



TECNOLÓGICO
NACIONAL DE MÉXICO®



El lenguaje de programación



Vera Reyes Rafael de Jesús

Índice de contenido

Tablas.....	II
Figuras	II
Capítulo 2. Manual de Usuario.....	1
Introducción	2
Instalación	3
Editor integrado	3
Estructura del lenguaje	10
Tipos de datos.....	10
Variables	11
Operadores	12
Funciones integradas	14
Estructuras de control	16
Condicional.....	16
Ciclo While.....	17
Ciclo For	18
Funciones	19
Comentarios.....	24
Compilación.....	25

Índice de Tablas y Figuras

Tablas

Tabla 1 - Tipos de datos	11
--------------------------------	----

Figuras

Ilustración 1 - Interfaz de inicio	4
Ilustración 2 - Menú Archivo.....	5
Ilustración 3 - Archivo nuevo.....	6
Ilustración 4 - Menú Editar	7
Ilustración 5 - Menú Compilador	8
Ilustración 6 - Menú Ayuda	9
Ilustración 7 - Acerca del compilador	10
Ilustración 8 - Variables no inicializadas.....	11
Ilustración 9 - Variables inicializadas	12
Ilustración 10 - Operadores aritméticos.....	13
Ilustración 11 - Operadores booleanos	14
Ilustración 12 - Ejemplo usando la función print	15
Ilustración 13 - Ejemplo usando la función input	16
Ilustración 14 - Ejemplo condicional if anidada.....	17
Ilustración 15 - Ejemplo ciclo while	18
Ilustración 16 - Ejemplo ciclo for	19
Ilustración 17 - Ejemplo de función	20
Ilustración 18 - Ejemplo función sin parametros.....	21
Ilustración 19 - Ejemplo función con parametros.....	21
Ilustración 20 - Ejemplo función sin return	22
Ilustración 21 - Ejemplo función con return	23
Ilustración 22 - Ejemplo completo de función	24
Ilustración 23 - Ejemplo de comentarios	25
Ilustración 24 - Ejemplo de archivos de compilación	26

Manual de Usuario

Introducción

En el presente documento se presenta la forma en que uno como usuario puede hacer uso del Compilador CMM y crear programas con el lenguaje C--, mostrando desde el proceso de instalación hasta como crear programas para un mejor aprovechamiento del mismo.

Este lenguaje es un lenguaje funcional y compilado, tomando principal inspiración del lenguaje C, pero haciéndolo mucho más sencillo y fácil de usar de manera que alguien que apenas está iniciando en la programación pueda hacerlo de una manera más amigable. Además, cabe mencionar que solo está disponible para sistemas Linux.

Instalación

Lo primero que se debe hacer antes de empezar a trabajar es tener el ambiente de trabajo listo para usar, es por ello que en esta sección se menciona como realizar el proceso de instalación tanto del compilador a utilizar como el software adicional.

El compilador CMM es un compilador que funciona exclusivamente para sistemas Linux de 64 bits, la distribución puede ser cualquier derivado de Debian o Red Hat, los pasos para instalación se presentan para ambos sistemas.

El compilador CMM está desarrollado en Java y para funcionar necesita tenerlo instalado, al menos su máquina virtual; la versión con la que trabaja es con Java 17 o superior, esto se puede comprobar fácilmente abriendo una terminal y ejecutando el siguiente comando:

```
java -version
```

Este debería de mostrar la versión de Java instalado, en caso de que no se encuentre en el sistema, la terminal lo mencionará.

Otro software que se requiere tener instalado es el ensamblador Netwide Assembler (nasm), este es el ensamblador que utiliza el compilador CMM, se encuentra fácilmente para instalar en sistemas Linux, el comando para instalarlo es el siguiente:

Para sistemas basados en Debian:

```
sudo apt install nasm
```

Para sistemas basados en Red Hat:

```
sudo dnf install nasm
```

Una vez se tenga instalado estos paquetes se podrá hacer uso del compilador CMM, este viene empaquetado en un archivo con extensión .jar, este es un empaquetado que utiliza Java para sus programas lo cual ayuda en la portabilidad, una vez se tenga el archivo jar, dependiendo del sistema, este puede ejecutarse haciendo doble click sobre él, en algunas distribuciones permite esto, pero si no, aún se puede lanzar la ejecución desde la terminal con el siguiente comando:

```
java -jar CompiladorCMM.jar
```

Editor integrado

El compilador cuenta con un editor integrado que hace más fácil el uso del mismo, este cuenta con diversas opciones para poder crear archivos para este lenguaje, así como también cargar y cerrar archivos, además de esto, permite la compilación de los archivos.

En un principio cuando se abre el editor se puede ver una interfaz simple, esto porque no tiene abierto ningún archivo.

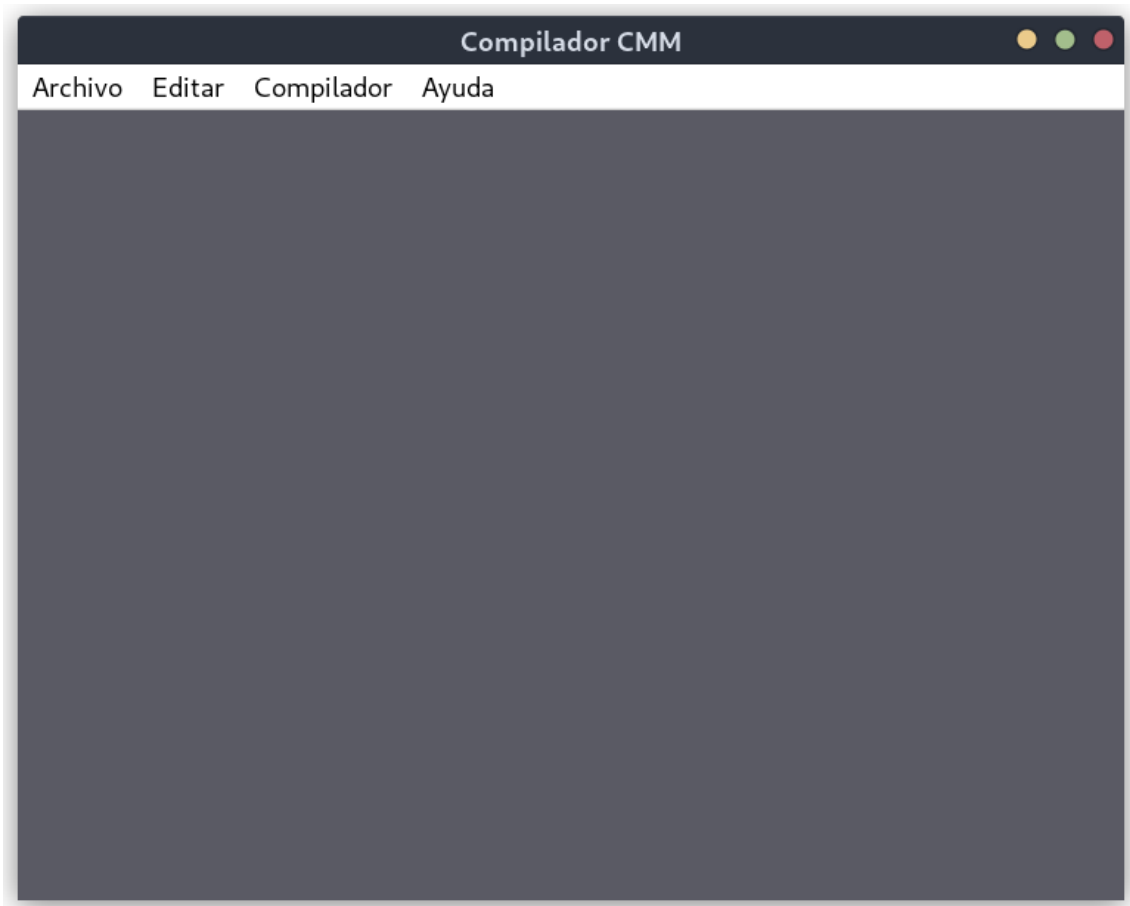


Ilustración 1 - Interfaz de inicio

En la parte superior se puede observar un menú, este cuenta con diversas opciones, la primera que se puede ver es la sección **Archivo**, en él se puede abrir o crear un archivo, en caso de que ya se tenga abierto un archivo se puede cerrar o simplemente guardar, además se tiene la opción de cerrar el editor, aunque esto también se puede hacer presionando en la opción de cerrar ventana.

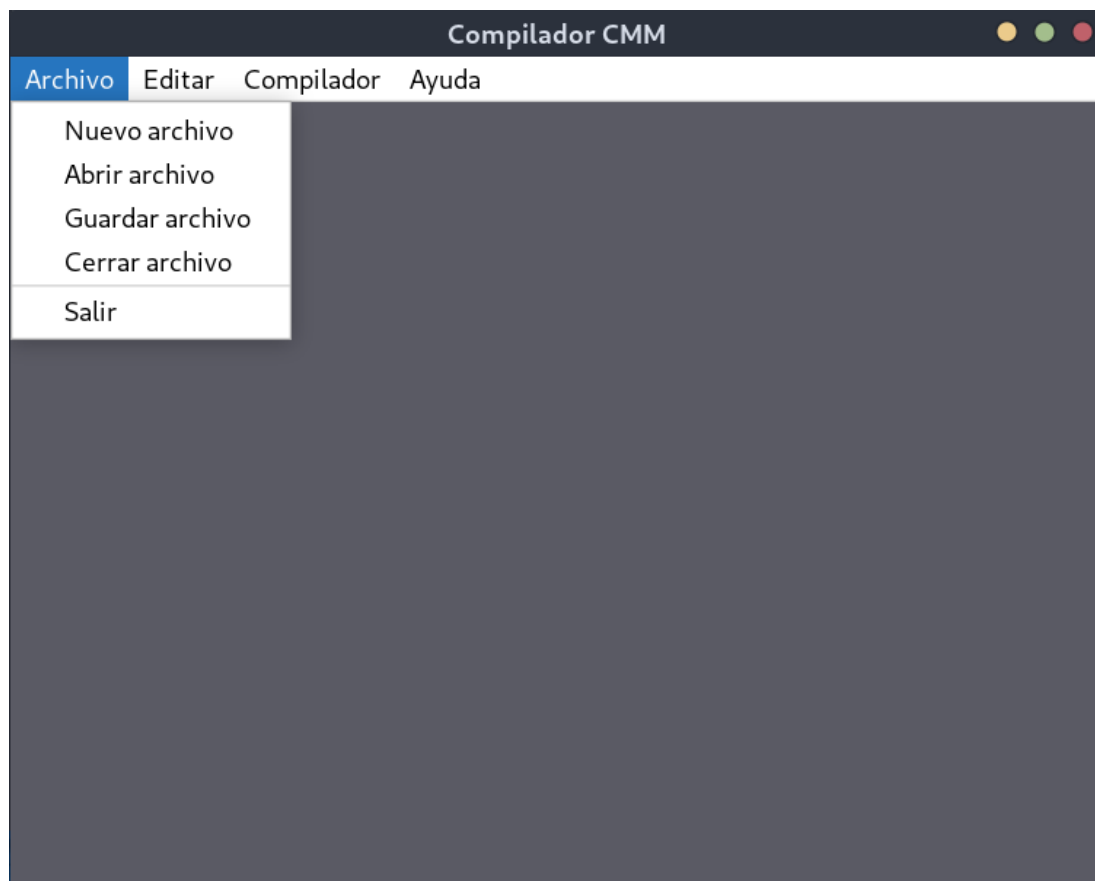


Ilustración 2 - Menú Archivo

En caso de que se cree un nuevo archivo, este se mostrará en el centro como se puede ver a continuación.

