



Taller #1 de practicas intermedias

Edgardo Andrés Nil Guzmán

carnet 201801119

Duración: 2horas

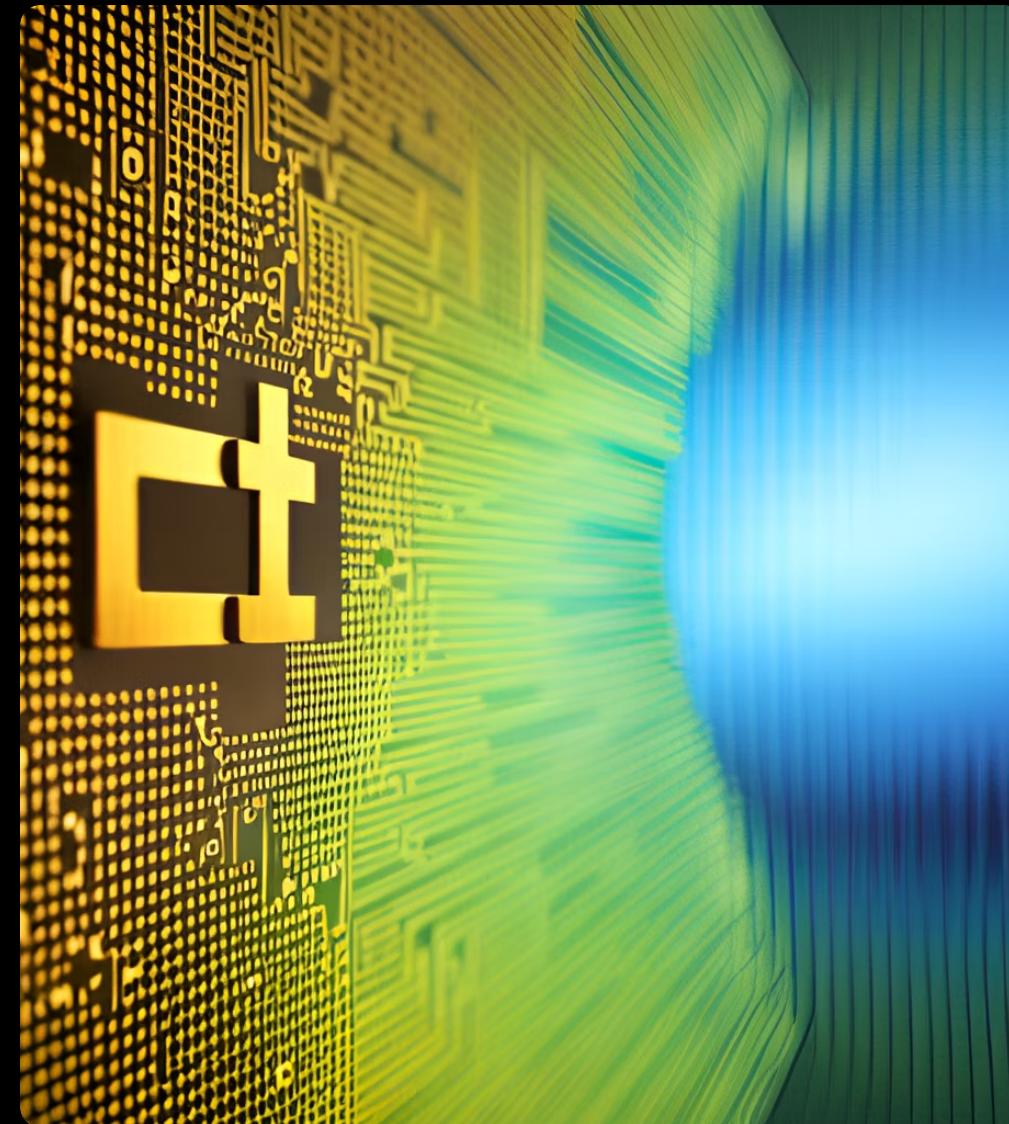
3:00-5:00

Fecha: 19-10-2023





Introducción a C++



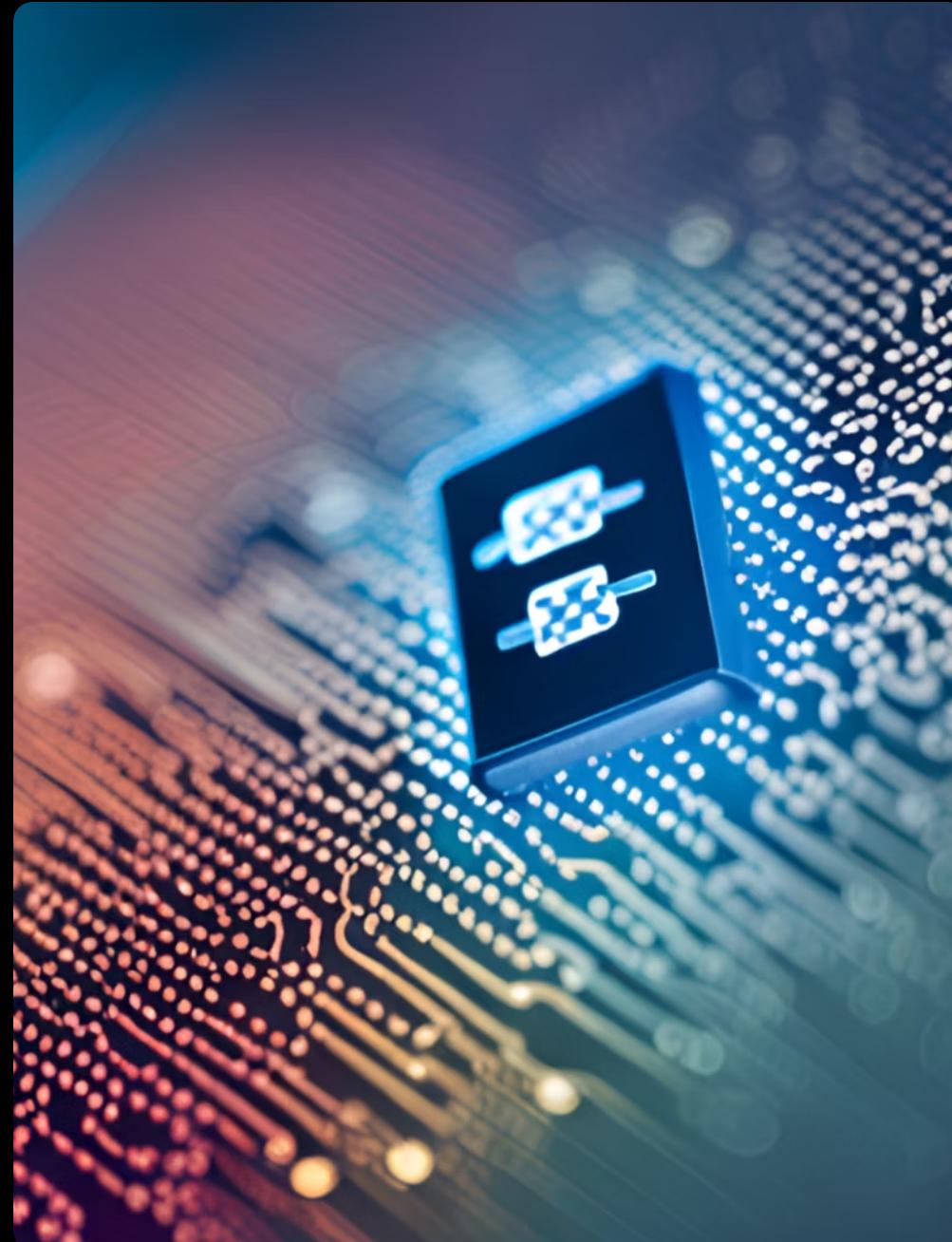


Introducción a la programación

La programación es el proceso de crear programas de computadora utilizando lenguajes de programación. C++ es un lenguaje de programación de alto nivel que se utiliza para crear aplicaciones de software, sistemas operativos, juegos y mucho más. C++ es un lenguaje de programación popular debido a su eficiencia, flexibilidad y capacidad para trabajar con sistemas de bajo nivel.

En esta sección, se presentarán los conceptos básicos de programación, incluyendo variables y tipos de datos, estructuras de control de flujo, funciones, arreglos, punteros, estructuras, clases y objetos, herencia, polimorfismo, plantillas, excepciones, entrada y salida de archivos y bibliotecas estándar de C++.

Además, se explorarán los diferentes tipos de programación que se pueden realizar con C++, incluyendo programación orientada a objetos, programación genérica, programación de bajo nivel, programación de aplicaciones gráficas, programación de juegos, programación de dispositivos embebidos, programación de sistemas operativos y programación de redes.





¿Qué es C++?

C++ es un lenguaje de programación de alto nivel y de propósito general que se utiliza para desarrollar aplicaciones de software. Fue creado en 1983 por Bjarne Stroustrup como una extensión del lenguaje de programación C. C++ es un lenguaje compilado, lo que significa que el código fuente se compila en un archivo ejecutable que se puede ejecutar en una máquina específica.

Ejemplo de programa en C++

Este es un ejemplo simple de un programa en C++ que imprime "¡Hola, mundo!" en la pantalla:

```
#include <iostream>

using namespace std;

int main()
{
    cout << "¡Hola, mundo!" << endl;
    return 0;
}
```



Variables y tipos de datos

En C++, una variable es un espacio en memoria que se utiliza para almacenar un valor, el cual puede ser modificado durante la ejecución del programa. Cada variable tiene un tipo de dato, el cual define qué valores puede almacenar y cómo se almacenan en memoria.

Tipos de datos

Los tipos de datos básicos en C++ son:

- int: para números enteros.
- float: para números decimales de precisión simple.
- double: para números decimales de precisión doble.
- char: para caracteres individuales.
- bool: para valores verdadero o falso.

Además, C++ permite definir tipos de datos personalizados mediante la creación de estructuras y clases.

Declaración de variables

