2025

Aprende lógica de programación de una vez

Enfoque práctico con ejercicios en C++

José Juan Hernández García

Aprende lógica de programación de una vez

Enfoque práctico con ejercicios en C++

Este libro está a la venta en
http://leanpub.com/aprendecplusplus

Esta versión se publicó el 22-10-2025

Este es un libro de Leanpub. Leanpub anima a los autores y publicadoras con el proceso de publicación. Lean Publishing es el acto de publicar un libro en progreso usando herramientas sencillas y muchas iteraciones para obtener *feedback* del lector hasta conseguir tener el libro adecuado.

© 2025, José Juan Hernández García

Tabla de contenido

Índice de programas

Índice de figuras

Introducción a C++

Características

- **C++** es un lenguaje de programación de alto nivel, compilado, de propósito general ampliamente utilizado y de alto rendimiento.
- C++ fue diseñado en 1979 por Bjarne Stroustrup y para su creación tomó como base el popular lenguaje de programación C.
  Por tanto, es un derivado del mítico lenguaje C, el cual fue creado en la década de los 70 por Dennis Ritchie para la programación del sistema operativo UNIX, y que surgió como un lenguaje orientado a la programación de sistemas y aplicaciones siendo su principal característica la eficiencia del código que produce.
- La expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C para incluir características de la **programación orientada a objetos** (POO).
- Es portable y con un gran número de compiladores en diferentes plataformas y sistemas operativos.
- Es un lenguaje muy didáctico, que permite aprender muchos otros lenguajes con gran facilidad.
- C++ admite **plantillas**, que permiten escribir código que puede funcionar con múltiples tipos de datos sin duplicar el código. Esto facilita la creación de funciones y bibliotecas, que facilita la reutilización de código y la abstracción.
- C++ tiene una comunidad de desarrolladores grande y activa que proporciona soporte, herramientas y bibliotecas de terceros. Además, hay una gran cantidad de recursos disponibles, como tutoriales, cursos y libros.
- Para escribir un programa en C++ necesitamos un editor de texto y un compilador para la plataforma y el sistema operativo que estemos utilizando. Normalmente estas herramientas se unifican en los Entornos Integrados de Desarrollo[1] (IDE). Ejemplos de IDE que podemos usar: Visual Studio Code[2], Dev-C++[3], Visual C++, ZinjaI,

[1] Aplicación que ofrece en un mismo programa distintas funcionalidades (editor de texto, compilador, …)
[2] https://code.visualstudio.com/
[3] https://dev-cpp.com/

Eclipse[4], Code::Blocks[5], …

- Durante los últimos años se han estandarizado[6] distintas versiones donde se han ido añadiendo nuevas funcionalidades al lenguaje. Estas versiones se nombran con el año en que son publicadas, de esta manera tenemos: C++98, C++03, C++11, C++14, C++17, C++20 (versión estable actual), C++23 (versión en desarrollo).

¿Qué puede hacerse con C++?

- **Desarrollo de software de sistemas:** C++ es muy popular para desarrollar sistemas operativos, controladores de dispositivos y otros programas de bajo nivel debido a su rendimiento y control sobre el hardware.
- **Desarrollo de videojuegos:** Populares motores de videojuegos como Unreal Engine[7] y Unity[8] (con su soporte nativo) utilizan C++ debido a su alto rendimiento y capacidad para controlar eficientemente los recursos del sistema.
- **Aplicaciones de escritorio:** en plataformas Windows, macOS y GNU/Linux.
- **Desarrollo de aplicaciones móviles:** para Android e iOS a través de *frameworks* como Qt o usando el soporte nativo de C++ en Android Studio[9].
- **Computación de alto rendimiento:** La velocidad y rendimiento de C++ lo hacen ideal para aplicaciones científicas, de análisis de datos y simulaciones que requieren una gran cantidad de cálculos y procesamiento de datos.
- **Desarrollo de bibliotecas y *frameworks*:** Muchas bibliotecas y *frameworks* ampliamente utilizados, como Boost, Qt o TensorFlow[10], están escritos en C++.
- **Programación embebida y controladores de hardware:** ya que C++ ofrece un buen equilibrio entre control de bajo nivel y abstracciones de alto nivel.

---

[4] https://eclipseide.org/
[5] https://www.codeblocks.org/
[6] https://isocpp.org/
[7] https://www.unrealengine.com/
[8] https://unity.com/es
[9] https://developer.android.com/studio
[10] https://www.tensorflow.org/

- **Desarrollo web:** Aunque no es el uso más común, C++ también se puede utilizar para escribir servidores web y aplicaciones *backend* de alto rendimiento.

Instalación del IDE Zinjal

Zinjal[11] es un IDE (del inglés: *Integrated Development Enviroment* = Entorno Integrado de Desarrollo) libre y gratuito para programar en C/C++. Pensado originalmente para ser utilizado por estudiantes de programación durante el aprendizaje, presenta una interfaz inicial muy sencilla, pero sin dejar de incluir funcionalidades avanzadas. Puedes ver las características[12] de Zinjal en su página web y en la sección Descargas[13] puedes encontrar el programa para las distintas plataformas con las instrucciones de instalación.

