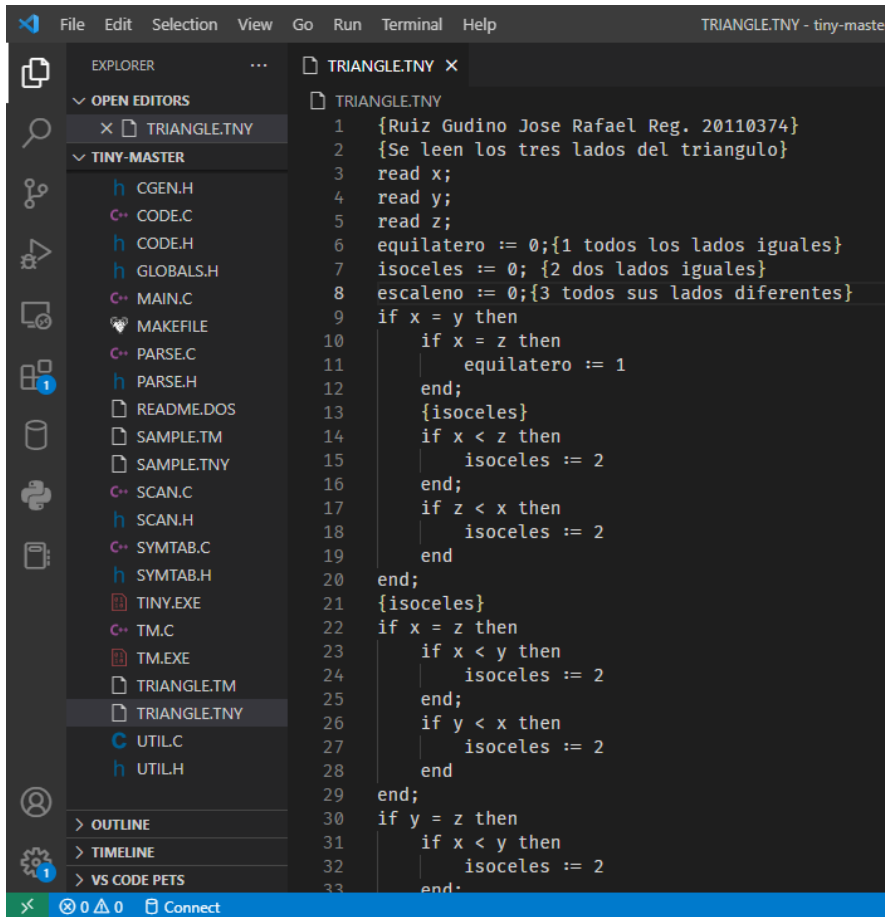
## Cargar el compilador TINY

Una vez instalado el entorno DOSBox que es donde se ejecutará el compilador, es necesario montar dicho compilador. TINY, como se ha visto, se compone de diferentes archivos, los cuales fueron proporcionados en una carpeta comprimida de parte del profesor, lo que facilitó el trabajo.

Así que una vez descargada la carpeta comprimida que contiene el compilador, se procede a descomprimirla en cualquier unidad de almacenamiento disponible. Es necesario guardar la ruta de almacenamiento, ya que será necesaria más adelante por lo que lo más conveniente es guardarla en el disco local directamente para así no estar escribiendo una ruta tan larga cada vez que se monte la unidad de almacenamiento en el emulador.



Debido a que el lenguaje TINY no es un lenguaje formal, no existen entornos de desarrollo integrado que atiendan específicamente a este lenguaje con características como la corrección de sintaxis en tiempo real. La edición de cualquier código fuente se recomienda hacer en cualquier editor de texto plano, como lo puede ser el bloc de notas de Windows, pero aun así se puede usar un IDE como Visual Studio Code para editar. El único requisito necesario para la codificación del programa es crear el archivo tipo TINY, que debe poseer la extensión “.TNY” y estar ubicado dentro de la carpeta del compilador. En este caso la codificación se hará Visual Studio Code.