Para declarar una variable en C++, se debe especificar su tipo de dato y un nombre único que la identifique. Por ejemplo:

```
int edad;  
float salario;  
char inicial;
```

En este ejemplo, se declaran tres variables: edad de tipo int, salario de tipo float e inicial de tipo char.



Operadores

Los operadores son símbolos que se utilizan para realizar operaciones matemáticas y lógicas en C++. A continuación, se presentan algunos de los operadores más comunes:

Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas básicas, como sumar, restar, multiplicar y dividir. Algunos de los operadores aritméticos más comunes son:

- + (suma)
- - (resta)
- * (multiplicación)
- / (división)

Operadores de asignación

Los operadores de asignación se utilizan para asignar un valor a una variable. El operador de asignación más común es el signo igual (=). Por ejemplo:

```
int x = 5;
```

Operadores de comparación

Los operadores de comparación se utilizan para comparar dos valores y devolver un valor booleano (verdadero o falso). Algunos de los operadores de comparación más comunes son:

- == (igual a)
- != (diferente de)
- < (menor que)
- > (mayor que)

A continuación se muestra un ejemplo de cómo se pueden utilizar algunos de estos operadores en C++:

```
int x = 5;  
int y = 3;  
int z = x + y;  
bool a = (x > y);  
bool b = (x == y);  
bool c = (x != y);
```

Estructuras de control de flujo

Las estructuras de control de flujo son herramientas que permiten controlar el flujo de ejecución de un programa. Estas estructuras permiten tomar decisiones y repetir acciones en función de ciertas condiciones.

Estructura if

La estructura if permite ejecutar un bloque de código si se cumple una condición. La sintaxis es la siguiente:

```
if (condicion) {  
    // bloque de código  
}
```

Por ejemplo, el siguiente código muestra un mensaje si el número ingresado por el usuario es positivo:

```
int numero;  
cout << "Ingrese un numero: ";  
cin >> numero;  
if (numero > 0) {  
    cout << "El numero es positivo";  
}
```

Estructura if-else

La estructura if-else permite ejecutar un bloque de código si se cumple una condición, y otro bloque de código si no se cumple. La sintaxis es la siguiente:

```
if (condicion) {  
    // bloque de código si se cumple la condición  
} else {  
    // bloque de código si no se cumple la condición  
}
```

Por ejemplo, el siguiente código muestra un mensaje si el número ingresado por el usuario es positivo, y otro mensaje si es negativo:

```
int numero;  
cout << "Ingrese un numero: ";  
cin >> numero;  
if (numero > 0) {  
    cout << "El numero es positivo";  
} else {  
    cout << "El numero es negativo";  
}
```

