****

**Universidad**

**tecnología de Honduras**

**Tercer parcial**

**del**

**Primer periodo**

**Alumno**

Elmer David Cortez Ruiz

**Carrera**

Ingeniería en computación

**Catedrático**

Ing. Jair Hernández

**Espacio formativo**

Análisis y diseño de algoritmos

**Fecha de entrega**

**18/03/2025**

**Investigar comentarios y estructuras de c++**

Los comentarios son notas que se añaden al código fuente para explicar lo que hace el código. Son ignorados por el compilador, lo que significa que no afectan la ejecución del programa. Los comentarios son cruciales para hacer que el código sea más legible y comprensible, tanto para el programador original como para otros que puedan leerlo o modificarlo en el futuro.

En C++, existen dos tipos principales de comentarios:

1. Comentarios de una sola línea (//):

\* Se utilizan para comentarios cortos que caben en una sola línea.

\* Comienzan con dos barras diagonales (//).

\* Todo lo que esté a la derecha de // hasta el final de la línea se considera un comentario.

#include <iostream> // Incluye la biblioteca para entrada/salida

int main() {

int numero = 10; // Declara una variable entera llamada numero y le asigna el valor 10

std::cout << "El valor es: " << numero << std::endl; // Imprime el valor de la variable en la consola

return 0; // Indica que el programa terminó exitosamente

}

2. Comentarios de varias líneas (/ ... /):

\* Se utilizan para comentarios más extensos que pueden ocupar varias líneas.

\* Comienzan con una barra diagonal seguida de un asterisco (/\*).

\* Terminan con un asterisco seguido de una barra diagonal (\*/).

\* Todo lo que esté entre /\* y \*/ se considera un comentario, incluso si abarca varias líneas.

/\*

Este es un comentario de varias líneas.

Puede extenderse por varias líneas

y se utiliza para proporcionar explicaciones más detalladas

sobre una sección de código.

\*/

#include <iostream>

int main() {

// ... código ...

return 0;

}

Buenas prácticas para usar comentarios:

\* Claridad: Los comentarios deben ser claros y concisos, explicando el por qué del código, no solo el qué.

\* Actualización: Mantén los comentarios actualizados a medida que modificas el código. Un comentario desactualizado puede ser más confuso que la ausencia de un comentario.

\* Moderación: No comentes cada línea de código obvia. Concéntrate en explicar lógica compleja, decisiones de diseño o el propósito de bloques de código.

\* Encabezados de archivos y funciones: Utiliza comentarios de varias líneas al principio de los archivos y las funciones para describir su propósito, autor, fecha de creación/modificación, parámetros (en el caso de funciones) y cualquier otra información relevante.

Estructura Fundamental de un Programa en C++

Un programa básico en C++ generalmente sigue la siguiente estructura:

// Comentarios generales sobre el archivo (autor, propósito, fecha, etc.)

#include <iostream> // Directivas de preprocesador (incluir encabezados)

// Declaraciones de espacios de nombres (opcional)

using namespace std; // Ejemplo: usar el espacio de nombres estándar

// Declaraciones de funciones (prototipos) (opcional)

int sumar(int a, int b);

// Declaraciones de variables globales (generalmente desaconsejado)

int variableGlobal = 0;

int main() {

// Cuerpo principal del programa

// Aquí es donde se ejecuta el código principal

// Declaración de variables locales

int numero1 = 5;

int numero2 = 10;

int resultado;

// Llamada a funciones

resultado = sumar(numero1, numero2);

// Salida de datos

cout << "La suma es: " << resultado << endl;

// Retorno de la función main (indica éxito o error)

return 0;

}

// Definiciones de funciones (implementación)

int sumar(int a, int b) {

return a + b;

}

2. Directivas de Preprocesador (#include):

\* Las líneas que comienzan con # son directivas para el preprocesador.

\* #include se utiliza para incluir archivos de encabezado (header files). Estos archivos contienen declaraciones de funciones, clases, constantes y otras entidades que puedes utilizar en tu programa.

\* <iostream> es un encabezado estándar que proporciona funcionalidades para la entrada y salida de datos (como std::cout para imprimir en la consola y std::cin para leer desde la consola).

\* Existen muchos otros encabezados estándar (como <string>, <vector>, <cmath>, etc.) y también puedes crear tus propios archivos de encabezado.