Estructura de un programa en C++

Veamos nuestro primer programa para estudiar su estructura:

```
// Incluir la biblioteca estándar para entrada y salida
#include <iostream>
// Usar el espacio de nombres estándar
using namespace std;

/* funcion main() Es la función principal
  donde empieza la ejecución del programa  */
int main(int argc, char *argv[]) {
    cout << "Hola mundo!!!";  // Imprime Hola mundo!!!

    // Indica que el programa terminó correctamente

    return 0;
}
```

Código 1: Programa "Hola mundo!"

[11] https://zinjai.sourceforge.net/
[12] https://zinjai.sourceforge.net/index.php?page=features.php

Elementos que forman parte de la estructura de un programa C++:

- `#include <iostream>`: indica que utilizaremos la librería `iostream`. En esta librería están definidas las funciones de entrada/salida, como mostrar información en la consola, por ejemplo, `cout`.
- `using namespace std;`: indicamos que usaremos el espacio de nombres estándar (`std`). Como podemos tener diferentes elementos en el lenguaje que se llamen igual, se utilizan espacios de nombres para agruparlos. Las funciones de entrada / salida como `cout` o `cin` están definidas en el espacio de nombres `std`, por lo tanto, si indicamos que vamos a usarlos no será necesario nombrarlos cuando escribamos las instrucciones. Si no indicamos que vamos a usar el espacio de nombres `std` la instrucción a escribir en pantalla quedaría de la siguiente forma:

```
std::cout << "Hola mundo!!!";
```

- `int main()`: es la **función principal del programa**. Todo programa la tiene, es el punto de entrada donde comienza la ejecución. Al ejecutar el programa son las instrucciones de esta función por las que se empieza. Esta función devuelve un valor entero (`int`) al sistema operativo. Si el programa va a tener parámetros en la línea de comandos, encontraremos esta función definida de esta manera:

```
int main()
```

  - `argc`: argumentos enviados al programa desde el entorno donde se ejecuta. Es un valor no negativo.
  - `argv`: argumentos pasados al programa desde el entorno de ejecución.

- `cout << "Hola mundo!!!";`: instrucción que imprime en pantalla. El operador `<<` es el operador de inserción que “envía” el texto `"Hola mundo!!!"` al flujo de salida.
- `return 0;`: como dijimos la función `main()` devuelve un valor entero (`0` si todo sale bien, distinto de `0` si se produce algún error).
- Los bloques de instrucciones se guardan entre corchetes `{` y `}`.
- Todas las instrucciones deben acabar en `;`.
- Podemos poner comentarios de una línea (utilizando los caracteres `//`) o comentarios de varias líneas (con los caracteres `/*` y `*/`). Todos los comentarios son ignorados por el compilador.
- C++ es ***case-sensitive*** (distingue entre mayúsculas y minúsculas).

Por ejemplo, `variable`, `Variable` y `VARIABLE` serían tres identificadores distintos. Hay ciertas convenciones a seguir: el nombre de las variables se suele poner siempre en minúsculas, mientras que el nombre de las constantes se suele poner en mayúsculas.

Palabras clave

Las palabras clave no pueden redefinirse o sobrecargarse.

A – C	D – P	R – Z
`alignas` (desde C++11)	`decltype` (desde C++11)	`register`
`alignof` (desde C++11)	`default`	`reinterpret_cast`
`and`	`delete`	`requires` (desde C++20)
`and_eq`	`do`	`return`
`asm`	`double`	`short`
`auto`	`dynamic_cast`	`signed`
`bitand`	`else`	`sizeof`
`bitor`	`enum`	`static`
`bool`	`explicit`	`static_assert` (desde C++11)
`break`	`export`	`static_cast`
`case`	`extern`	`struct`
`catch`	`false`	`switch`
`char`	`float`	`template`
`char8_t` (desde C++20)	`for`	`this`
`char16_t` (desde C++11)	`friend`	`thread_local` (desde C++11)
`char32_t` (desde C++11)	`goto`	`throw`
`class`	`if`	`true`
`compl`	`inline`	`try`
`concept` (desde C++20)	`int`	`typedef`
`const`	`long`	`typeid`
`consteval` (desde C++20)	`mutable`	`typename`
`constexpr` (desde C++11)	`namespace`	`union`
`constinit` (desde C++20)	`new`	`unsigned`
`const_cast`	`noexcept` (desde C++11)	`using`
`continue`	`not`	`virtual`
`co_await` (desde C++20)	`not_eq`	`void`
`co_return` (desde C++20)	`nullptr` (desde C++11)	`volatile`
`co_yield` (desde C++20)	`operator`	`wchar_t`
	`or`	`while`
	`or_eq`	`xor`
	`private`	`xor_eq`
	`protected`	
	`public`	

Tabla 1: Palabras clave

Normas de nombrado

Al igual que en otros lenguajes, C++ tiene reglas para nombrar variables y funciones:

- **No pueden comenzar con un número**: los nombres de variables deben comenzar con una letra o un guion bajo (_), pero nunca con un número.

```
int 1edad = 25;  // Incorrecto
int edad1 = 25;  // Correcto
```

- **No pueden contener espacios**: los nombres deben ser una sola palabra sin espacios.

```
int mi edad = 25;  // Incorrecto
int miEdad = 25;  // Correcto
```

- **No pueden incluir símbolos especiales** como `@`, `#`, `%`, `!`, etc., pero pueden usar guiones bajos (_).