Estructura switch

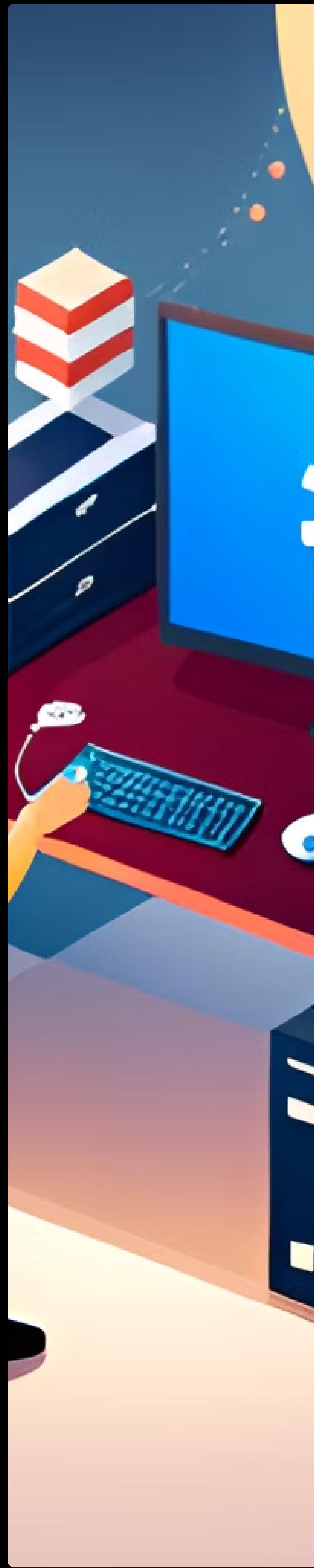
La estructura switch permite ejecutar diferentes bloques de código en función del valor de una variable. La sintaxis es la siguiente:

```
switch (variable) {  
    case valor1:  
        // bloque de código si variable es igual a valor1  
        break;  
    case valor2:  
        // bloque de código si variable es igual a valor2  
        break;  
    default:  
        // bloque de código si variable no es igual a ninguno  
        de los valores anteriores  
}
```

Por ejemplo, el siguiente código muestra un mensaje en función del día de la semana:

```
int diaSemana;  
cout << "Ingrese el dia de la semana (1-7): ";  
cin >> diaSemana;
```

```
switch (diaSemana) {  
    case 1:  
        cout << "Lunes";  
        break;  
    case 2:  
        cout << "Martes";  
        break;  
    case 3:  
        cout << "Miercoles";  
        break;  
    case 4:  
        cout << "Jueves";  
        break;  
    case 5:  
        cout << "Viernes";  
        break;  
    case 6:  
        cout << "Sabado";  
        break;  
    case 7:  
        cout << "Domingo";  
        break;  
    default:  
        cout << "Dia invalido";  
}
```





Funciones

Las funciones son bloques de código que realizan una tarea específica y pueden ser llamadas desde cualquier parte del programa. Son una herramienta esencial en C++ para modularizar el código y hacerlo más fácil de entender y mantener.

Sintaxis

La sintaxis básica de una función es la siguiente:

```
tipo_de_retorno nombre_de_la_función(argumentos) {  
    // Código de la función  
    return valor_de_retorno;  
}
```

Donde:

- **tipo_de_retorno**: el tipo de dato que devuelve la función. Puede ser cualquier tipo de dato válido en C++, incluyendo void si la función no devuelve ningún valor.
- **nombre_de_la_función**: el nombre de la función, que debe ser único dentro del programa.
- **argumentos**: los valores que se pasan a la función como entrada. Pueden ser cualquier tipo de dato válido en C++, y pueden ser uno o varios.
- **valor_de_retorno**: el valor que devuelve la función. Este valor debe ser del mismo tipo que el tipo_de_retorno de la función, o void si la función no devuelve ningún valor.

Ejemplo

A continuación se muestra un ejemplo de una función que calcula el área de un rectángulo:

```
float calcular_area_rectangulo(float base, float altura) {  
    float area = base * altura;  
    return area;  
}
```

Esta función recibe dos argumentos, la base y la altura del rectángulo, y devuelve el área como un valor de tipo float. Para llamar a esta función desde otra parte del programa, simplemente se utiliza el nombre de la función y se le pasan los argumentos correspondientes:

```
float area = calcular_area_rectangulo(5.0, 3.0);
```

Conclusión

Las funciones son una herramienta esencial en C++ para modularizar el código y hacerlo más fácil de entender y mantener. Con una sintaxis sencilla y versátil, las funciones permiten realizar tareas específicas de manera eficiente y reutilizable en diferentes partes del programa.



Arreglos

Los arreglos son estructuras de datos que permiten almacenar una colección de elementos del mismo tipo. Cada elemento en un arreglo está identificado por un índice, que comienza en 0 y termina en el tamaño del arreglo menos 1.

Por ejemplo, podemos declarar un arreglo de enteros de la siguiente manera:

```
int numeros[5];
```

Este arreglo tiene un tamaño de 5 elementos. Podemos asignar valores a cada elemento utilizando el índice correspondiente:

```
numeros[0] = 10;numeros[1] = 20;numeros[2] = 30;numeros[3] = 40;numeros[4] = 50;
```

También podemos inicializar un arreglo al momento de declararlo:

```
int numeros[5] = {10, 20, 30, 40, 50};
```



Punteros

Los punteros son una característica importante de C++ que permiten trabajar con la memoria de la computadora de manera más eficiente. Un puntero es una variable que almacena la dirección de memoria de otra variable.

Declaración y uso de punteros

Para declarar un puntero, se utiliza el operador "*" antes del nombre de la variable. Por ejemplo, para declarar un puntero a un entero, se utilizaría la siguiente sintaxis:

```
int* miPuntero;
```

Para asignar la dirección de memoria de una variable a un puntero, se utiliza el operador "&" antes del nombre de la variable. Por ejemplo, para asignar la dirección de memoria de la variable "miVariable" a "miPuntero", se utilizaría la siguiente sintaxis:

```
miPuntero = &miVariable;
```

Para acceder al valor almacenado en la dirección de memoria apuntada por un puntero, se utiliza el operador "*" antes del nombre del puntero. Por ejemplo, para acceder al valor almacenado en la dirección de memoria apuntada por "miPuntero", se utilizaría la siguiente sintaxis:

```
int valor = *miPuntero;
```

Ejemplo de uso de punteros

En este ejemplo, se utiliza un puntero para intercambiar los valores de dos variables enteras:

```
void intercambiar(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int x = 5;
    int y = 10;
    intercambiar(&x, &y);
    std::cout << x << " " << y << std::endl;
    return 0;
}
```



Estructuras

Las estructuras son tipos de datos definidos por el usuario que permiten almacenar diferentes tipos de datos relacionados en una sola entidad. Por ejemplo, si queremos almacenar información sobre una persona, podemos crear una estructura que contenga su nombre, edad y dirección.