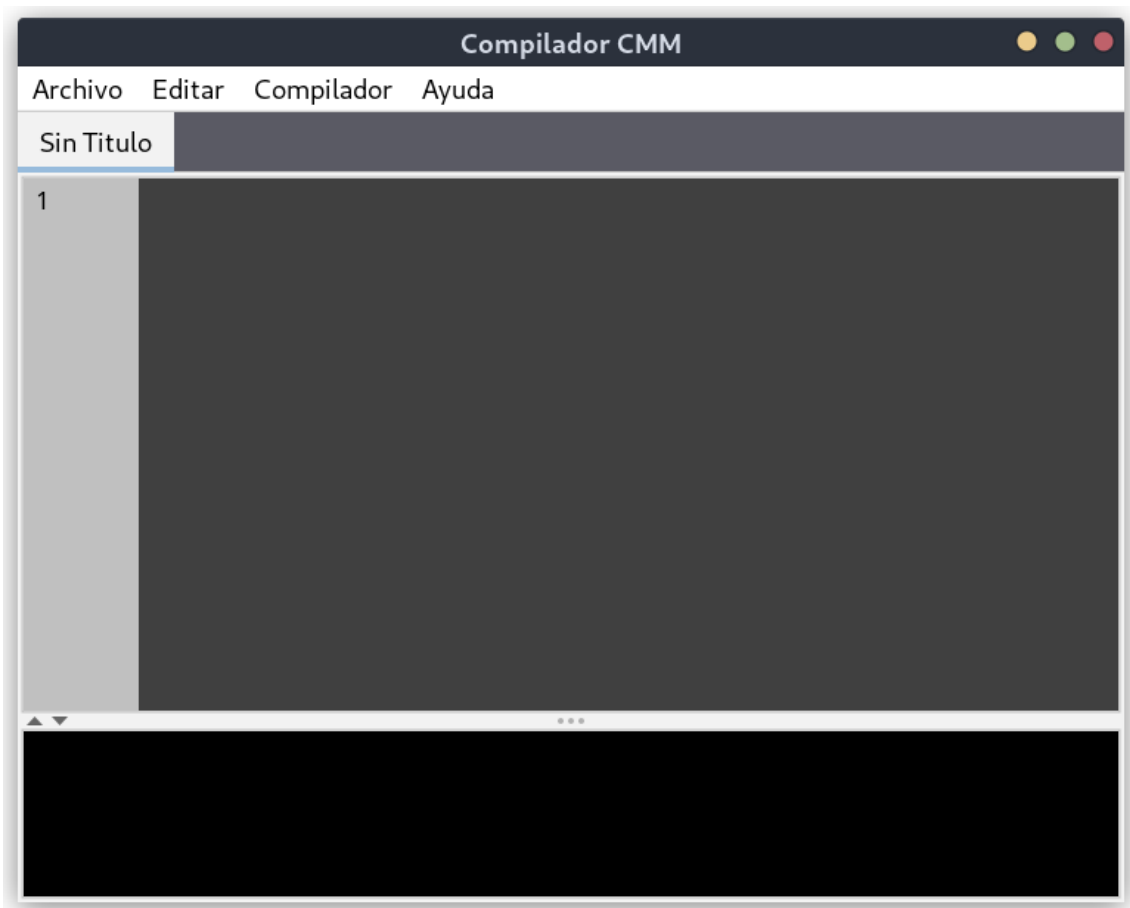


Ilustración 3 - Archivo nuevo

Este archivo se crea sin título y si se desea compilar pedirá guardar el archivo, además, nótese que aparece una sección en negro en la parte inferior, esta es la sección de mensajes y es donde se mostrará avisos por parte del compilador, en caso de presentarse algún error ahí se indicará o por su contraparte, si la compilación tiene éxito, solo mostrará un mensaje de compilación exitosa.

La segunda sección que tenemos es la de **Editar**, en esta sección podemos elegir opciones sobre el portapapeles, por ejemplo, copiar, pegar o cortar, en caso de que el usuario desee copiar un texto seleccionado en el editor al portapapeles, puede hacerlo con la opción de **copiar**, por el contrario si lo que desea es quitarlo del editor y pasarlo al portapapeles puede seleccionar la opción de **cortar** o, si tiene algún texto en el portapapeles y quiere incluirlo en el editor lo puede hacer con la opción **pegar**.

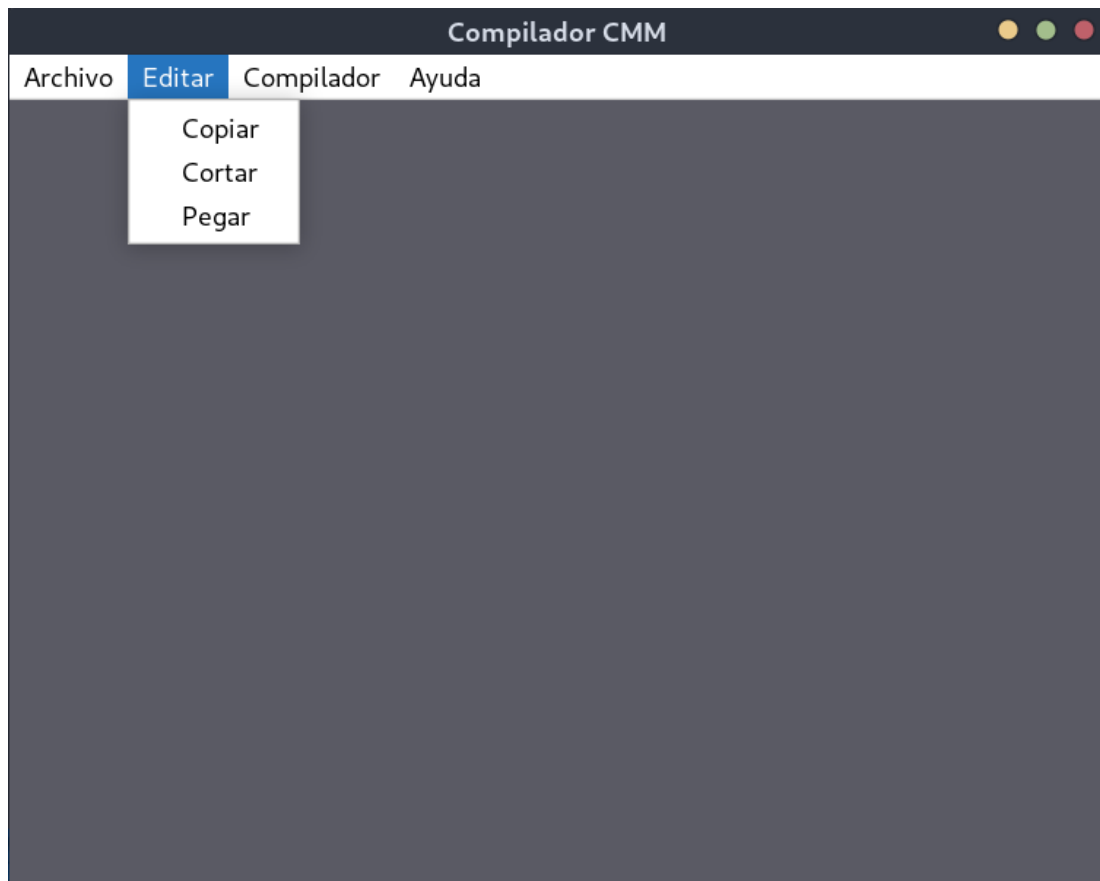


Ilustración 4 - Menú Editar

La siguiente sección que se encuentra se llama **Compilador** y es referente a la compilación, en él solo se encuentra una opción y es la que permite compilar el archivo que se encuentre abierto en el editor.



Ilustración 5 - Menú Compilador

La última sección que se encuentra es la de **Ayuda**, esta sección solo tiene una opción llamada **Acerca de** la cual muestra información referente al editor y al compilador.