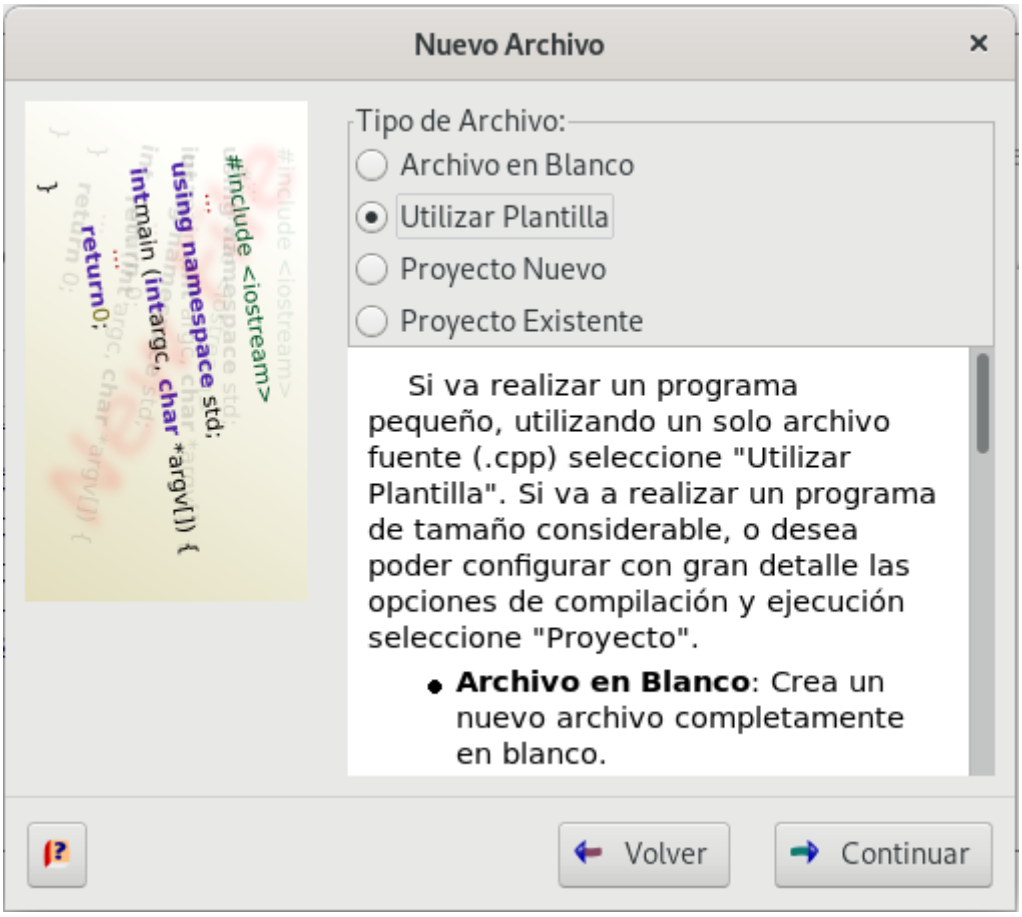
```
int edad$ = 25;  // Incorrecto
int _edad = 25;  // Correcto
```

- **Convenciones de estilo**: aunque C++ no lo exige, es común usar la convención de *camelCase* (como en `miVariable`) o *snake_case* (como en `mi_variable`) para nombres de variables, según el estilo del equipo o proyecto.