The screenshot shows the Visual Studio Code interface with the TINYMASTER project open. The Explorer sidebar on the left shows the project structure, including files like CGEN.H, CODE.C, CODE.H, GLOBALS.H, MAIN.C, MAKEFILE, PARSE.C, PARSE.H, README.DOS, SAMPLE.TM, SAMPLE.TNY, SCAN.C, SCAN.H, SYMTAB.C, SYMTAB.H, TINY.EXE, TM.C, TM.EXE, TRIANGLE.TM, TRIANGLE.TNY, UTIL.C, and UTIL.H. The main editor window displays the content of TRIANGLE.TNY, which is a TINY program for triangle classification. The code includes comments in Spanish and logic to read three sides and classify the triangle as equilateral, isosceles, or scalene.

```
1 {Ruiz Gudino Jose Rafael Reg. 20110374}
2 {Se leen los tres lados del triangulo}
3 read x;
4 read y;
5 read z;
6 equilatero := 0;{1 todos los lados iguales}
7 isoceles := 0; {2 dos lados iguales}
8 escaleno := 0;{3 todos sus lados diferentes}
9 if x = y then
10   if x = z then
11     equilatero := 1
12   end;
13   {isocoles}
14   if x < z then
15     isocoles := 2
16   end;
17   if z < x then
18     isocoles := 2
19   end
20 end;
21 {isocoles}
22 if x = z then
23   if x < y then
24     isocoles := 2
25   end;
26   if y < x then
27     isocoles := 2
28   end
29 end;
30 if y = z then
31   if x < y then
32     isocoles := 2
33   end;
```

## Código del programa

```
{Ruiz Gudino Jose Rafael Reg. 20110374}
{Se leen los tres lados del triangulo}
read x;
read y;
read z;
equilatero := 0;{1 todos los lados iguales}
isocoles := 0; {2 dos lados iguales}
escaleno := 0;{3 todos sus lados diferentes}
if x = y then
  if x = z then
    equilatero := 1
  end;
  {isocoles}
  if x < z then
    isocoles := 2
```

```

    end;
    if z < x then
        isoceles := 2
    end
end;
{isocceles}
if x = z then
    if x < y then
        isoceles := 2
    end;
    if y < x then
        isoceles := 2
    end
end;
if y = z then
    if x < y then
        isoceles := 2
    end;
    if y < x then
        isoceles := 2
    end
end;
{escaleno}
if x < y then
    if y < z then
        escaleno := 3
    end;
    if z < x then
        escaleno := 3
    end
end;
if y < x then
    if x < z then
        escaleno := 3
    end;
    if y < z then
        escaleno := 3
    end;
    if z < y then
        escaleno := 3
    end
end;
{Impresion del tipo de triangulo}
if equilatero = 1 then
    write equilatero
end;
if isoceles = 2 then
    write isoceles
end;
if escaleno = 3 then
    write escaleno
end

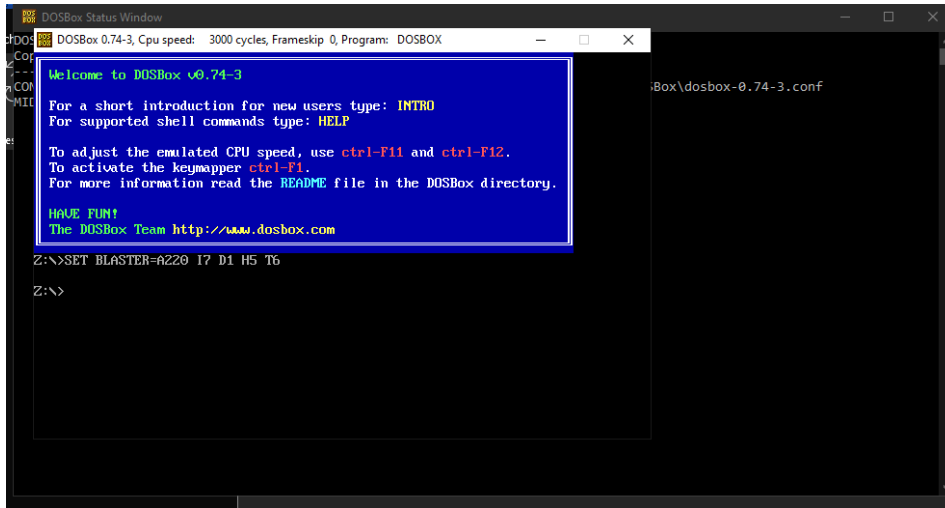
```