La sintaxis para crear una estructura en C++ es la siguiente:

```
struct NombreEstructura {  
    tipoDato1 variable1;  
    tipoDato2 variable2;  
    tipoDato3 variable3;  
};
```

Por ejemplo, podemos crear una estructura para almacenar información sobre un estudiante:

```
struct Estudiante {  
    string nombre;  
    int edad;  
    string carrera;  
};
```

Una vez que hemos definido la estructura, podemos crear variables de ese tipo y acceder a sus miembros:

```
Estudiante estudiante1;  
estudiante1.nombre = "Juan Perez";  
estudiante1.edad = 20;  
estudiante1.carrera = "Ingeniería en Sistemas";
```



Clases y objetos

C++ es un lenguaje de programación orientado a objetos, lo que significa que se basa en el concepto de clases y objetos. Una clase es una plantilla que define las propiedades y los métodos que tendrán los objetos creados a partir de ella. Un objeto es una instancia de una clase, es decir, una variable que contiene los datos y los comportamientos definidos por la clase.

Veamos un ejemplo sencillo:

```
class Persona {  
public:  
    string nombre;  
    int edad;  
};  
  
int main() {  
    Persona persona1;  
    persona1.nombre = "Juan";  
    persona1.edad = 25;  
    cout << persona1.nombre << " tiene " << persona1.edad << " años." << endl;  
    return 0;  
}
```

En este ejemplo, se define la clase Persona con dos propiedades: nombre y edad. Luego se crea un objeto de la clase Persona llamado persona1, se le asignan valores a sus propiedades y se imprime su información en la consola.



Herencia en C++

La herencia es uno de los conceptos clave de la programación orientada a objetos en C++. Permite que una clase herede los atributos y métodos de otra clase, lo que puede ahorrar tiempo y esfuerzo en la programación. Por ejemplo, si creamos una clase "Animal" con atributos como "nombre" y "edad", y métodos como "comer" y "dormir", podemos crear una clase "Perro" que herede estos atributos y métodos, y agregar otros específicos de la clase "Perro", como "ladrar" y "jugar".

Ejemplo de herencia en C++

Aquí hay un ejemplo simple de cómo se ve la herencia en C++:

```
// Clase base
class Animal {
protected:
    string nombre;
    int edad;
public:
    void comer() {
        cout << "El animal está comiendo" << endl;
    }
};

// Clase derivada
class Perro : public Animal {
public:
    void ladrar() {
        cout << "El perro está ladrando" << endl;
    }
};

int main() {
    Perro miPerro;
    miPerro.comer(); // Hereda el método de la clase base
    miPerro.ladrar(); // Método específico de la clase Perro
    return 0;
}
```

Explicación del ejemplo

En este ejemplo, creamos una clase base "Animal" con los atributos "nombre" y "edad", y el método "comer". Luego, creamos una clase derivada "Perro" que hereda estos atributos y métodos, y agrega el método "ladrar". En el programa principal, creamos un objeto de la clase "Perro" y llamamos a los métodos "comer" y "ladrar".



Polimorfismo

El polimorfismo es una técnica de programación orientada a objetos que permite a los objetos de diferentes clases responder al mismo mensaje o método de manera diferente. En C++, el polimorfismo se puede lograr mediante el uso de clases abstractas e interfaces.

Ejemplo

Supongamos que tenemos una clase Animal y dos clases derivadas, Perro y Gato. Ambas clases derivadas tienen un método llamado sonido() que devuelve el sonido que hace el animal correspondiente. Podemos crear un puntero a la clase Animal y asignarle una instancia de Perro o Gato. Luego, podemos llamar al método sonido() en el puntero y obtener el sonido correspondiente del animal:

```
class Animal {  
public:  
    virtual std::string sonido() = 0;  
};  
class Perro : public Animal {  
public:  
    std::string sonido() override { return "Guau!"; }  
};  
class Gato : public Animal {  
public:  
    std::string sonido() override { return "Miau!"; }  
};  
  
int main() {  
    Animal* animal1 = new Perro();  
    Animal* animal2 = new Gato();  
    std::cout << animal1->sonido() << std::endl; // Output: Guau!  
    std::cout << animal2->sonido() << std::endl; // Output: Miau!  
}
```



Plantillas

Las plantillas son una característica de C++ que permite la creación de funciones y clases genéricas. Esto significa que se pueden crear funciones y clases que trabajen con diferentes tipos de datos sin tener que escribir código específico para cada tipo.

Funciones Plantilla

Una función plantilla es una función que puede trabajar con diferentes tipos de datos. Para crear una función plantilla, se utiliza la palabra clave "template" seguida de los parámetros de plantilla y el cuerpo de la función.

```
Ejemplo:template <typename T>
T max(T a, T b) {
    return a > b ? a : b;
}

int main() {
    int x = 5, y = 10;
    double d1 = 3.14, d2 = 2.72;

    std::cout << max(x, y) << std::endl; // Output: 10
    std::cout << max(d1, d2) << std::endl; // Output: 3.14

    return 0;
}
```

Clases Plantilla

Una clase plantilla es una clase que puede trabajar con diferentes tipos de datos. Para crear una clase plantilla, se utiliza la palabra clave "template" seguida de los parámetros de plantilla y el cuerpo de la clase.

```
Ejemplo:template <typename T>
class MiClase {
public:
    T dato;

    MiClase(T d) {
        dato = d;
    }

    void mostrarDatos() {
        std::cout << "Dato: " << dato << std::endl;
    }
};

int main() {
    MiClase<int> c1(5);
    MiClase<std::string> c2("holo");

    c1.mostrarDatos(); // Output: Dato: 5
    c2.mostrarDatos(); // Output: Dato: hola

    return 0;
}
```



Excepciones

Las excepciones son eventos que ocurren durante la ejecución de un programa que interrumpen el flujo normal de ejecución. En C++, podemos manejar estas excepciones para evitar que el programa se detenga abruptamente y proporcionar una salida elegante al usuario.