Compilación y ejecución de programas

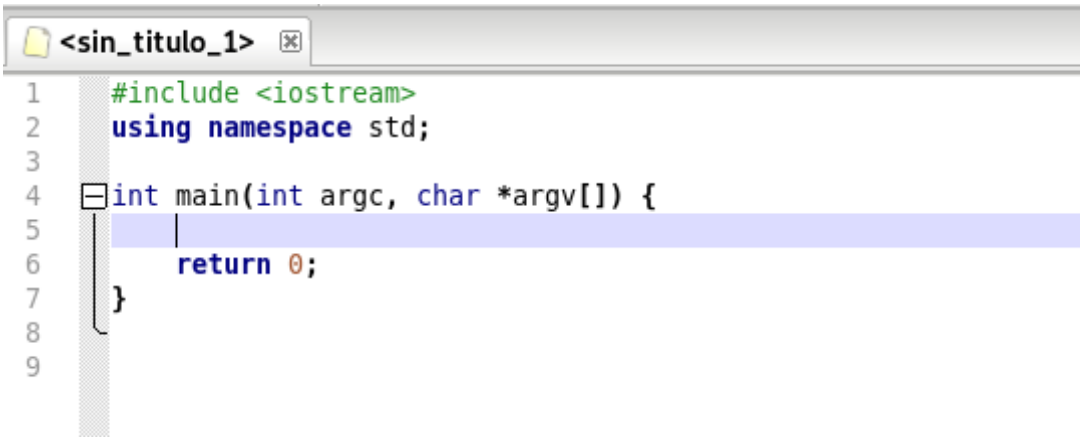
Vamos a compilar y ejecutar nuestro primer programa en C++.

Para ello vamos a seguir los siguientes pasos:

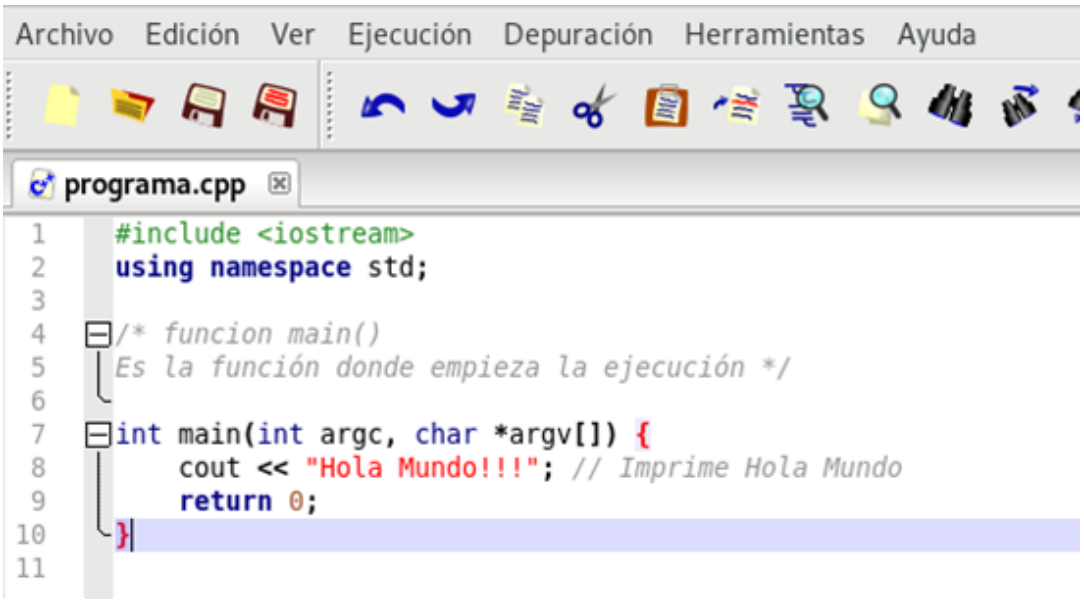


Paso 1. Crear un nuevo programa. Para ello seleccione la opción `Nuevo...` del menú `Archivo` (o pulsando CTRL + N).

Se desplegará inmediatamente el Asistente para Nuevo Archivo. Allí seleccione la opción `Utilizar Plantilla` y haga clic en el botón `Continuar`: elegimos la versión de C++ que vamos a usar, por ejemplo, `Programa C++14 en Blanco` y obtendremos un fichero con la estructura de nuestro programa:



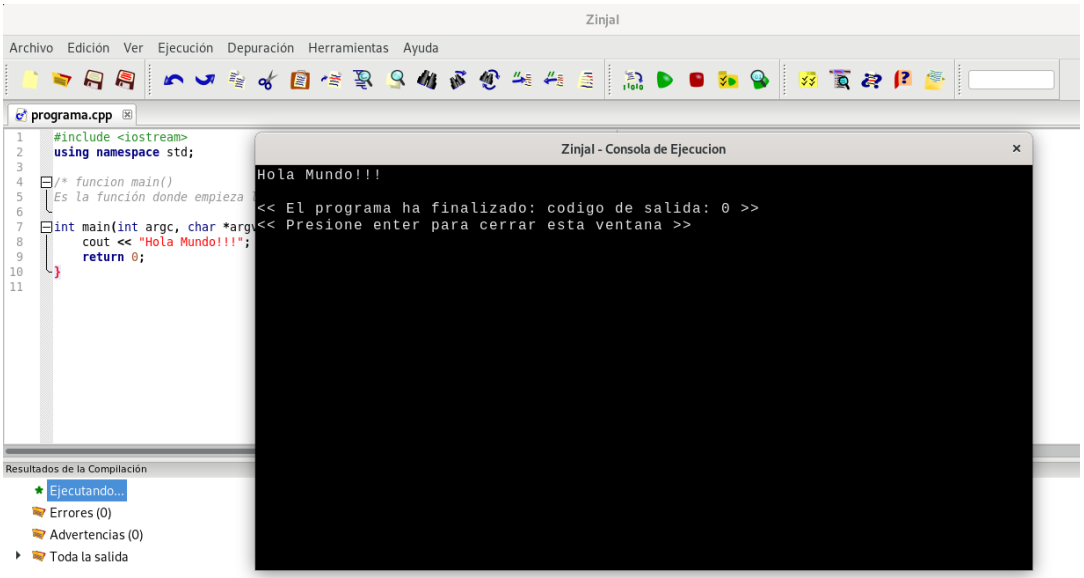
Paso 2. Escribimos el programa. Recuerda que el programa se puede guardar (Opción `Archivo -> Guardar` o CTRL + S) en un fichero en nuestro sistema de archivos con la extensión `cpp`.