# Resultados

## Montaje

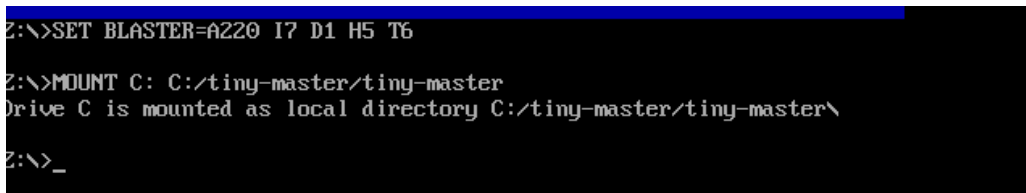
Primero se inicia la ejecución de DOSBox, el cual abrirá dos ventanas y se trabajará con la que tenga un recuadro azul. Después de esto, se tiene que el montar la unidad de almacenamiento del ordenador a la máquina virtual para así de esta manera poder usar el compilador y correr los programas.



Para montar la unidad de almacenamiento se debe usar el siguiente comando:

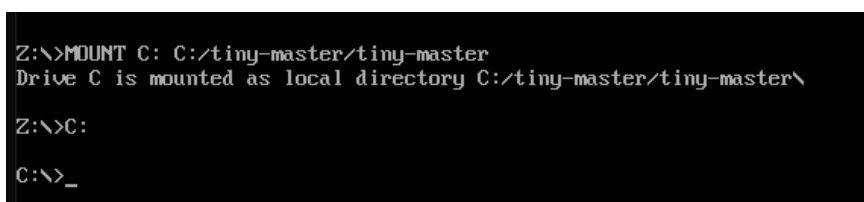
MOUNT "Letra de la unidad" "dirección del directorio del compilador"

En mi caso fue el siguiente "MOUNT C: C:/tiny-master/tiny-master". Es importante recalcar que para escribir hay que verificar donde están cada símbolo especial que se quiera escribir ya que el emulador hace uso del teclado estadounidense.



Si todo ocurre de manera correcta debe arrojar el mensaje "Drive [Letra] is mounted as local directory [dirección]".

Como se puede apreciar en la imagen anterior, DOSBox inicia por defecto en una unidad virtual llamada Z:, es por ello que el paso siguiente sería ubicar la ejecución en la unidad de almacenamiento para nuestra compilación, entonces se ingresa el nombre de la unidad donde se encuentra la carpeta tiny-master: C:



## Compilación

Considerando todo lo anterior, así como el hecho de que el archivo a compilar se encuentra dentro de la carpeta del compilador con la extensión “.TNY” (y que en caso de que se omita el compilador la agrega), se procede a compilar primeramente el programa. Esto se realiza a través del comando siguiente: “TINY NOMBRE\_ARCHIVO.TNY”

En mi caso sería el siguiente: “TINY TRIANGLE.TNY”.

```
C:\>TINY TRIANGLE.TNY
TINY COMPILATION: TRIANGLE.TNY
Building Symbol Table...
Symbol table:
Variable Name  Location  Line Numbers
-----
escaleno       5         8  41  44  49  52  62  63
equilatero     3         6  11  56  57
x              0         3   9  10  14  17  22  23  26  31  34  39
  40  43  47  48
y              1         4   9  23  26  30  31  34  39  47  51
z              2         5  10  14  17  22  30  40  43  48  51
isocoles       4         7  15  18  24  27  32  35  59  60
Checking Types...
Type Checking Finished
C:\>
```

En caso de ser compilado con éxito mostrará una tabla de símbolos con las variables utilizadas en el programa, la localidad y las líneas de código en las que se mencionan. Además, en la carpeta del compilador, debido a que es la que contiene el código fuente, se generará un archivo de código objetivo con extensión “.tm” el cual podrá ser utilizado por la máquina TM más adelante.