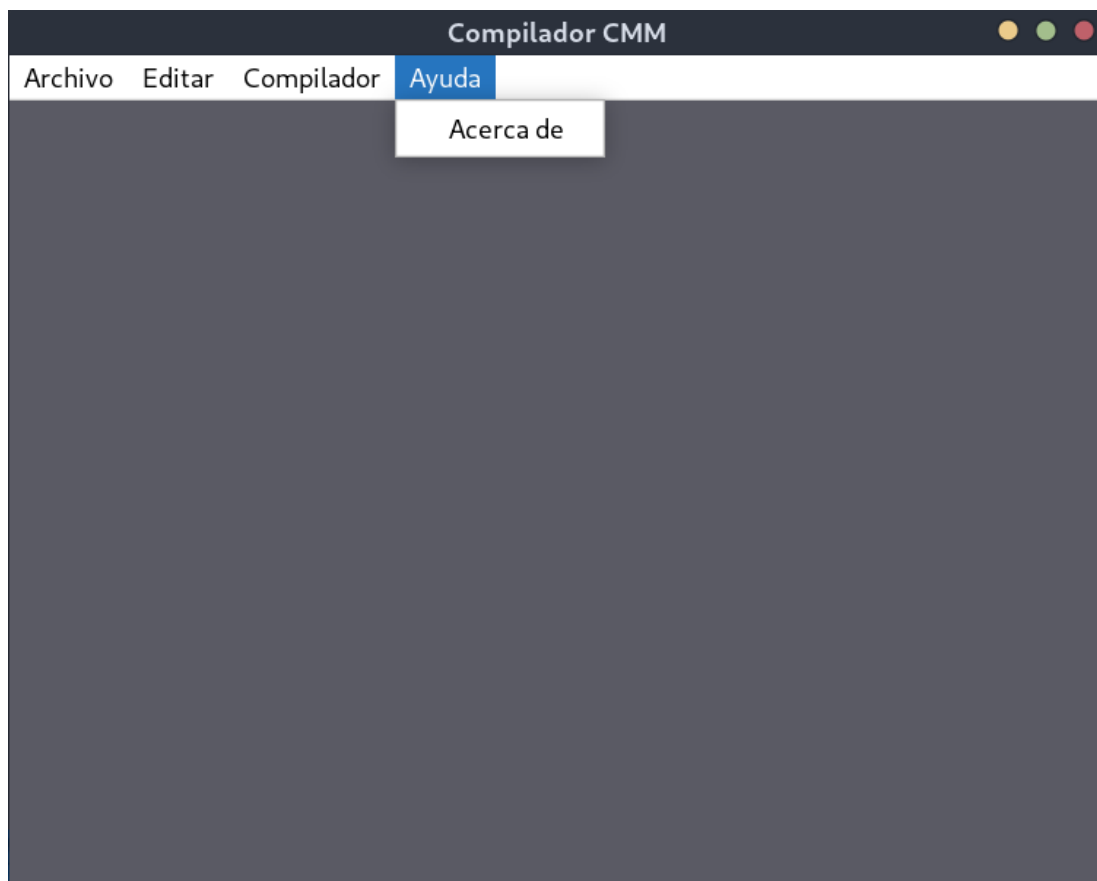


Ilustración 6 - Menú Ayuda

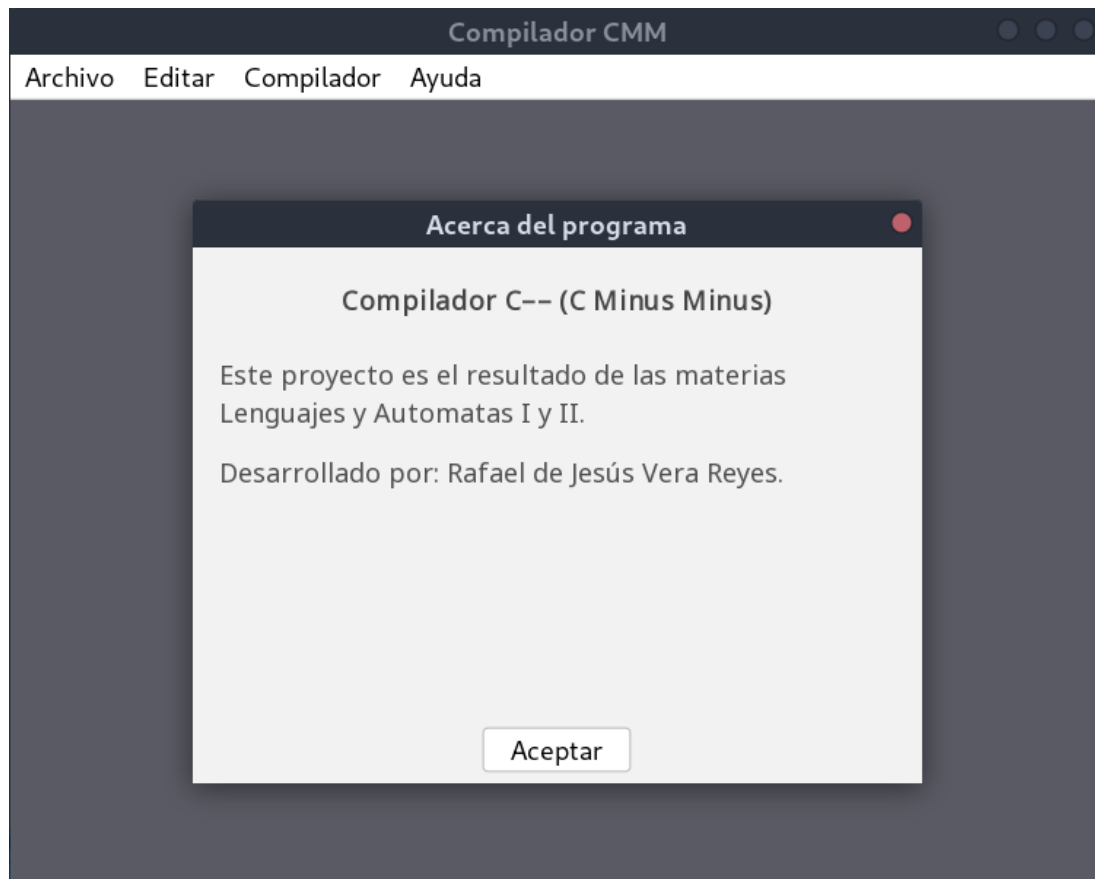


Ilustración 7 - Acerca del compilador

Estructura del lenguaje

El lenguaje C-- es un lenguaje que sigue diversas reglas y los programas hechos con él deben contar con cierta estructura para su correcto funcionamiento, por ejemplo, todos los programas deben iniciar con una función main (el concepto de función se abordará más adelante en este documento) el cual indica el inicio de la ejecución del programa, así como también el uso de punto y coma (;) para indicar el final de una instrucción, entre otras reglas que debe seguir un programa para poder funcionar.

Tipos de datos

El lenguaje C-- es un lenguaje tipado y cuenta con diferentes tipos de datos que permitan una mejor abstracción sobre lo que se está haciendo, por ejemplo, podemos trabajar con números o con cadenas de caracteres que formen un texto, el manejar los tipos de datos beneficia a evitar errores y tener un mejor orden con lo que se desee que el programa haga, de esta manera, si hacemos operaciones aritméticas, esperamos que se trabaje con números; este lenguaje cuenta con los tipos de dato **int** (integer, entero en español), **char** (character, carácter en español), **string** (character string, cadena de caracteres en español) y **bool** (boolean, booleano en español), estos tipos de datos son los básicos de los lenguajes de alto nivel y permiten crear desde programas sencillos hasta programas más avanzados, además, contar con tipos de datos permite evitar errores de compilación y ejecución permitiendo entender mejor lo que el programa está realizando.

<i>Tipo de dato</i>	<i>Alcance del tipo de dato</i>
<i>int</i>	Números enteros positivos entre 0 y 65,536^2-1.
<i>char</i>	Un solo carácter.
<i>string</i>	Varios caracteres que no rebasen los 255.
<i>bool</i>	true o false

Tabla 1 - Tipos de datos

Variables

Las variables son un espacio en memoria reservado para almacenar valores del tipo de dato dado, esto significa que si defino una variable que almacene valores numéricos (int), entonces, solo podrá almacenar valores de ese tipo de dato. Para definir una variable se debe especificar primeramente el tipo de dato, de ahí, definir el nombre que tendrá dicha variable, el nombre de variable debe tener como primer carácter una letra minúscula, mayúscula o guion bajo, el resto de caracteres que conforme el nombre de la variable puede ser los antes mencionados o un número.

Ejemplos de nombre de variables pueden ser: ***día***, ***Hola***, ***numero1***, ***nombre_variable***, ***SALUDO***, ***a***, ***_b***, ***_***, entre otros.

Además, se puede definir una variable con un valor inicial, a esto se le conoce como variable inicializada, y permite que una variable no quede con un valor vacío cuando se crea, para esto se debe definir con un símbolo de igual (=) después del nombre de la variable y después de dicho símbolo escribir el valor que tendrá dicha variable.

A continuación, se presenta ejemplos de variables inicializadas y no inicializadas.

The screenshot shows a window titled "Compilador CMM" with a menu bar (Archivo, Editar, Compilador, Ayuda) and a file tab "hola-mundo.cmm". The code editor contains the following C code:

```

1 void main() {
2     int numero;
3     char caracter;
4     string texto;
5     bool booleano;
6 }

```

Below the code editor, a status bar displays the message "Compilación finalizada." (Compilation finished).

Ilustración 8 - Variables no inicializadas

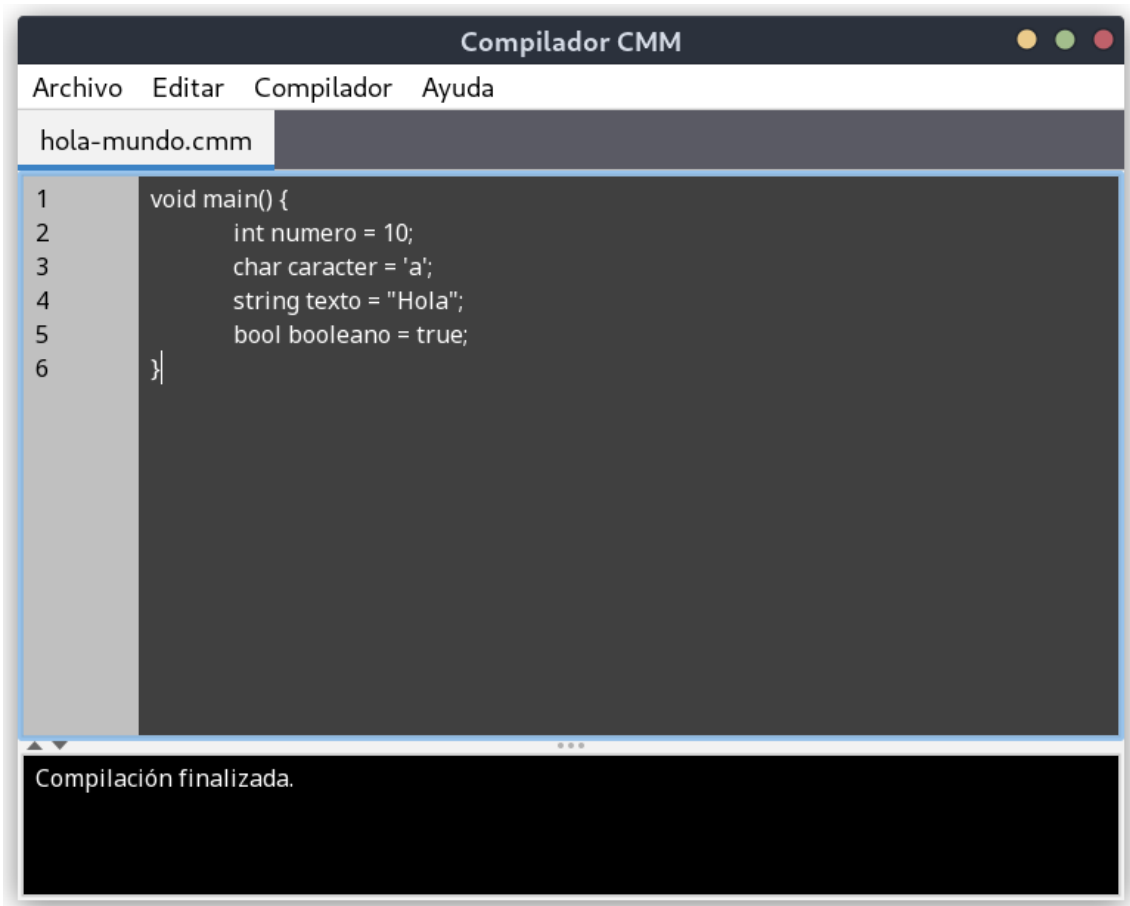


Ilustración 9 - Variables inicializadas

Operadores

Los operadores permiten realizar modificaciones sobre los valores almacenados en las variables y estas se dividen en dos principales, los operadores aritméticos y los operadores lógicos. Los operadores aritméticos permiten crear operaciones básicas como lo son suma, resta, multiplicación y división, mientras que los operadores lógicos permiten operaciones que den como resultado un verdadero o falso, por ejemplo, una compuerta lógica AND u OR, también hay operaciones como mayor que (>), mayor o igual que (>=), menor que (<), menor o igual que (<=) que permiten comprobar valores numéricos y operadores como igual que (==) o diferente que (!=), los cuales permiten comprobar si el valor de una variable es igual o diferente a otro valor dado.