Paso 3. Para intentar ejecutar el programa presione F9, o seleccione la opción `Ejecutar` del menú `Ejecución`.

Esta acción guarda el archivo (si aún no tiene nombre lo hará en un directorio temporal) y lo compila. Si la compilación es exitosa lo ejecuta. Aparecerá en la parte inferior de la ventana principal el panel de **Resultados de la Compilación**, en el cual se muestra el estado de la compilación y los resultados de esta.

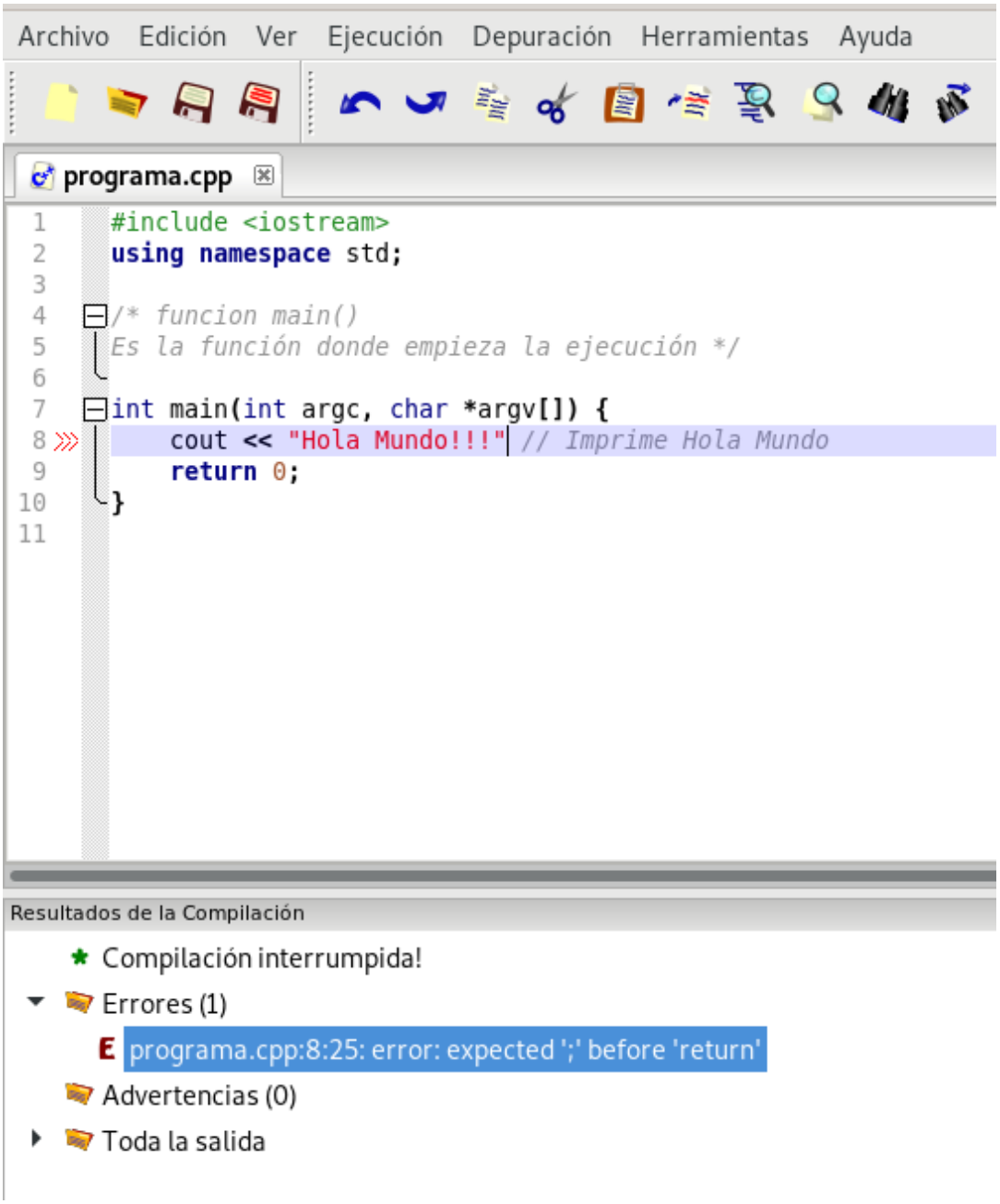
Si la compilación no tiene errores aparece una ventana de terminal donde vemos la ejecución. Luego de finalizar la ejecución, Zinjal informará del código de retorno del programa (el `0` de la línea final `return 0;`, el cual sirve para saber si se ejecutó correctamente) y esperará a que presione Enter antes de cerrar la ventana, para permitirle observar los resultados.