En el caso de que el programa se haya compilado exitosamente, en consola se mostrará una tabla de símbolos con las variables utilizadas en el programa, su localización y además de las líneas de código donde se usan. Asimismo en la carpeta del compilador se generara un nuevo archivo de código objetivo que tiene extensión “.TM” el cual será utilizado por la máquina TM para así ejecutar el programa.

TM.C	8/13/2017 6:19 PM	C Source File	16 KB
TM.EXE	8/13/2017 6:19 PM	Application	14 KB
TRIANGLE.TM	11/9/2022 10:42 PM	TM File	6 KB
TRIANGLE.TNY	11/9/2022 9:22 PM	TNY File	2 KB
UTIL.C	8/13/2017 6:19 PM	C Source File	5 KB
UTIL.H	8/13/2017 6:19 PM	C Header File	1 KB

Si se llegara el caso de que el archivo no se compilará exitosamente se mostrará en consola los errores señalando la línea en la que se encuentra el error y el tipo de error que sucedió.

```
C:\>TINY TRIANGLE.TNY
TINY COMPILATION: TRIANGLE.TNY
>>> Syntax error at line 17: unexpected token -> ;
>>> Syntax error at line 18: unexpected token -> ID, name= isocoles
C:\>
```

## Ejecución

Ya que la compilación haya sido exitosa y se generó el archivo “.TM”, a través del siguiente comando se hará uso de la máquina TM para así ejecutar el programa realizado:

“TM NOMBRE\_ARCHIVO.TM”

En este caso el comando es el siguiente: “TM TRIANGLE.TM”.

Después para poder proseguir con la ejecución del programa realizado, se debe de escribir el comando “go”.

```
C:\>TM TRIANGLE.TM
TM simulation (enter h for help)...
Enter command: go
Enter value for IN instruction: _
```

Posteriormente, por la naturaleza del programa, se deben ingresar tres números que corresponden a las longitudes de cada lado del triángulo para así determinar de qué tipo es.

## Prueba triángulo equilátero

```
TM simulation (enter h for help)...
Enter command: go
Enter value for IN instruction: 5
Enter value for IN instruction: 5
Enter value for IN instruction: 5
OUT instruction prints: 1
HALT: 0,0,0
Halted
Enter command:
```

Como se puede apreciar, al ingresar tres números iguales se imprime en consola el número uno que corresponde al triángulo equilátero.

## Prueba triángulo isósceles

<pre>C:\&gt;TM TRIANGLE.TM TM simulation (enter h for help)... Enter command: go Enter value for IN instruction: 4 Enter value for IN instruction: 2 Enter value for IN instruction: 4 OUT instruction prints: 2 HALT: 0,0,0</pre>	<pre>C:\&gt;TM TRIANGLE.TM TM simulation (enter h for help)... Enter command: go Enter value for IN instruction: 2 Enter value for IN instruction: 4 Enter value for IN instruction: 4 OUT instruction prints: 2 HALT: 0,0,0</pre>	<pre>TM simulation (enter h for help)... Enter command: go Enter value for IN instruction: 4 Enter value for IN instruction: 4 Enter value for IN instruction: 2 OUT instruction prints: 2 HALT: 0,0,0</pre>
--	--	--

Como se puede apreciar en esta otra prueba, al ingresar dos números iguales y uno distinto sin importar el orden en que se ingresen, se imprime en consola el número dos que corresponde al triángulo isósceles.