Sintaxis básica

Para lanzar una excepción en C++, se utiliza la palabra clave "throw" seguida de un objeto de excepción. Por ejemplo:

```
throw runtime_error("Ocurrió un error.");
```

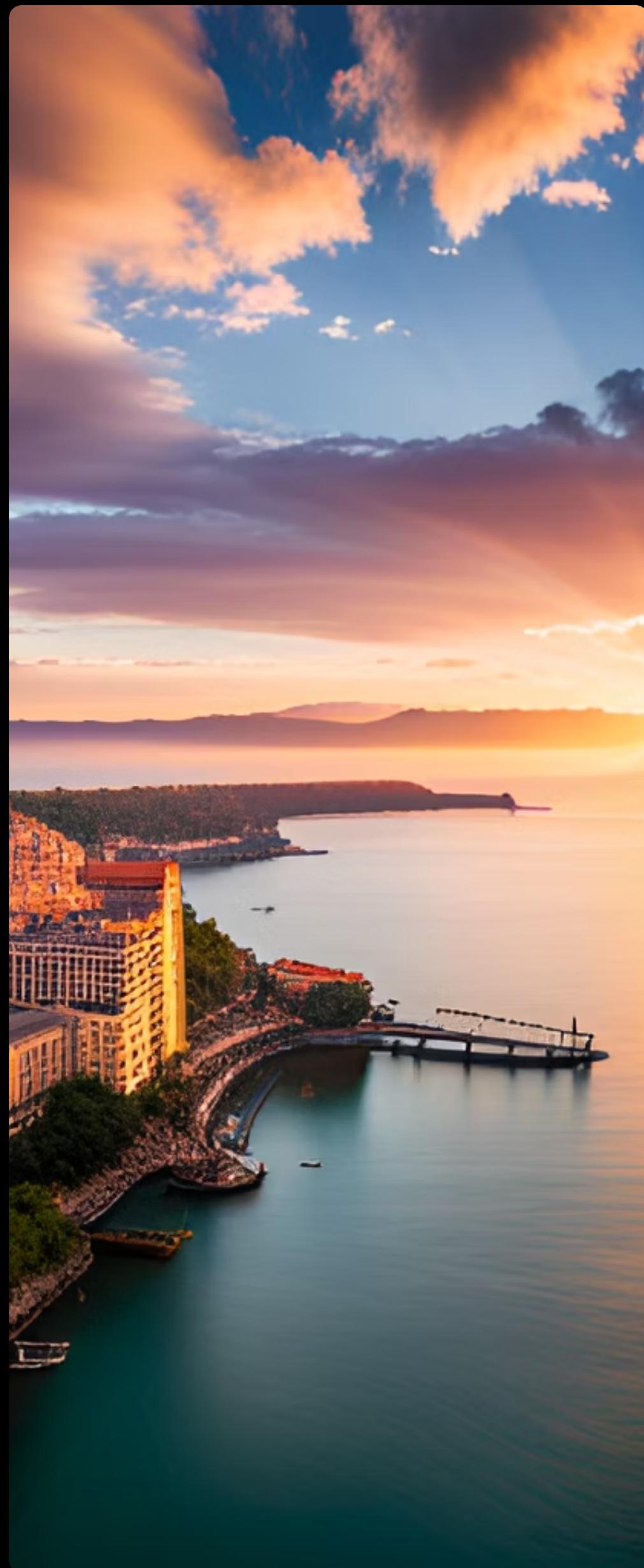
Para manejar una excepción en C++, se utiliza un bloque "try-catch". El bloque "try" contiene el código que puede lanzar una excepción, mientras que el bloque "catch" contiene el código que maneja la excepción. Por ejemplo:

```
try {  
    // código que puede lanzar una excepción  
}  
catch (exception& e) {  
    // código que maneja la excepción  
}
```

Ejemplo

Supongamos que queremos dividir dos números ingresados por el usuario, pero queremos manejar la excepción de división por cero. Podemos hacerlo de la siguiente manera:

```
try {  
    int a, b;  
    cout << "Ingrese dos números: ";  
    cin >> a >> b;  
    if (b == 0) {  
        throw runtime_error("División por cero.");  
    }  
    cout << "Resultado: " << a / b << endl;  
}  
catch (exception& e) {  
    cout << "Error: " << e.what() << endl;  
}
```



Entrada y salida de archivos

En C++, podemos leer y escribir archivos utilizando las bibliotecas estándar de entrada y salida. Para leer un archivo, primero debemos abrirlo en modo de lectura y luego leer los datos del archivo. Para escribir en un archivo, primero debemos abrirlo en modo de escritura y luego escribir los datos en el archivo.

Lectura de archivos

Para leer un archivo en C++, primero debemos abrirlo utilizando la función "fstream" y especificando el nombre del archivo y el modo de apertura. El modo de apertura puede ser "in" para lectura o "out" para escritura. Luego podemos leer los datos del archivo utilizando la función "getline" o ">>".

Ejemplo

Supongamos que tenemos un archivo llamado "datos.txt" con los siguientes datos:

- Juan Perez
- Maria Garcia
- Pedro Sanchez

Podemos leer los datos del archivo de la siguiente manera:

```
#include <iostream>
#include <fstream>
```

```
using namespace std;
```

```
int main()
{
    string nombre;
    ifstream archivo("datos.txt");

    if(archivo.is_open())
    {
        while(getline(archivo, nombre))
        {
            cout << nombre << endl;
        }
        archivo.close();
    }
    else
    {
        cout << "No se pudo abrir el archivo" << endl;
    }

    return 0;
}
```

Este programa abrirá el archivo "datos.txt", leerá los datos línea por línea y los imprimirá en la pantalla.

Escritura de archivos

Para escribir en un archivo en C++, primero debemos abrirlo utilizando la función "ofstream" y especificando el nombre del archivo y el modo de apertura. El modo de apertura puede ser "in" para lectura o "out" para escritura. Luego podemos escribir los datos en el archivo utilizando la función "<<".

Ejemplo

Supongamos que queremos escribir los siguientes datos en un archivo llamado "salida.txt":

- Juan Perez
- Maria Garcia
- Pedro Sanchez

Podemos escribir los datos en el archivo de la siguiente manera:

```
#include <iostream>
#include <fstream>
```

```
using namespace std;
```

```
int main()
{
    ofstream archivo("salida.txt");

    if(archivo.is_open())
    {
        archivo << "Juan Perez\n";
        archivo << "Maria Garcia\n";
        archivo << "Pedro Sanchez\n";
        archivo.close();
    }
    else
    {
        cout << "No se pudo abrir el archivo" << endl;
    }

    return 0;
}
```

Este programa creará el archivo "salida.txt" y escribirá los datos en él.



Bibliotecas estándar de C++

iostream

La biblioteca iostream proporciona funciones para la entrada y salida de datos en C++. Incluye los objetos cin y cout para la entrada y salida de datos por consola, respectivamente.

Ejemplo:

```
#include <iostream>

using namespace std;

int main() {
    int num;
    cout << "Ingrese un número entero: ";
    cin >> num;
    cout << "El número ingresado es: " <<
num << endl;
    return 0;
}
```

string

La biblioteca string proporciona una clase para el manejo de cadenas de caracteres en C++. Incluye funciones para la concatenación, comparación y búsqueda de cadenas, entre otras.