Pero, si hemos tenido algún error pasamos al siguiente paso.

Paso 4. Si hemos cometido algún error, por ejemplo, se nos ha olvidado un `;` al final de una instrucción, en el proceso de compilación obtendremos errores:

Al hacer doble clic sobre el error en el panel de compilación el cursor se desplaza hacia la línea que lo provocó.



Tenemos que corregir todos los errores sintácticos para que el proceso de compilación se pueda realizar.

Atajos de teclado

Algunas combinaciones de teclas para aprovechar mejor las facilidades de edición en ZinjaI:

- `F9`: Realiza todos los pasos necesarios para probar un programa (guardar, compilar y ejecutar).
  Si se presiona `Shift+F9`, se evita el último paso; es decir, sólo se compila.
  Esto sirve para saber si el código es sintácticamente correcto.
- `Ctrl+<`: Si la compilación arroja errores o advertencias, con esta combinación se pueden recorrer los mismos.
- `Ctrl+H`: Busca la librería que contiene la declaración de una determinada clase, función, variable o macro e inserta al principio del archivo el `#include` correspondiente para poder utilizarla.
- `Ctrl+L`: Duplica la línea actual o las líneas seleccionadas. Es útil en muchos casos en que el código incluye líneas casi idénticas, equivale a copiar y pegar esas líneas.
  `Ctrl+Shift+L`: Elimina la línea actual o las líneas seleccionadas.
- `Ctrl+T, Ctrl+Shift+T`: Desplazan la línea actual o las líneas seleccionadas una posición más arriba en el código. Sirven para mover fragmentos de código líneas arriba o abajo.

Diferencias entre C, C++ y C#

En el mundo de la programación, tres lenguajes han sido pilares fundamentales en el desarrollo de software: **C**, **C++** y **C#**. Cada uno de ellos han jugado un papel crucial en la evolución de la industria, y aunque comparten ciertas raíces, sus enfoques y aplicaciones varían significativamente.

El lenguaje **C** es conocido por ser un lenguaje de bajo nivel con una gran influencia en sistemas operativos y software de alto rendimiento. **C** amplía las capacidades de C con programación orientada a objetos y otras características avanzadas. **C#** se diseñó como un lenguaje más moderno y versátil, con una sintaxis clara y potentes herramientas de desarrollo.

Exploremos las **diferencias clave** entre estos lenguajes, sus principales características y en qué escenarios es más conveniente utilizar cada uno.

Características de C

El lenguaje **C** es uno de los pilares fundamentales en el mundo de la programación. Es un lenguaje de **bajo nivel**, con una sintaxis eficiente y directa que permite un control absoluto sobre el hardware. Su diseño centrado en la eficiencia lo convierte en ideal para **sistemas operativos**, controladores de dispositivos y software embebido.

Una de las principales características de C es su **gestión manual de memoria** a través de funciones como `malloc()` y `free()`, lo que le otorga un gran rendimiento, pero también una mayor responsabilidad al programador. Esto significa que cualquier error en la administración de memoria puede derivar en **fugas de memoria o accesos indebidos**, lo que lo hace propenso a errores en manos inexpertas.

C es un lenguaje estructurado, lo que implica que su diseño fomenta una organización lógica del código en funciones y módulos, permitiendo una mejor legibilidad y mantenibilidad. Además, gracias a su compatibilidad con múltiples arquitecturas, es uno de los lenguajes más utilizados en sistemas críticos y de alto rendimiento.

Características de C++

C++ nace como una extensión de C con el objetivo de incluir **programación orientada a objetos (POO)**, sin perder la eficiencia del lenguaje original. Esto significa que hereda muchas de las características de C, como la gestión manual de memoria y la posibilidad de programar en bajo nivel, pero agrega potentes mejoras que facilitan la modularidad y reutilización del código.

Una de sus mayores fortalezas es la **POO**, la cual permite organizar el código en **clases y objetos**, promoviendo una mayor flexibilidad y escalabilidad. Gracias a esto, los proyectos grandes pueden manejarse de manera más eficiente, reduciendo la complejidad y facilitando el mantenimiento del software.

Otra característica clave es su compatibilidad con la **programación genérica**, que se logra a través de **plantillas (*templates*)**. Esto permite escribir código flexible y reutilizable sin sacrificar rendimiento.