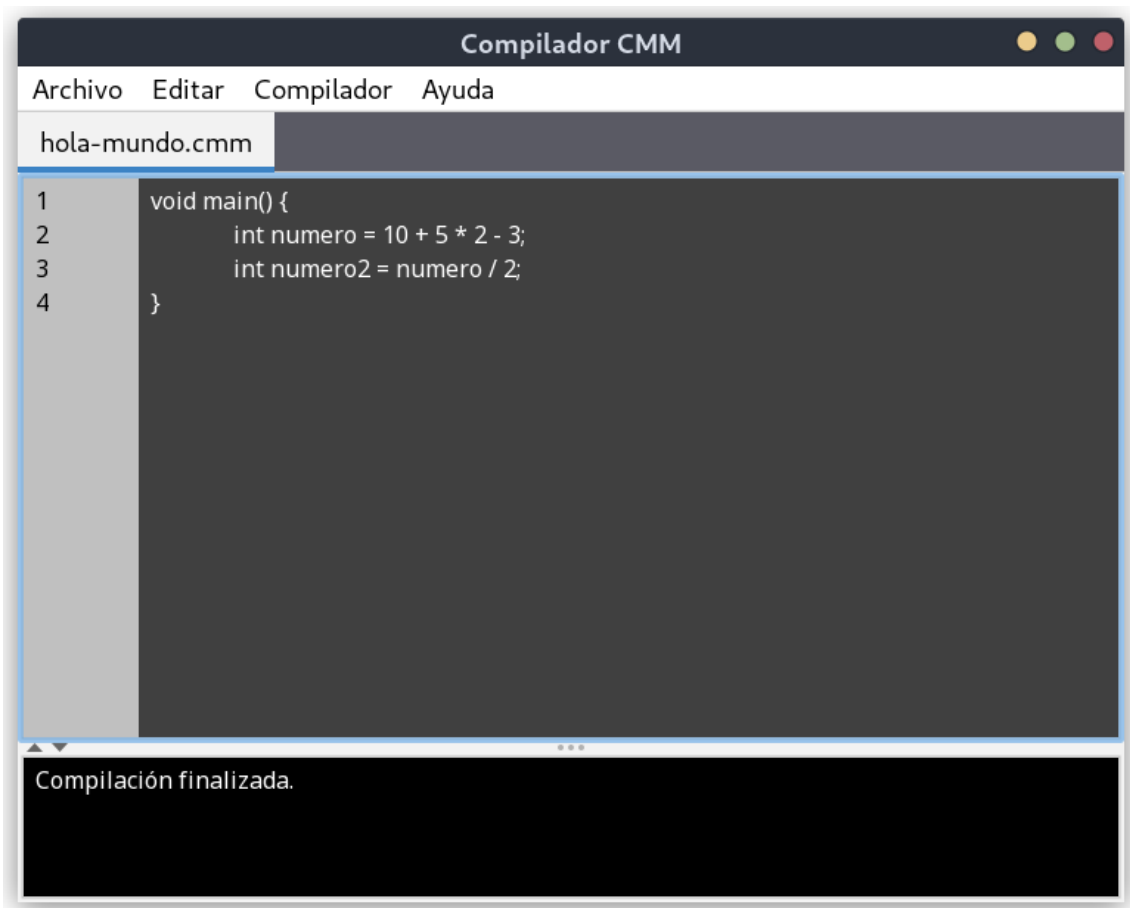


Ilustración 10 - Operadores aritméticos

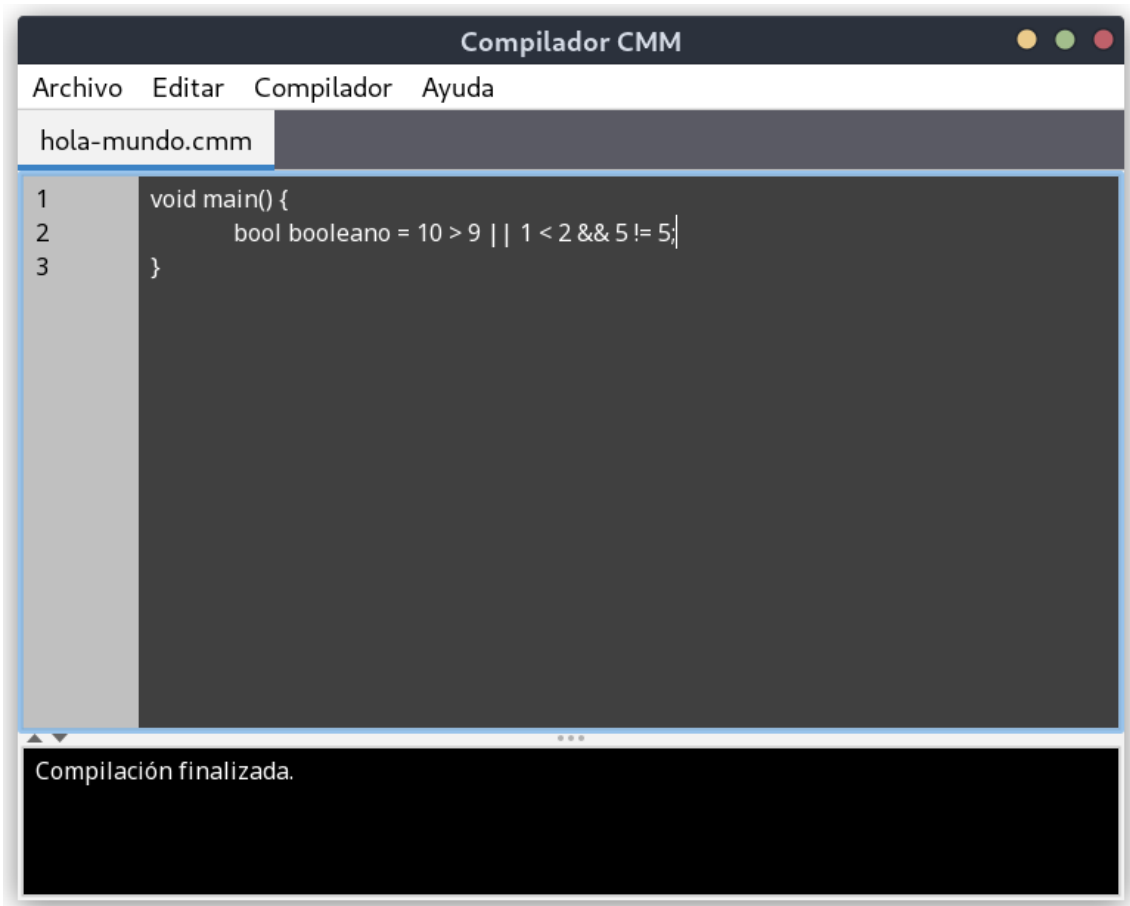


Ilustración 11 - Operadores booleanos

Funciones integradas

El concepto de función aún no se aborda, pero se puede definir rápidamente como una instrucción escrita internamente que realiza una acción dada, pero la acción o acciones que realice dependerá de lo que se le defina, esto se podrá entender mejor una vez se mire más a detalle el funcionamiento de estas instrucciones.

El lenguaje C++ cuenta con dos funciones integradas, una se llama ***print()*** y la otra se llama ***input()***.

La función ***print()*** permite mostrar en pantalla el valor de una variable, esto significa que si tengo definida una variable de tipo ***string*** llamada ***hola*** que almacena el texto ***“Hola mundo”***, si uno como programador quisiera mostrar este texto en pantalla debería de hacer uso de la función ***print()*** y mandarle la variable que se desea imprimir, en este caso la variable ***hola***, pero para mandar esta variable se debe definir el nombre de la variable cuyo valor se desee mostrar en pantalla dentro de los paréntesis definidos en la función, como se muestra en el siguiente ejemplo.

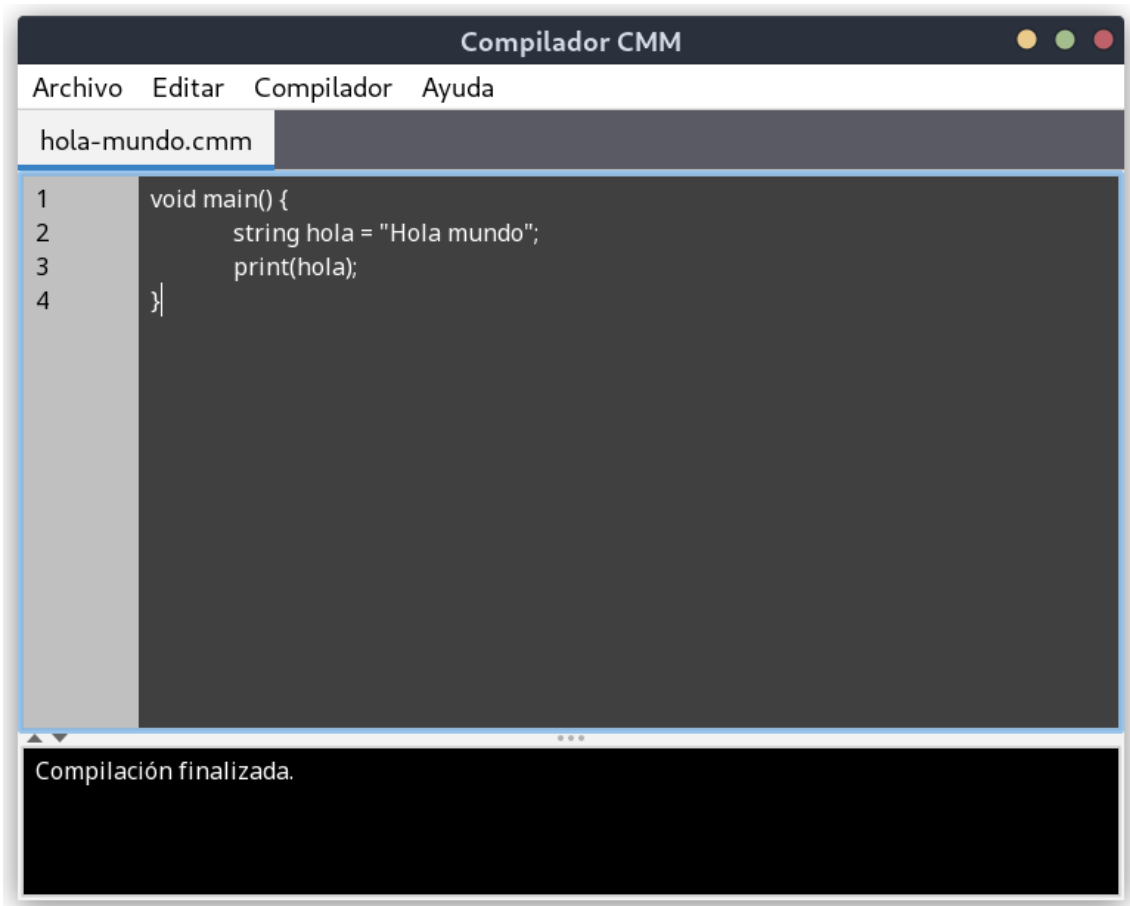


Ilustración 12 - Ejemplo usando la función print

La función ***input()*** permite ingresar valores durante la ejecución del programa almacenando el valor ingresado en una variable, estos valores varían dependiendo del tipo de dato de la variable y se debe mandar la variable donde se desee almacenar el valor que ingrese el usuario, a continuación, se muestra un ejemplo de cómo usar la función.