Ejemplo:

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string nombre;
    cout << "Ingrese su nombre: ";
    getline(cin, nombre);
    cout << "Hola, " << nombre << "!" <<
endl;
    return 0;
}
```



Programación orientada a objetos

La programación orientada a objetos (POO) es un paradigma de programación que se basa en el uso de objetos para representar conceptos del mundo real. C++ es un lenguaje de programación que soporta la POO y nos permite crear clases y objetos para modelar nuestros programas.

Algunos de los conceptos clave de la POO en C++ son:

- Clases: son plantillas que nos permiten crear objetos con características similares.
- Objetos: son instancias de una clase y representan una entidad del mundo real.
- Atributos: son las variables que definen el estado de un objeto.
- Métodos: son las funciones que pueden ser llamadas por los objetos de una clase.

Veamos un ejemplo sencillo de cómo se puede utilizar la POO en C++ para crear una clase "Persona":

```
class Persona {  
public:  
    string nombre;  
    int edad;  
    void saludar() {  
        cout << "Hola, mi nombre es " << nombre << " y tengo " << edad << " años" << endl;  
    }  
};  
  
int main() {  
    Persona persona1;  
    persona1.nombre = "Juan";  
    persona1.edad = 30;  
    persona1.saludar();  
    return 0;  
}
```

En este ejemplo, hemos creado una clase "Persona" con dos atributos (nombre y edad) y un método (saludar) que imprime un saludo en la consola. Luego, en la función "main", creamos un objeto de la clase "Persona" y le asignamos valores a sus atributos para finalmente llamar al método "saludar".



Programación genérica

La programación genérica es una técnica de programación que permite escribir algoritmos y estructuras de datos independientes del tipo de dato con el que se trabajará. En C++, esto se logra mediante el uso de plantillas.

Ejemplo de plantilla de función

A continuación, se muestra un ejemplo de una plantilla de función que intercambia el valor de dos variables de cualquier tipo:

```
template <typename T>
void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 5, y = 10;
    swap(x, y);
    // Ahora x es 10 y y es 5
    return 0;
}
```



Programación de bajo nivel

C++ es un lenguaje de programación de bajo nivel que permite un control más preciso y cercano al hardware. La programación de bajo nivel se refiere a la escritura de código que interactúa directamente con el hardware de la computadora, como la memoria y los registros de la CPU. Esto puede ser útil para aplicaciones que requieren un alto rendimiento o que necesitan acceder a recursos del sistema de bajo nivel.

Algunas de las características de C++ que lo hacen ideal para la programación de bajo nivel son:

- Acceso directo a la memoria y los registros de la CPU.
- Control preciso sobre la gestión de recursos de la computadora.
- Capacidad para escribir código optimizado para un hardware específico.

A continuación, se presentan algunos ejemplos de programación de bajo nivel en C++:

Acceso directo a la memoria

En C++, es posible acceder directamente a la memoria de la computadora utilizando punteros. Por ejemplo, el siguiente código asigna un valor a una ubicación específica de memoria:

```
int *ptr = (int*)0x1000;  
*ptr = 42;
```

Este código asigna el valor 42 a la dirección de memoria 0x1000. Sin embargo, es importante tener en cuenta que el acceso directo a la memoria puede ser peligroso y debe usarse con precaución.

Optimización de código

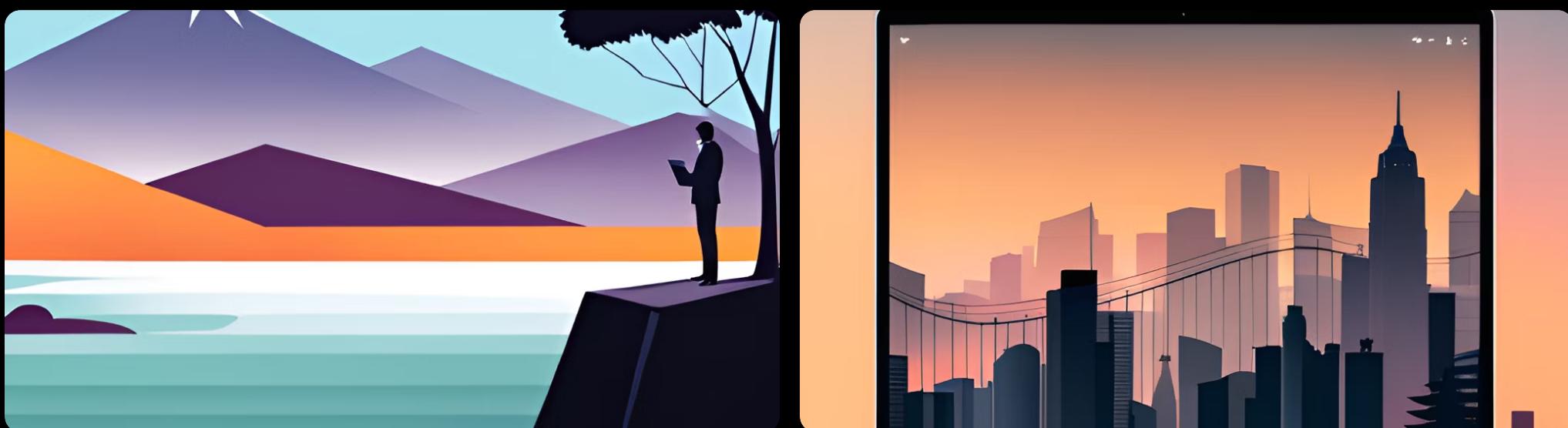
C++ permite escribir código optimizado para un hardware específico utilizando instrucciones de ensamblaje. Por ejemplo, el siguiente código utiliza una instrucción de ensamblaje para sumar dos números enteros sin utilizar la operación de suma de C++:

```
int a = 5;  
int b = 7;  
asm("addl %%ebx,%%eax;" : "=a"(a) : "a"(a), "b"(b));
```

Este código utiliza la instrucción de ensamblaje "addl" para sumar los valores de a y b y almacenar el resultado en a. La sintaxis de la instrucción de ensamblaje puede variar según la plataforma de hardware y el compilador utilizado.



Programación de aplicaciones gráficas



Biblioteca gráfica SFML

SFML es una biblioteca gráfica multiplataforma para C++ que proporciona una API sencilla y coherente para el desarrollo de aplicaciones multimedia, incluyendo gráficos 2D y 3D, sonido, entrada de teclado y ratón, etc. Es fácil de aprender y usar, y es adecuada tanto para principiantes como para desarrolladores experimentados.

Ejemplo: Creación de una ventana

El siguiente código muestra cómo crear una ventana con SFML:

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(800, 600),
    "Mi ventana");

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();
        // Dibujar aquí
        window.display();
    }

    return 0;
}
```