Además, C++ introduce el concepto de **excepciones**, una forma más estructurada de manejar errores en comparación con C.

C++ también ofrece un **mayor nivel de abstracción**, sin perder la posibilidad de optimización manual. Esto lo hace ideal para una amplia gama de aplicaciones, desde el desarrollo de videojuegos hasta sistemas financieros y de simulación científica.

Características de C#

C# es un lenguaje moderno y **de alto nivel**, diseñado para la productividad y facilidad de uso. Su sintaxis es más amigable y su enfoque está totalmente **orientado a objetos**, lo que lo convierte en una excelente opción para el desarrollo de aplicaciones de gran escala, incluyendo software empresarial, videojuegos y servicios en la nube.

A diferencia de C y C++, C# gestiona la memoria de manera automática mediante un **recolector de basura (*garbage collector*)**, lo que reduce significativamente los errores comunes relacionados con la gestión de memoria. Esta automatización permite que los desarrolladores se enfoquen más en la lógica del negocio en lugar de en los detalles técnicos de la administración de recursos.

Otra característica de C# es su fuerte **tipado y seguridad en tiempo de ejecución**, lo que minimiza errores y mejora la estabilidad del software. Además, su integración con potentes entornos de desarrollo facilita la creación de aplicaciones en diversos dominios, incluyendo el desarrollo web, aplicaciones móviles y servicios *cloud*.

C# también destaca por su enfoque en la **programación asíncrona**, lo que permite crear aplicaciones altamente responsivas y eficientes en el manejo de procesos concurrentes. Gracias a esto, es una elección ideal para el desarrollo de aplicaciones modernas que requieren un rendimiento óptimo en entornos distribuidos.

Paradigmas de programación soportados

Cuando analizamos las diferencias entre **C, C++ y C#**, uno de los aspectos más relevantes es el paradigma de programación que cada uno de estos lenguajes implementa. Si bien los tres tienen una base común, su evolución ha permitido que cada uno adopte enfoques distintos en el desarrollo de software. A continuación, exploramos los paradigmas de programación que caracterizan a cada uno.

C: Programación estructurada

El lenguaje **C** se caracteriza por ser un **lenguaje de programación estructurada**, lo que significa que su enfoque está basado en la división del código en **funciones** bien definidas. Este paradigma fomenta la creación de programas **modulares**, donde cada parte del software está compuesta por funciones específicas que pueden ser reutilizadas en diferentes partes del sistema.

Principales características de la programación estructurada en C:

- Uso de estructuras de control como `if-else`, `switch-case` y bucles para controlar el flujo del programa.
- Eliminación del uso de saltos (`goto`), promoviendo un código **más legible y mantenible**.
- Implementación de funciones para dividir las tareas en **bloques lógicos independientes**.

A pesar de ser un lenguaje de bajo nivel, C permite escribir programas altamente eficientes y optimizados, lo que lo hace ideal para el desarrollo de sistemas embebidos, software de sistemas y aplicaciones de alto rendimiento.

C++: POO y genérica

La evolución de C dio lugar a **C++**, un lenguaje que amplía sus capacidades con la introducción de la **programación orientada a objetos (POO)** y la **programación genérica**. Paradigmas que permiten desarrollar software más estructurado, reutilizable y escalable.

La **programación orientada a objetos** introduce conceptos como:

- **Clases y objetos**, que permiten encapsular datos y comportamientos en entidades autónomas.
- **Herencia**, que facilita la creación de jerarquías de clases para la reutilización de código.
- **Polimorfismo**, que permite a un mismo método o función comportarse de manera distinta según el contexto.

Por otro lado, la **programación genérica** permite escribir código que puede trabajar con distintos tipos de datos sin necesidad de reescribir funciones o estructuras. Esto se logra mediante el uso de **plantillas (*templates*)**, una característica esencial para el desarrollo de bibliotecas eficientes y reutilizables.

Gracias a esta combinación de paradigmas, C++ es ampliamente utilizado en aplicaciones de alto rendimiento, videojuegos, desarrollo de software de sistemas y simulaciones.

C#: Programación Orientada a Objetos y Eventos

C# surge con un enfoque moderno de la **programación orientada a objetos**, pero también adopta el paradigma de **programación orientada a eventos**. Así, además de las características tradicionales de la **POO**, como **clases, herencia y polimorfismo**, facilita el desarrollo de aplicaciones interactivas y reactivas.

Algunos aspectos clave de la **programación orientada a eventos** son:

- Uso de **delegados y eventos** para responder a acciones del usuario o cambios en el sistema.
- Implementación de modelos de **suscripción-publicación**, permitiendo una comunicación eficiente entre distintos componentes del software.
- Integración con **interfaces de usuario** gráficas, como en el desarrollo de aplicaciones de escritorio y móviles.

Esta combinación de paradigmas convierte a C# en una opción ideal para el desarrollo de aplicaciones empresariales, soluciones de software escalables y entornos con fuerte orientación a la interacción del usuario.