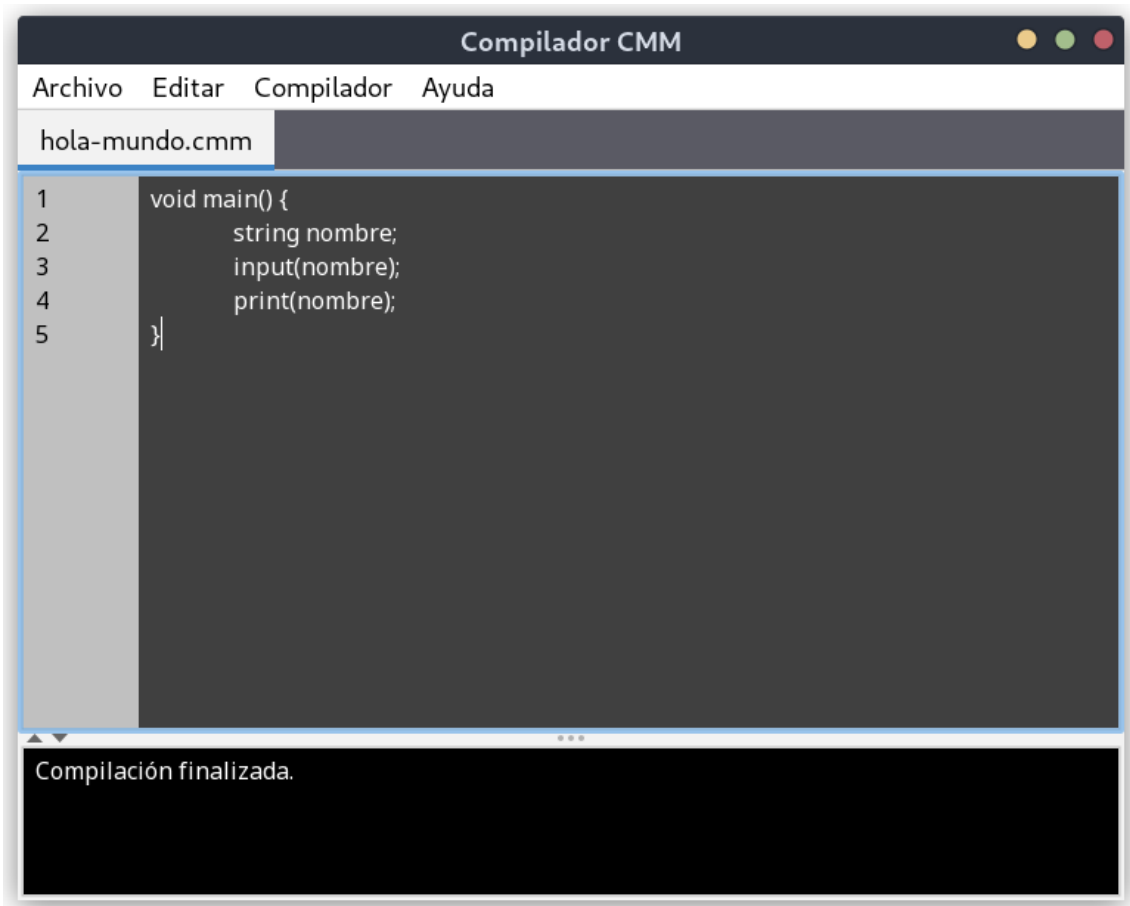


Ilustración 13 - Ejemplo usando la función input

Estructuras de control

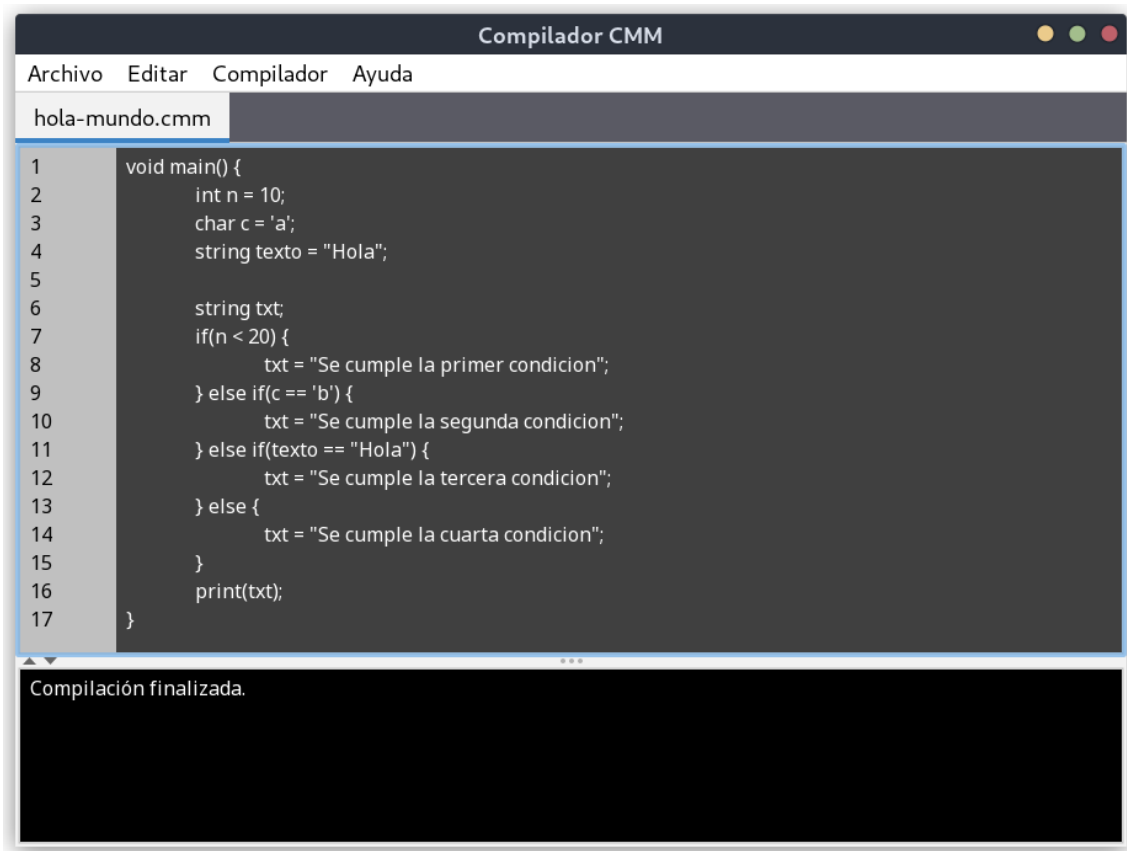
El lenguaje de programación cuenta con estructuras de control, estas son instrucciones que sirven para definir el rumbo que debe tomar el programa durante la ejecución. Llegado a cierto punto, estas estructuras pueden hacer que el programa realice ciertas acciones u otras, o que realice determinadas acciones durante repetidas veces.

Condicional

La estructura condicional se basa en la comprobación de una condición para realizar dichas acciones definidas, incluso, en caso de que dicha condición no se cumpla se puede definir que rumbo debe tomar la ejecución del programa, para esto se utiliza la sentencia **if** la cual permite verificar si una condición dada es verdadera o falsa, esto lo hace mediante valores booleanos, donde el resultado depende de si es verdadero o falso, por ejemplo, verificar si un número es mayor, mayor o igual, menor, o menor o igual, incluso para valores no numéricos, si una cadena de caracteres es igual o diferente a otra, esto también puede aplicarse a valores numéricos. En dado caso de que la condición que se le plantee sea verdadera, entonces, se entra en el cuerpo del condicional donde se ejecutan las instrucciones que dentro se planteen.

En caso de que la condición sea falsa y no se cumpla, simplemente no entra en esta estructura de control, aunque, si se desea que el programa haga algo cuando su condición es falsa se puede utilizar la condicional **else**, esta sentencia es opcional, pero si se utiliza está indicando que debe realizar ciertas instrucciones en caso de que la condicional **if** no se cumpla, además, se puede tener condicionales anidadas donde se pueda verificar entre varias posibles condiciones, por ejemplo, en caso de que la

condicional **if** no se cumpla se puede utilizar la condicional **else if** donde se indique otra condición y así repetidamente, en dado caso de que la última condicional **else if** no se cumpla se puede agregar una sentencia **else** que se ejecute.



```
Compilador CMM
Archivo Editar Compilador Ayuda
hola-mundo.cmm
1 void main() {
2     int n = 10;
3     char c = 'a';
4     string texto = "Hola";
5
6     string txt;
7     if(n < 20) {
8         txt = "Se cumple la primer condicion";
9     } else if(c == 'b') {
10        txt = "Se cumple la segunda condicion";
11    } else if(texto == "Hola") {
12        txt = "Se cumple la tercera condicion";
13    } else {
14        txt = "Se cumple la cuarta condicion";
15    }
16    print(txt);
17 }
```

Compilación finalizada.

Ilustración 14 - Ejemplo condicional if anidada

Ciclo While

La estructura de control **while** es una estructura repetitiva, esto quiere decir, que va a ejecutar las instrucciones que se le hayan indicado más de una vez, mientras se cumpla la condición que se le haya dado, si la condición no se cumple entonces no se ejecutan las instrucciones dadas, se define con la sentencia **while** y, como se mencionaba, se debe definir una condición la cual se ingresa entre paréntesis, las instrucciones se ingresan dentro de símbolos de llave **{ }**.

A continuación, se presenta un ejemplo del uso del ciclo while para imprimir números del 0 al 9.

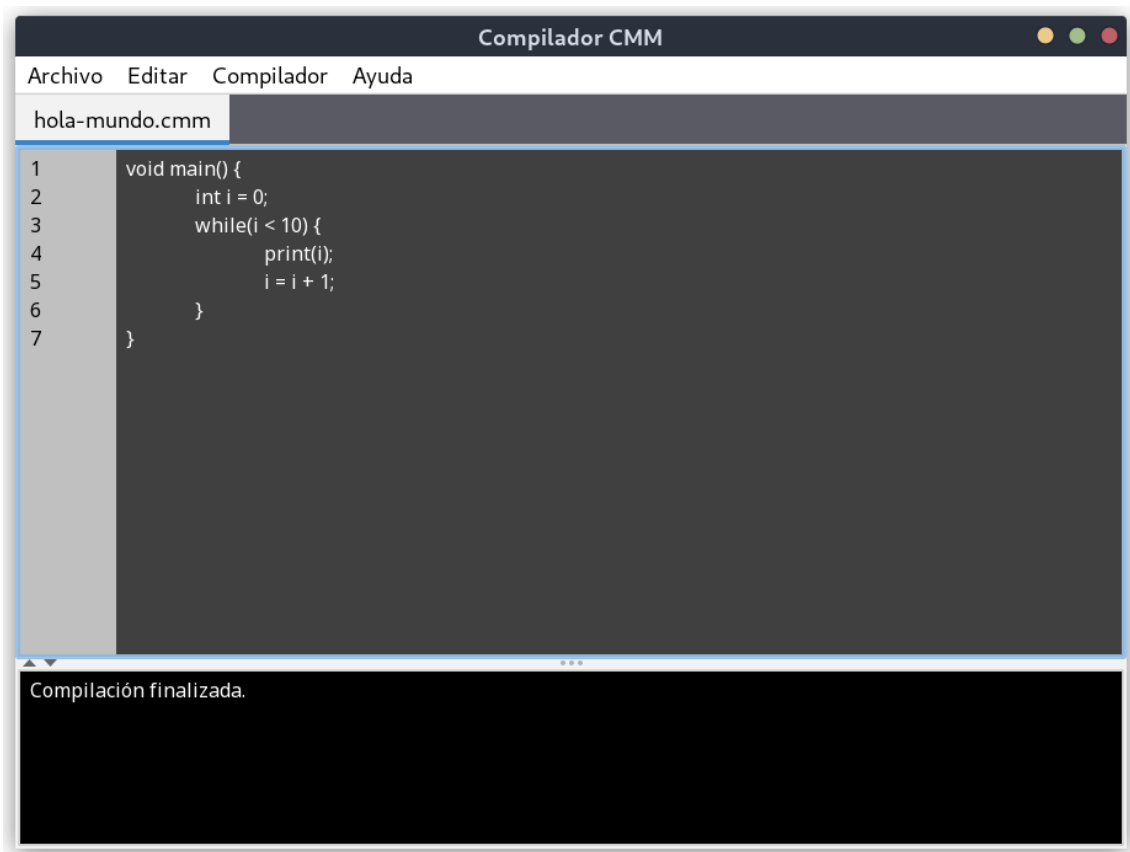


Ilustración 15 - Ejemplo ciclo while

Ciclo For

La estructura de control for es parecida a la estructura while, ambas son estructuras repetitivas que ejecutan las instrucciones dadas mientras se cumpla la condición dada, la diferencia radica en que la estructura de control for, además de la condición, debe definirse la inicialización de las variables a usar y una última instrucción a realizar después de la ejecución de las instrucciones dadas, para que se entienda mejor se muestra el siguiente ejemplo que imprime los números del 0 al 9.