Ejemplo: Dibujar un círculo

El siguiente código muestra cómo dibujar un círculo en una ventana con SFML:

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(800, 600),
    "Mi ventana");

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();

        sf::CircleShape shape(100.f);
        shape.setFillColor(sf::Color::Green);
        shape.setPosition(200.f, 150.f);
        window.draw(shape);

        window.display();
    }

    return 0;
}
```



Programación de juegos

C++ es un lenguaje de programación popular para la creación de juegos debido a su eficiencia y flexibilidad. Los juegos modernos pueden ser muy complejos, pero con C++, puedes crear juegos simples y divertidos mientras aprendes los conceptos básicos de programación.

Creando un juego simple

Un juego simple puede ser creado utilizando la biblioteca gráfica SFML (Simple and Fast Multimedia Library). Aquí hay un ejemplo de un juego simple en C++:

```
#include <SFML/Graphics.hpp>
```

```
int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "My Game");
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        window.clear();
        window.display();
    }
    return 0;
}
```

Este código crea una ventana de 200x200 píxeles con el título "My Game". El bucle principal del juego espera eventos, como hacer clic en el botón de cerrar ventana, y dibuja la ventana en cada iteración.

Añadiendo gráficos

Para hacer el juego más interesante, podemos agregar gráficos. Aquí hay un ejemplo de cómo agregar una imagen a la ventana:

```
#include <SFML/Graphics.hpp>
```

```
int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "My Game");
    sf::Texture texture;
    if (!texture.loadFromFile("image.png"))
    {
        return EXIT_FAILURE;
    }
    sf::Sprite sprite(texture);
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        window.clear();
        window.draw(sprite);
        window.display();
    }
    return 0;
}
```

Este código carga una imagen llamada "image.png" y la muestra en la ventana. Para hacer esto, se utiliza la clase Texture para cargar la imagen y la clase Sprite para mostrarla.

Añadiendo interacción

Para hacer el juego aún más interesante, podemos agregar interacción. Aquí hay un ejemplo de cómo detectar cuando se hace clic en la imagen:

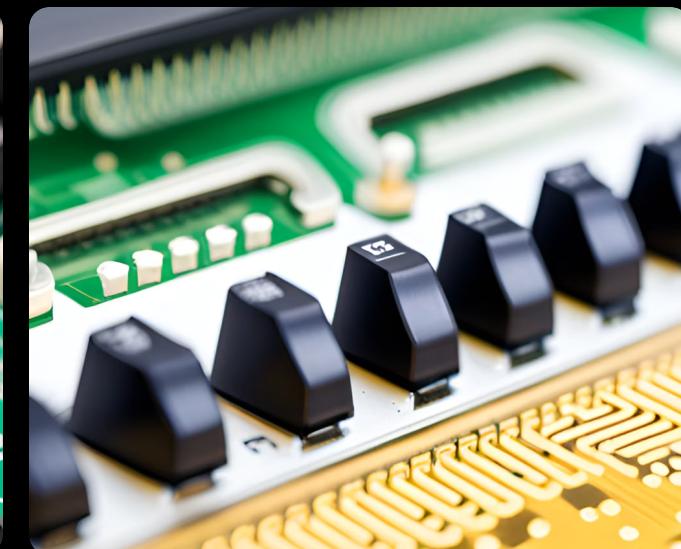
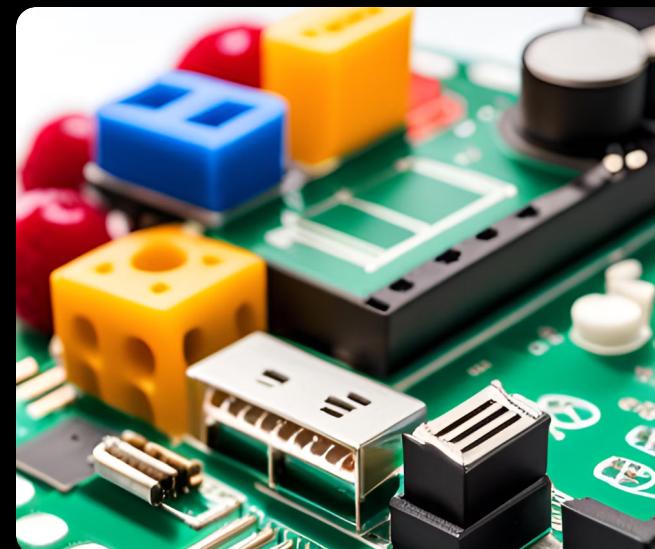
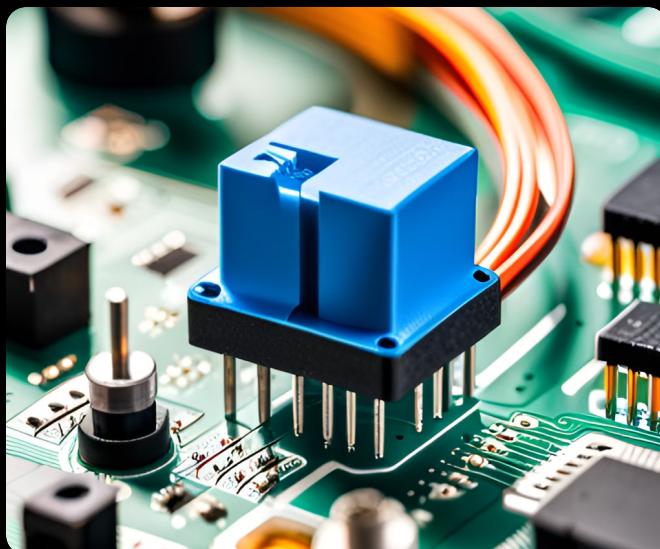
```
#include <SFML/Graphics.hpp>
```