## Prueba triángulo escaleno

```
C:\>TM TRIANGLE.TM
TM simulation (enter h for help)...
Enter command: go
Enter value for IN instruction: 1
Enter value for IN instruction: 2
Enter value for IN instruction: 3
OUT instruction prints: 3
HALT: 0,0,0
C:\>TM TRIANGLE.TM
TM simulation (enter h for help)...
Enter command: go
Enter value for IN instruction: 3
Enter value for IN instruction: 1
Enter value for IN instruction: 2
OUT instruction prints: 3
HALT: 0,0,0
```

```
C:\>TM TRIANGLE.TM
TM simulation (enter h for help)...
Enter command: go
Enter value for IN instruction: 3
Enter value for IN instruction: 2
Enter value for IN instruction: 1
OUT instruction prints: 3
HALT: 0,0,0
```

```
C:\>TM TRIANGLE.TM
TM simulation (enter h for help)...
Enter command: go
Enter value for IN instruction: 2
Enter value for IN instruction: 3
Enter value for IN instruction: 1
OUT instruction prints: 3
HALT: 0,0,0
```

Como se puede apreciar en esta otra prueba, al ingresar tres números distintos sin importar el orden en que se ingresen, se imprime en consola el número tres que corresponde al triángulo escaleno.

Obsérvese que la ejecución termina con el mensaje “HALT: 0,0,0” lo cual indica que se terminó de ejecutar el programa correctamente. Para finalizar la emulación de la máquina TM es necesario escribir el comando quit en minúsculas o simplemente “q” y para salir de la consola se puede utilizar el comando exit.

```
HALT: 0,0,0
Halted
Enter command: q
Simulation done.
```

## Conclusiones

El documento ha presentado el cumplimiento del objetivo de realizar un programa en lenguaje y compilado en TINY resolviendo la problemática de determinar si un triángulo es equilátero, isósceles o equilátero dadas las longitudes de sus tres lados. Este trabajo ha conllevado cierto grado de complejidad ya que primeramente hay que comprender el funcionamiento del compilador y sus necesidades para que pueda correr (arquitectura de 16 bits), además que la parte un poco más complicada es adaptarse a su lenguaje de programación, el cual es mucho mas limitado que cualquier otro lenguaje convencional, por lo que ciertamente la complejidad de realizar cualquier programa (por mas sencillo que se pueda ver) aumenta en este lenguaje.

A través del uso del compilador me pude dar cuenta de lo complejo que pueden llegar a ser para poder correr un lenguaje de programación, ya que debemos de recordar que un compilador pasa por diversas fases: el análisis léxico, análisis semántico y análisis sintáctico para después pasar a la optimización y pasar a ensamblador y código máquina. Por lo que al realizar este trabajo pude repasar y reforzar todos los conocimientos que se vieron de autómatas y gramática durante el transcurso del semestre. Y aunque TINY es pequeño por sus fines educativos fue interesante ver cómo se requieren muchas partes y puede llegar a resultar ser muy complejo el desarrollo de un compilador.

Para mí el verdadero reto fue realizar el programa, debido las limitadas operaciones lógicas del lenguaje. Esto porque mi problemática era determinar el tipo de triángulo dados las longitudes de los lados, que, aunque suena muy sencilla y se podría hacer en unas muy pocas líneas a través de estructuras de control selectivas en cualquier lenguaje de programación común, debemos recordar que tiny solo maneja “menor que” e “igual que”. Por lo que tuve que cambiar mi lógica de programador y modificar mi perspectiva de ver el problema para así poder darle una solución, lo cual implicó agregar más líneas de código para considerar todos los posibles casos usando solo esas dos estructuras de control selectivas. También hay que agregar que este lenguaje hace uso de una sintaxis muy poco convencional, la cual puede llegar a confundir e incluso frustrar si no se sabe bien lo que se está haciendo.

Pero al final fue satisfactorio poder contemplar como pude realizar este reto satisfactoriamente y observar que el programa compilara y se ejecutara de manera exitosa. Por lo que el conocimiento adquirido se puede ver reflejado en el trabajo presentado.

## Referencias

1. colaboradores de Wikipedia. (2022, 18 agosto). *Compilador*. Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/Compilador>
2. *Estructura de un compilador*. (2013, 26 julio). Cursos gratis. <https://conocimientosweb.net/dcmf/ficha22734.html>
3. Louden, K. C. (2004). *Construcción de compiladores: principios y práctica*. Thomson.
4. Westreicher, G. (2020, 12 noviembre). *Triángulo*. Economipedia. <https://economipedia.com/definiciones/triangulo.html>