Cuando utilizas un paradigma de programación, no estas atado a usar sólo este, puedes hacer una combinación de diferentes paradigmas de ser necesario.

Tipos de datos básicos

Datos y tipos de datos

C++ es un **lenguaje fuertemente tipado**, por tanto, los tipos de datos determinan las características y el comportamiento de las variables que se

El **tipo de dato** representa la clase de datos con el que trabajaremos. Podemos clasificar los tipos de datos tal que:

- **Tipos de datos simples**:
    - Números enteros (`int`)
    - Números reales o en coma flotante (`float`, `double`)
    - Valores lógicos o booleanos (`bool`)
    - Caracteres individuales (`char`)
- **Tipos de datos complejos**:
    - Enumeraciones (`enum`)
    - Estructuras (`struct`)
    - *Arrays*
    - Cadenas de caracteres (`string`)
    - Uniones (`union`)
    - Clases (`class`)

Los datos con los que podemos trabajar en un programa son:

- **Literales**: que permiten representar valores. Por ejemplo, un literal entero podría ser el 5.
- **Variables**: son un identificador que guarda un valor. Las variables se declaran de un determinado tipo de datos, así, una variable entera puede guardar datos enteros.
- **Constantes**: mientras que el valor de una variable puede cambiar, las constantes no pueden cambiar durante la ejecución del programa
- **Expresiones**: permiten hacer operaciones entre los distintos datos. El tipo de dato de una expresión dependerá del resultado de la operación.

Según el tipo de datos con los que trabajemos tenemos distintos tipos de operadores:

- **Operadores aritméticos**: para hacer operaciones con tipos de datos numéricos.
- **Operadores relacionados**: permiten comparar datos y nos devuelven valores lógicos.
- **Operadores lógicos**: permiten trabajar con valores lógicos.
- **Operadores de asignación**: permiten asignar valores a variables.
- **Otros operadores**: veremos algunos operadores más, por ejemplo, para trabajar con bits o con punteros.

La **precedencia o prioridad de los operadores** es la siguiente:

Prioridad	Operador	Descripción	Asociatividad
1	`()`	Paréntesis	N/A
2	`++, --`	Incremento y decremento (postfijo)	dcha a izda
3	`+, -`	Unario positivo y negativo	dcha a izda
4	`*, /, %`	Multiplicación, división, módulo	izda a dcha
5	`+, -`	Adición, sustracción	izda a dcha
6	`<<, >>`	Desplazamiento de bits	izda a dcha
7	`<, >, <=, >=`	Comparación	izda a dcha
8	`==, !=`	Igualdad, desigualdad	izda a dcha
9	`&`	AND bit a bit	izda a dcha
10	`^`	XOR bit a bit	izda a dcha
11	`` ` ``	OR bit a bit	izda a dcha
12	`&&`	AND lógico	izda a dcha
13	`||`	OR lógico	izda a dcha
14	`?:`	Operador condicional (ternario)	dcha a izda
15	`=, +=, -=`	Asignación y operadores compuestos	dcha a izda

Los operadores con mayor precedencia se evalúan antes que los operadores con menor precedencia. Si dos operadores tienen la misma precedencia, su evaluación se determina por la asociatividad del operador (izquierda a derecha o derecha a izquierda).

Aunque la precedencia de operadores permite omitir paréntesis, es una buena práctica usarlos para mejorar la claridad del código.

Literales

Los **literales** permiten representar valores, los cuales pueden ser de diferentes tipos. De esta manera tenemos diferentes tipos de literales:

- **Literales enteros**: Para representar números enteros utilizamos cifras enteras. Ejemplos números en base decimal: `5`,`-12`..., en base octal: `077` y en hexadecimal `0xfe`.
- **Literales reales**: Utilizamos un punto para separar la parte entera de la decimal. Por ejemplo: `3.14159`. También podemos usar la letra `e` o `E` seguida de un exponente con signo para indicar la potencia de 10 a utilizar, por ejemplo: `6.63e-34`, `35E20`.
- **Literales booleanos o lógicos**: Los valores lógicos solo tienen dos valores: `false` para indicar el valor falso, y `true` para indicar el valor verdadero.
- **Literales carácter**: Para indicar un valor de tipo carácter usamos la comilla simple `'`. Por ejemplo `'a'`. Tenemos algunos caracteres especiales que son muy útiles, por ejemplo `\n` indica nueva línea y `\t` indica tabulador.
- **Literales cadenas de caracteres**: Una cadena de caracteres es un conjunto de caracteres. Para indicar cadenas de caracteres usamos las dobles comillas `"`. Por ejemplo: `"Hola"`.

Constantes

Una **constante** es un identificador que utilizamos para representar **un valor no puede ser modificado** durante la ejecución del programa.

Su uso facilita la actualización y el mantenimiento del código (si necesitas cambiar el valor de una constante, solo necesitas hacerlo en un lugar).

Para definir constantes usamos:

```
#define identificador valor
```