```
int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "My Game");
    sf::Texture texture;
    if (!texture.loadFromFile("image.png"))
    {
        return EXIT_FAILURE;
    }
    sf::Sprite sprite(texture);
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::MouseButtonPressed)
            {
                if (sprite.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y))
                {
                    // Do something when the sprite is clicked
                }
            }
        }
        window.clear();
        window.draw(sprite);
        window.display();
    }
    return 0;
}
```

Este código detecta cuando se hace clic en la ventana y verifica si el clic está dentro de los límites del sprite. Si es así, se puede agregar código para hacer algo cuando se hace clic en el sprite.



Programación de dispositivos embebidos



Introducción a la programación de dispositivos embebidos

Los dispositivos embebidos son sistemas informáticos diseñados para realizar tareas específicas y se encuentran en una amplia variedad de dispositivos, desde electrodomésticos hasta automóviles. La programación de dispositivos embebidos implica el uso de lenguajes de programación de bajo nivel para interactuar directamente con el hardware del dispositivo.

Lenguajes de programación para dispositivos embebidos

C++ es uno de los lenguajes de programación más utilizados para dispositivos embebidos debido a su capacidad para interactuar directamente con el hardware y su eficiencia en términos de uso de recursos.

Ejemplos de programación de dispositivos embebidos con C++

Control de motores con Arduino y C++; Interfaz de usuario con Raspberry Pi y C++; Medición de temperatura con microcontroladores y C++.



Programación de sistemas operativos

La programación de sistemas operativos es una rama avanzada de la programación en C++. Esta área se enfoca en el desarrollo de software que interactúa directamente con el hardware de un sistema, permitiendo la creación de sistemas operativos y drivers de dispositivos. La programación de sistemas operativos es esencial para el desarrollo de sistemas embebidos y aplicaciones de bajo nivel.

Ejemplo de código

A continuación se presenta un ejemplo sencillo de código que utiliza la librería `<iostream>` para imprimir un mensaje en la pantalla del sistema:

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hola mundo!" << endl;
    return 0;
}
```



Programación de redes

La programación de redes en C++ es una habilidad esencial para aquellos que desean desarrollar aplicaciones de red eficientes y escalables. C++ ofrece una amplia variedad de bibliotecas y herramientas para la programación de redes, lo que permite a los desarrolladores crear aplicaciones de red de alta calidad y con un rendimiento excepcional.

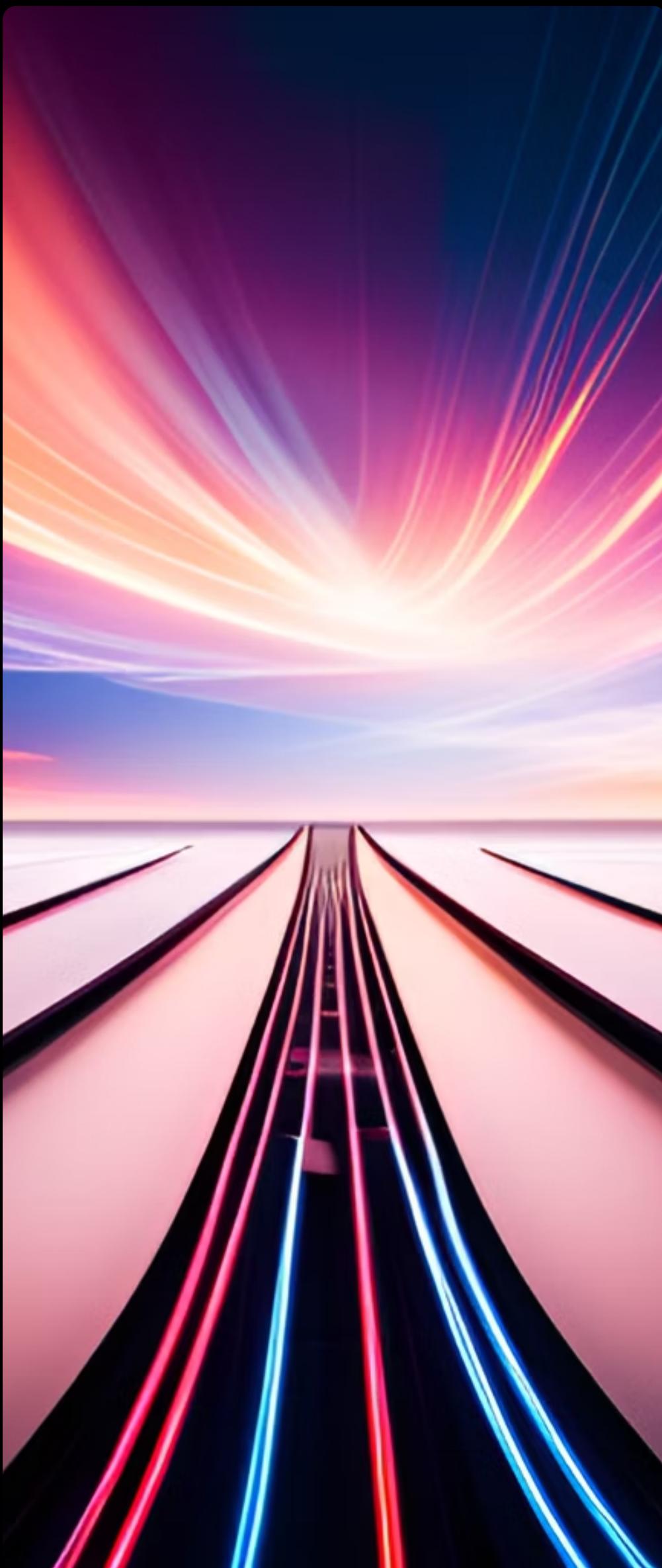
Ejemplo de uso de sockets en C++

Un ejemplo común de programación de redes en C++ es el uso de sockets. Los sockets son una forma de comunicación entre procesos en una red, y son esenciales para la creación de aplicaciones de red en C++. A continuación, se muestra un ejemplo básico de cómo usar sockets en C++:

```
#include <iostream>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

int main() {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons(8080);
    connect(sock, (struct sockaddr *)&server,
            sizeof(server));
    char message[] = "Hello, world!";
    send(sock, message, strlen(message), 0);
    close(sock);
    return 0;
}
```

Este ejemplo muestra cómo crear un socket, conectarse a un servidor y enviar un mensaje. Aunque es un ejemplo muy simple, ilustra los conceptos básicos de la programación de redes en C++ y puede ser utilizado como punto de partida para aplicaciones más complejas.





Conclusiones

En conclusión, C++ es un lenguaje de programación muy poderoso y versátil que permite desarrollar una amplia variedad de aplicaciones, desde sistemas operativos hasta juegos y dispositivos embebidos. Con una curva de aprendizaje un poco pronunciada, pero con la ayuda de ejemplos fáciles de comprender y hacer, cualquier persona puede aprender a programar en C++ y aprovechar todas sus funcionalidades.