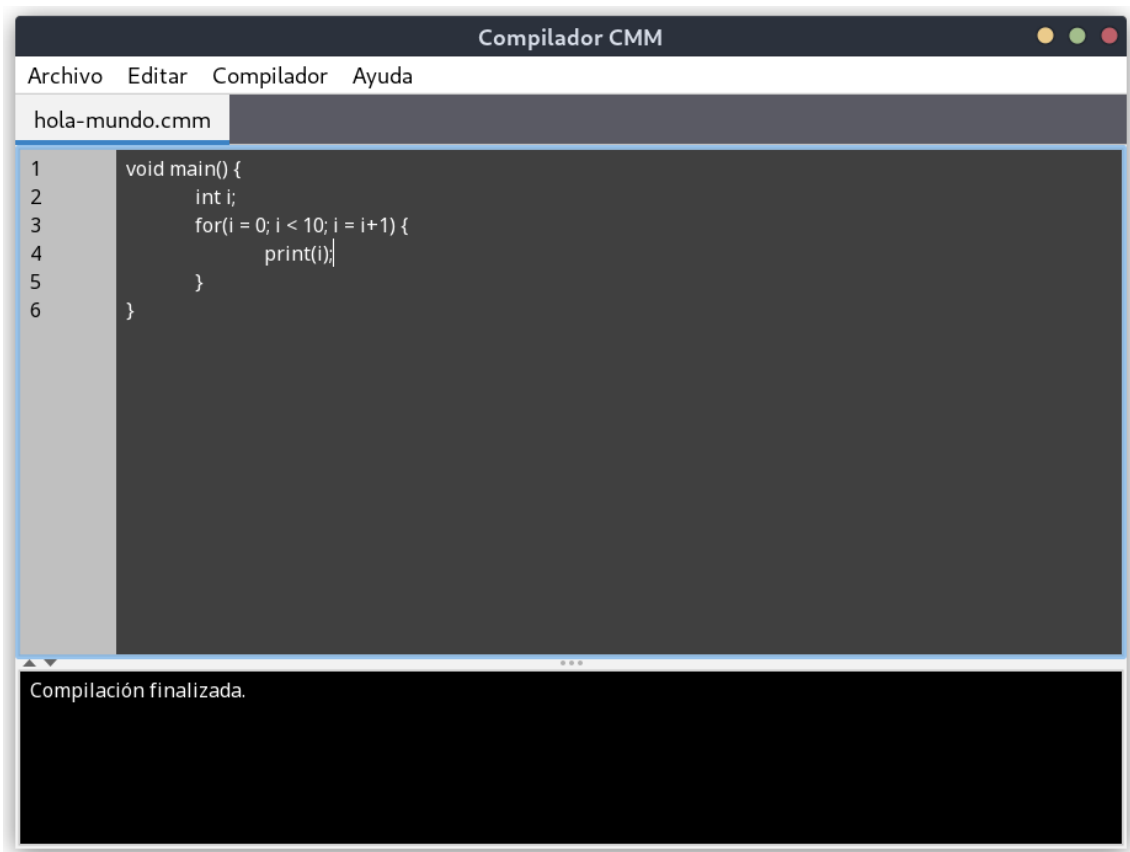


Ilustración 16 - Ejemplo ciclo for

Funciones

El lenguaje C++ es un lenguaje funcional, esto significa que se puede escribir pequeñas porciones de código, llamadas funciones; con un fin en específico y estas funciones pueden ser invocadas en cualquier punto de la ejecución, las funciones permiten a los desarrolladores ahorrar código y mantener una mejor abstracción de lo que están haciendo.

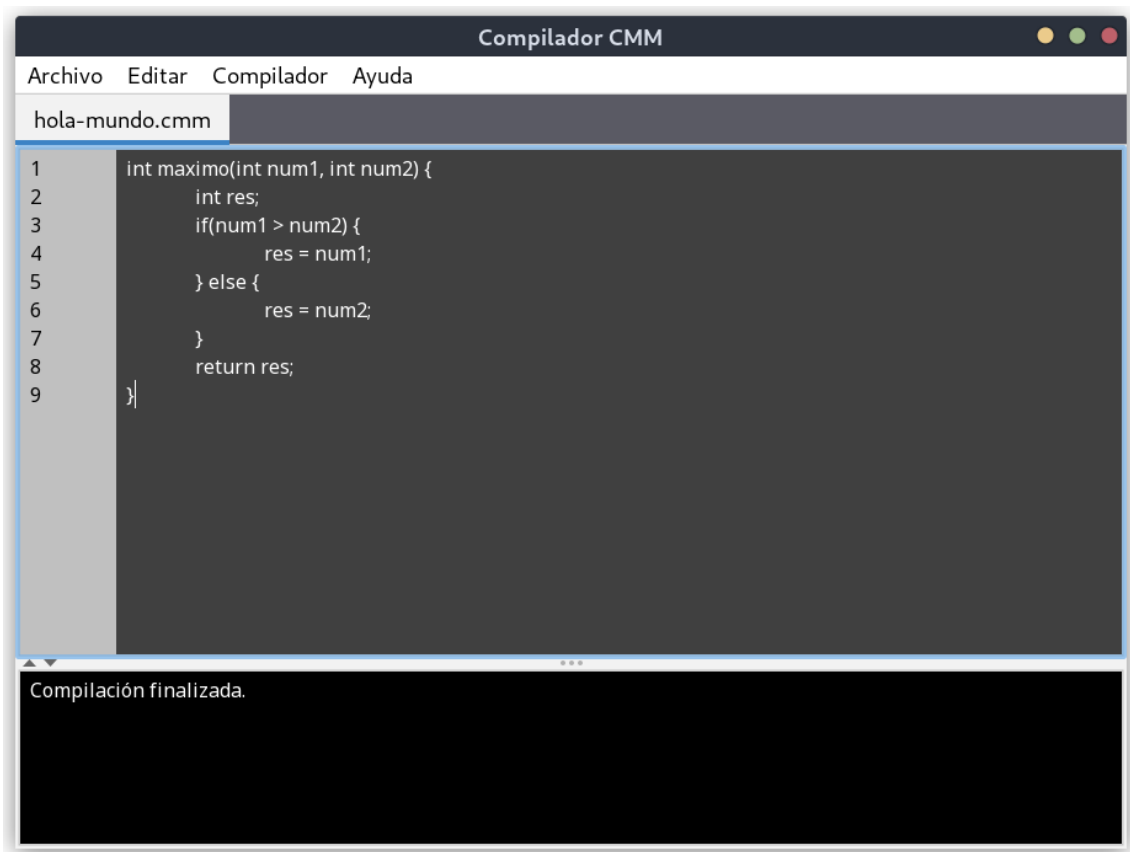


Ilustración 17 - Ejemplo de función

Para declarar una función se debe especificar primeramente su tipo de dato, esto para definir si la función hace ciertas acciones y al final devuelve un valor como resultado, es entonces que este tipo de dato que se define en un principio indica el tipo de dato del valor que va a devolver, estos tipos de datos son los anteriormente mencionados **int**, **char**, **string**, **bool**; aunque, si una función no está pensada para devolver algún valor se usa **void**, indicando que no tendrá valor de retorno.

Después se debe definir el nombre que tendrá la función, este nombre se define igual que el nombre de una variable, el primer carácter debe ser una letra minúscula, mayúscula o guion bajo, de ahí el resto de caracteres que compongan el nombre puede ser minúsculas, mayúsculas, números o guion bajo.

A continuación se debe definir los parámetros de la función, los parámetros son variables que se inicializan a la hora en que se manda a llamar a la función y existen dentro de la función que se está ejecutando, la creación de variables como parámetros es opcional, pero ya sea que se quiera agregar o no, se debe definir el área de parámetros, para definirlo se debe agregar un paréntesis que abre y uno que cierra después del nombre de la función, en caso de que no se quiera definir parámetros solo se debe dejar vacío dentro de estos paréntesis, en caso de que se desee crear parámetros se debe definir como variables no inicializadas, esto significa, se debe definir el tipo de dato de la variable seguido del nombre de la misma, si se desea definir más de un parámetro se debe agregar una coma al final del último parámetro y definir el tipo de dato y nombre del siguiente parámetro.

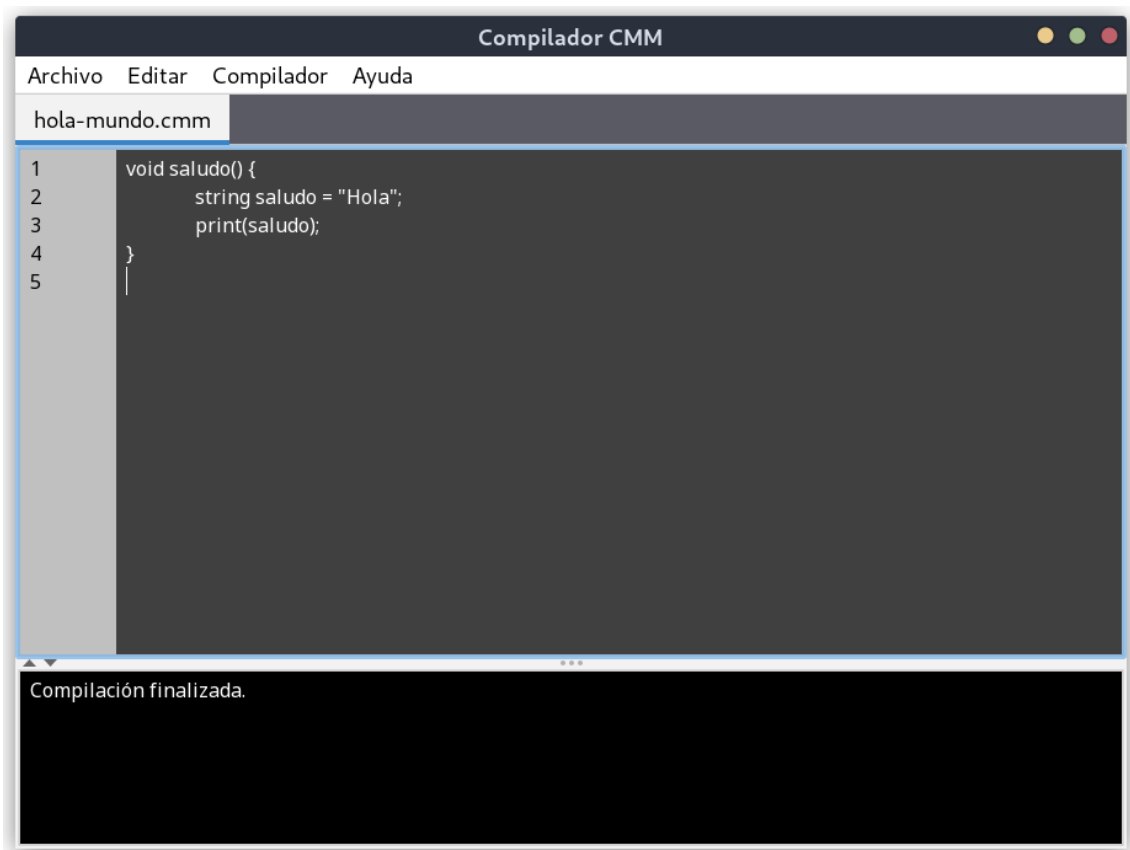


Ilustración 18 - Ejemplo función sin parametros

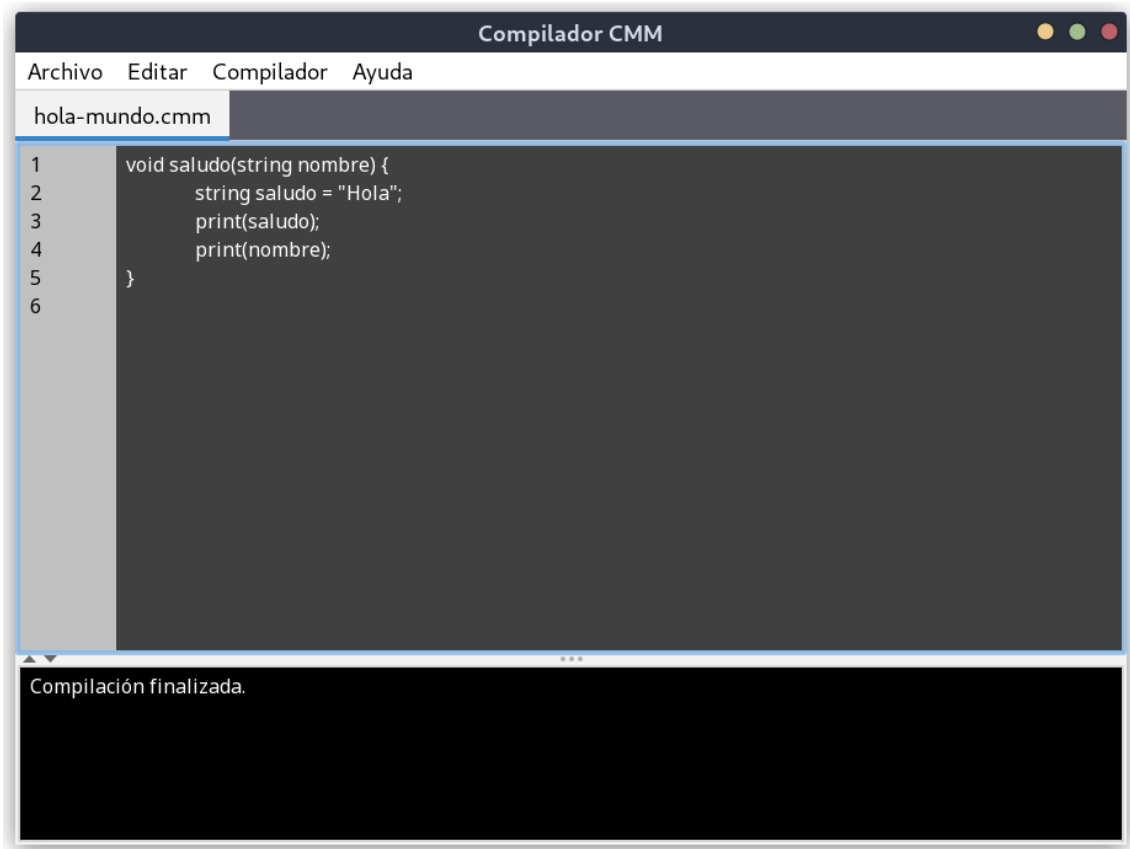
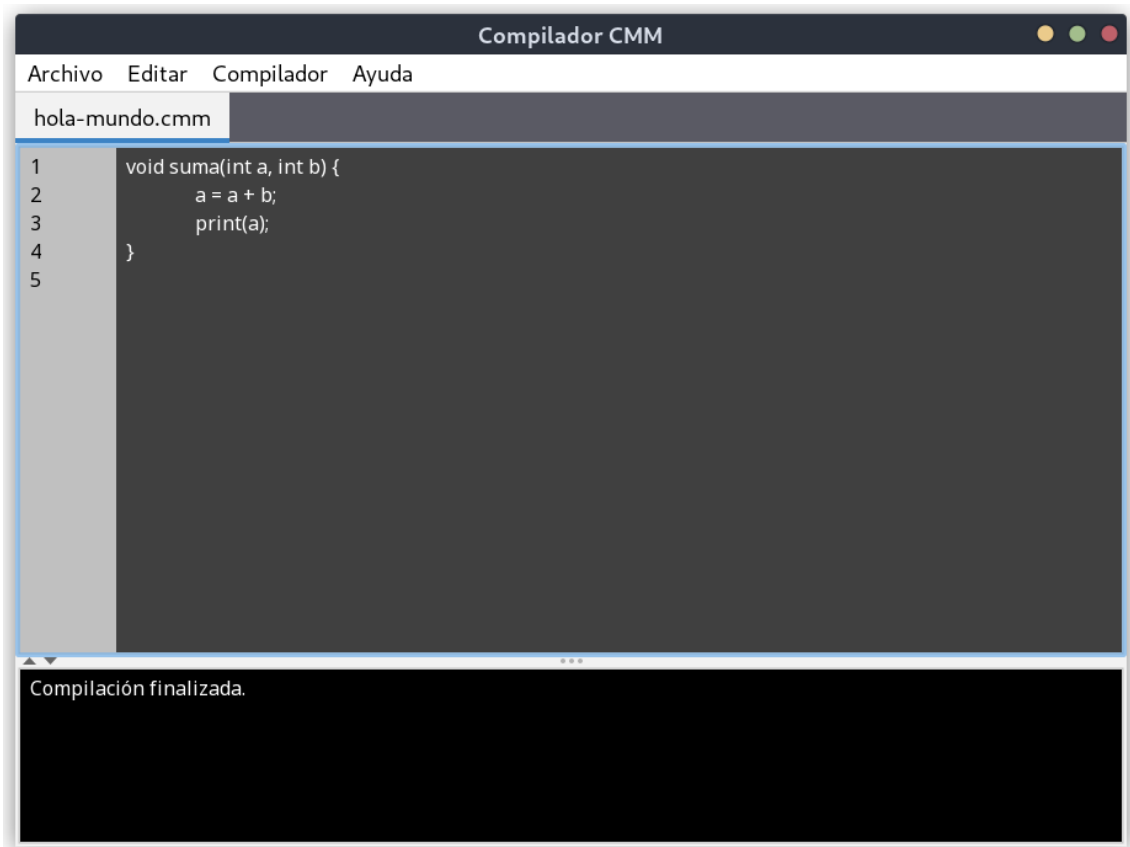


Ilustración 19 - Ejemplo función con parametros

Como siguiente paso se debe definir el cuerpo de la función, es decir, que instrucciones realizará la función, para definir el cuerpo de una función se debe agregar una llave que abre `{` y una llave que cierra `}`, dentro de estas llaves se debe escribir todas las instrucciones que se desea que realice dicha función, estas instrucciones son las ya mencionadas a lo largo de este documento, además, en caso de que la función se haya definido con un tipo de dato diferente a **`void`** se deberá de escribir al final la instrucción **`return`** indicando el final de la función y devolviendo el valor de retorno, cabe mencionar que la sentencia **`return`** solo acepta variables y el tipo de dato de estas debe ser acorde al tipo de dato de la función.



```
Compilador CMM
Archivo  Editar  Compilador  Ayuda
hola-mundo.cmm
1 void suma(int a, int b) {
2     a = a + b;
3     print(a);
4 }
5
Compilación finalizada.
```

Ilustración 20 - Ejemplo función sin return

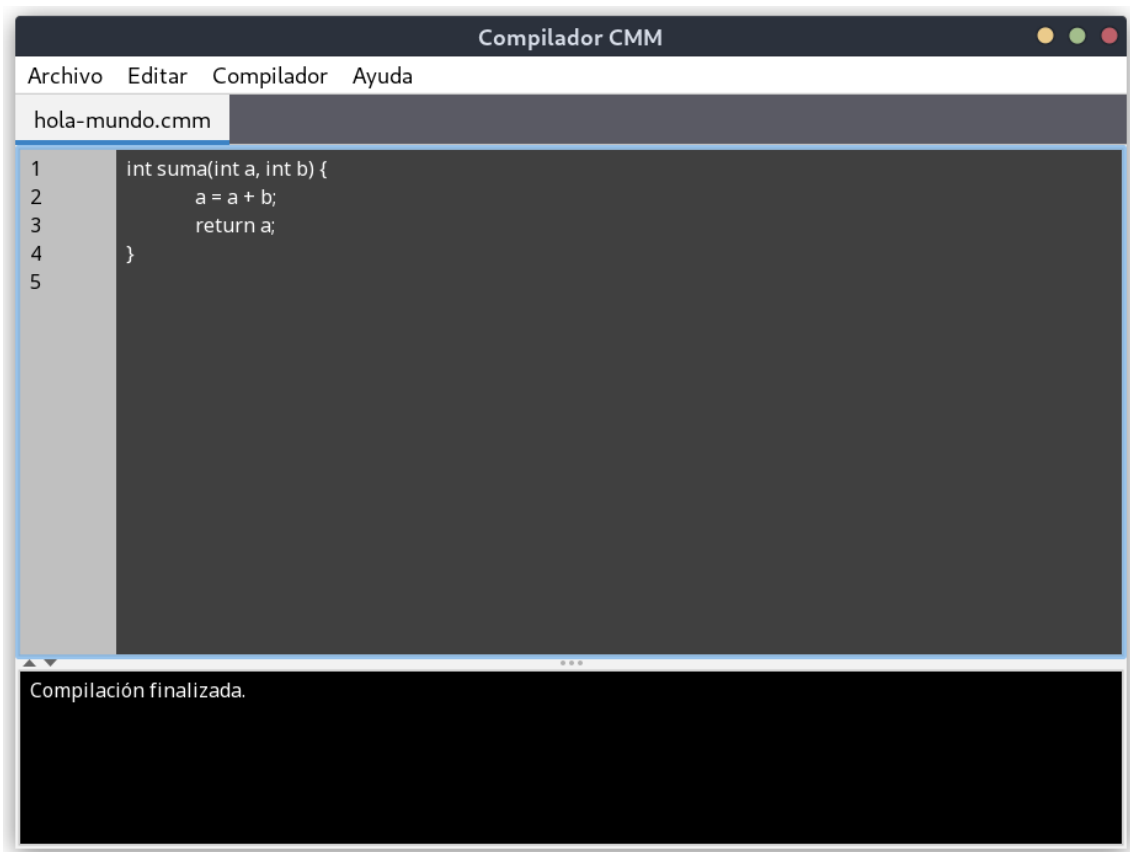


Ilustración 21 - Ejemplo función con return

Por último, para mandar a llamar a estas funciones creadas por el usuario se debe escribir el nombre de la función que se desea ejecutar seguido de paréntesis que abre y que cierra donde dentro se debe escribir los valores de los parámetros, estos deben tener la misma cantidad de valores que de parámetros y estar divididos por comas.

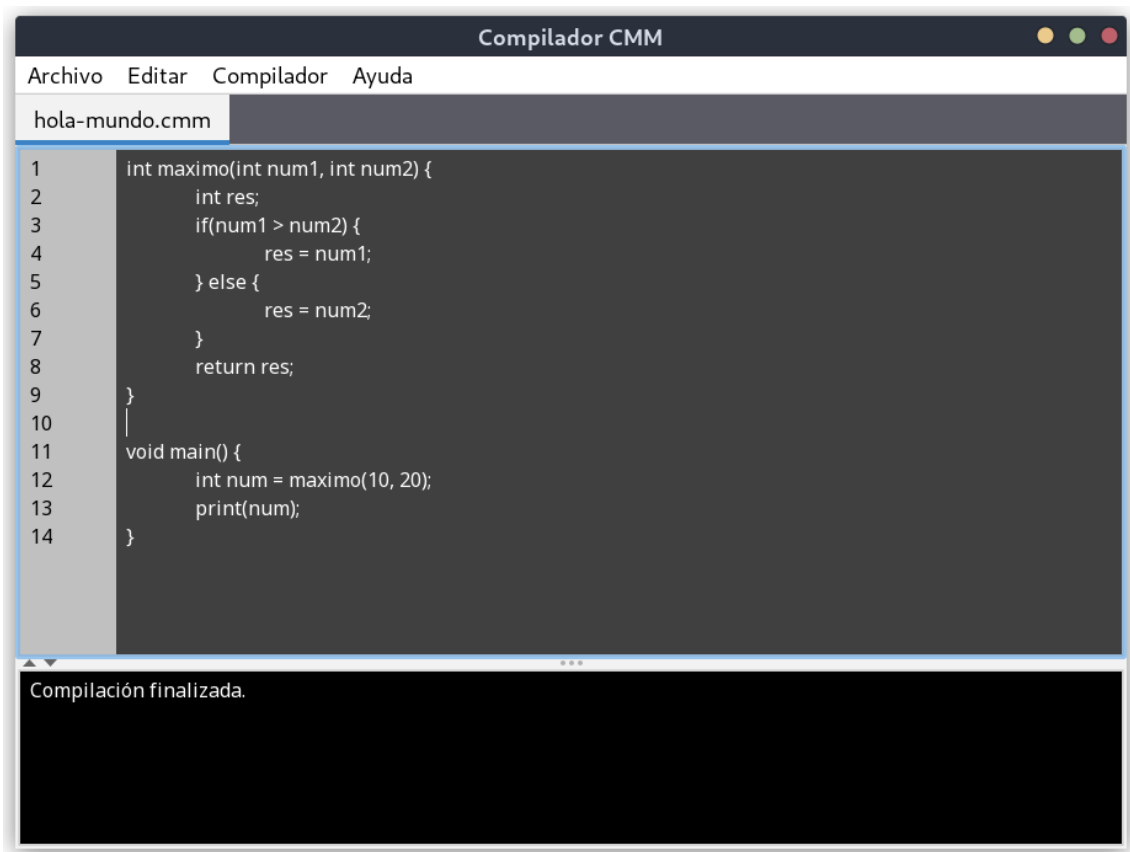


Ilustración 22 - Ejemplo completo de función

Como adicional, debe notarse que la estructura que han seguido los programas hasta ahora han sido dentro de una función llamada main, esta función es una función especial ya que define el inicio de la ejecución del programa, de forma que, cuando se compile el programa, si no se define una función main, el programa no tendrá punto de ejecución, por lo tanto, no hará nada; de igual forma mencionar que las instrucciones print e input antes mencionadas son en realidad funciones, pero estas son funciones ya definidas dentro del lenguaje.

Comentarios

Si un usuario desea agregar comentarios a su código para dejar una explicación sobre su programa lo puede hacer. Los comentarios se inician con `/*` y terminan con `*/`, y todo lo que esté dentro de estos se toma como comentario, además estos se pueden poner en cualquier punto del programa.

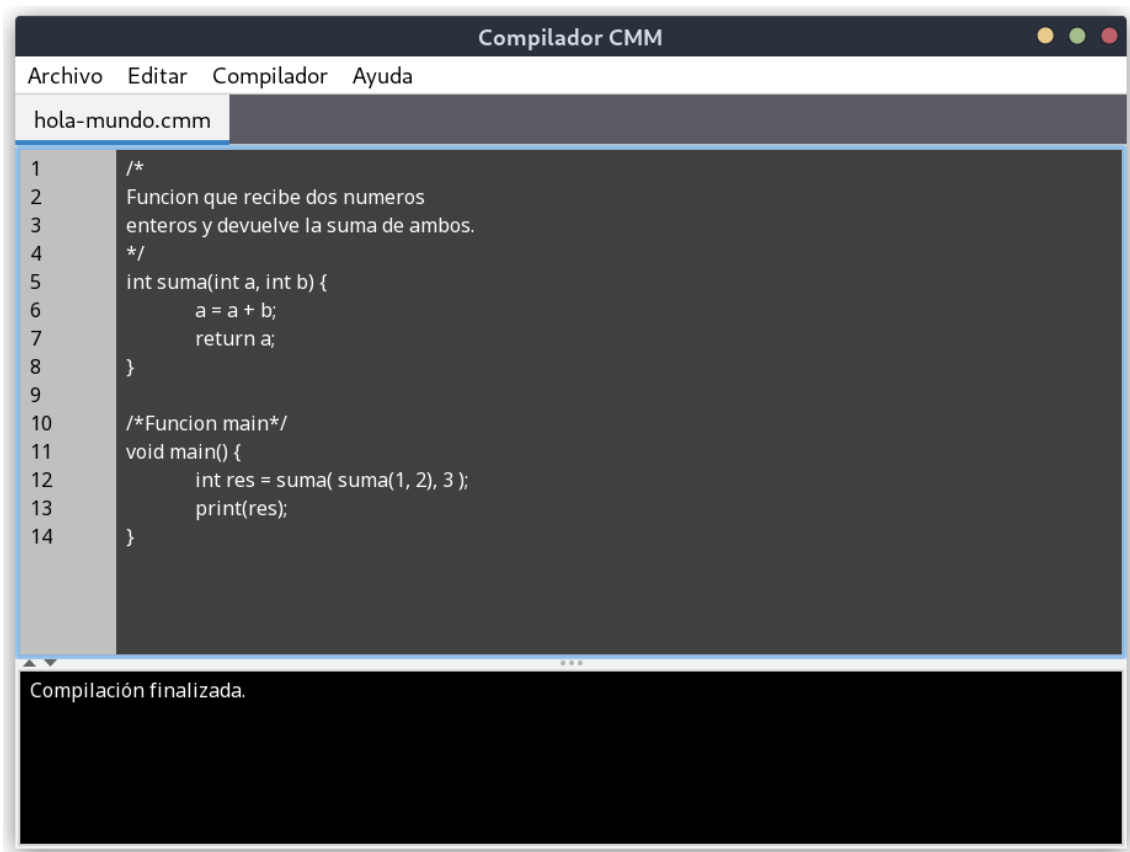


Ilustración 23 - Ejemplo de comentarios

Compilación

Una vez se tiene el código escrito se puede mandar a compilar con la opción de **Compilador > Compilar** en el menú superior, si el código es correcto y no presenta errores, al compilar debe generarse unos archivos en el mismo directorio donde se guardó el archivo con extensión .cmm, estos archivos mantienen el mismo nombre que el código fuente, pero son diferentes entre ellos.

Cada uno pertenece a una fase del proceso de compilación, el primer archivo que se genera es con extensión .asm y almacena el código escrito transpilado a código ensamblador; el segundo archivo es con extensión .o y es el que genera la compilación del archivo ensamblador, este archivo también se conoce como archivo de objetos; por último, se genera un archivo sin extensión, este archivo es un ejecutable en lenguaje maquina generado tras compilar el archivo de objetos.

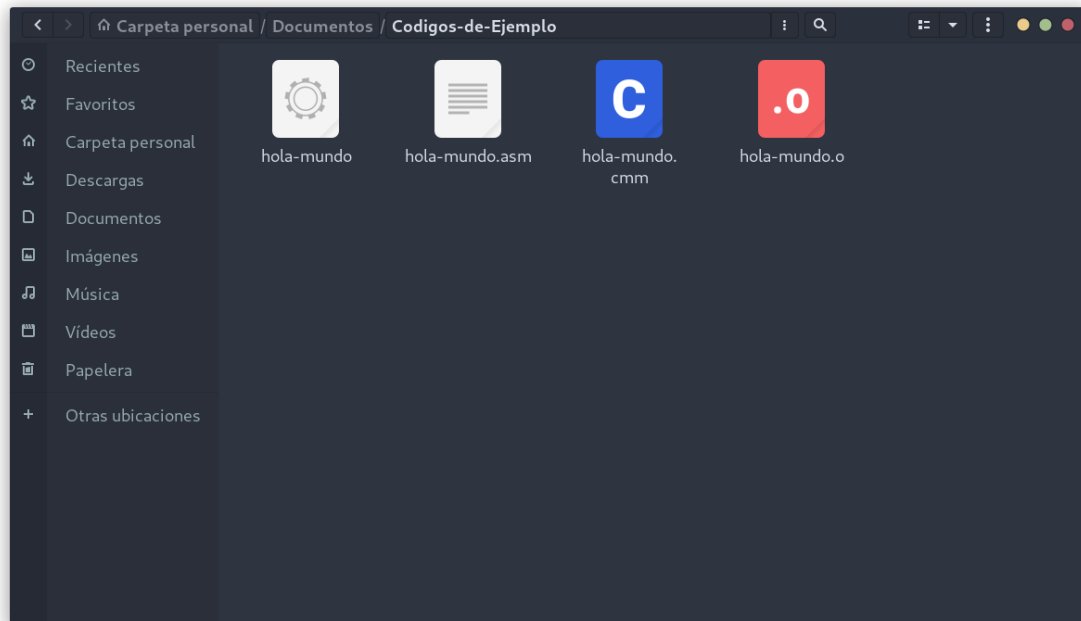


Ilustración 24 - Ejemplo de archivos de compilación