3. Declaraciones de Espacios de Nombres (using namespace std;):

\* Los espacios de nombres (namespaces) se utilizan para organizar el código y evitar conflictos de nombres entre diferentes partes de un programa o entre diferentes bibliotecas.

\* std es el espacio de nombres estándar de C++. Contiene muchas funciones y clases de uso común (como cout, cin, endl, string, etc.).

\* La línea using namespace std; permite utilizar los nombres dentro del espacio de nombres std directamente (por ejemplo, cout en lugar de std::cout).

\* Si bien es común en programas pequeños y para principiantes, en proyectos más grandes se recomienda ser más específico y utilizar std:: antes de los elementos del espacio de nombres para evitar posibles colisiones de nombres.

4. Declaraciones de Funciones (Prototipos):

\* Un prototipo de función declara la existencia de una función antes de que se defina completamente.

\* Incluye el tipo de retorno de la función, el nombre de la función y la lista de parámetros con sus tipos.

\* El prototipo termina con un punto y coma (;).

\* Los prototipos son útiles para permitir que una función llame a otra función que se define más adelante en el mismo archivo o en otro archivo.

5. Declaraciones de Variables Globales (Generalmente Desaconsejado):

\* Las variables globales se declaran fuera de cualquier función y tienen un alcance global, lo que significa que pueden ser accedidas desde cualquier parte del programa.

\* Si bien son posibles, el uso excesivo de variables globales generalmente se desaconseja porque pueden dificultar el seguimiento de cómo y dónde se modifican los datos, lo que puede llevar a errores y hacer que el código sea más difícil de mantener.

6. La Función main():

\* La función main() es el punto de entrada principal de todo programa en C++.

\* La ejecución del programa comienza siempre en la función main().

\* Debe haber exactamente una función main() en cada programa ejecutable de C++.

\* Generalmente, se declara como int main() o int main(int argc, char \*argv[]) (para recibir argumentos de la línea de comandos).

\* La función main() debe devolver un valor entero al sistema operativo. Un valor de 0 generalmente indica que el programa terminó exitosamente.

Vamos a desglosar cada parte de esta estructura:

1. Comentarios Generales:

\* Como se mencionó anteriormente, es una buena práctica comenzar un archivo con un comentario de varias líneas que describa el propósito del archivo, el autor, la fecha de creación o modificación, y cualquier otra información relevante.

7. Cuerpo Principal del Programa:

\* Dentro de la función main(), se escribe la lógica principal del programa.

\* Esto incluye declaraciones de variables locales (variables que solo son accesibles dentro de la función main()), llamadas a otras funciones, estructuras de control de flujo (como if, else, for, while), operaciones de entrada/salida, etc.

8. Definiciones de Funciones (Implementación):

\* La definición de una función contiene el código real que se ejecuta cuando se llama a la función.

\* Incluye el encabezado de la función (tipo de retorno, nombre y lista de parámetros) seguido del cuerpo de la función encerrado entre llaves {}.

**Que son los encabezados de C++ (librerías) y de 4 ejemplos**

Los encabezados permiten modularizar el código y reutilizar componentes, facilitando el mantenimiento y la legibilidad del programa. son archivos que contienen definiciones de funciones, macros, constantes y declaraciones de tipos y clases que se pueden incluir en otros archivos de código fuente. Estos archivos tienen la extensión .h o .hpp. Aquí te dejo algunos ejemplos comunes de encabezados en C++:

1. <iostream>: Proporciona funcionalidades para la entrada y salida estándar, como std::cin y std::cout.

#include <iostream>

1. <vector>: Define la clase std::vector, que es una plantilla de contenedor que proporciona una matriz dinámica.

#include <vector>

1. <string>: Contiene la definición de la clase std::string, que maneja cadenas de caracteres.

#include <string>

1. <cmath>: Proporciona funciones matemáticas como sqrt, sin, cos, etc.

#include <cmath>

**Tipos de datos que se usan en C++**

En C++, existen varios tipos de datos que se utilizan para almacenar diferentes tipos de información

**Tipos de Datos Primitivos**

1. Enteros (int)

* int: Entero estándar.
* short: Entero corto.
* long: Entero largo.
* long long: Entero muy largo.

1. Punto Flotante

* float: Precisión simple.
* double: Precisión doble.
* long double: Precisión extendida.

1. Caracteres

* char: Carácter individual.
* wchar\_t: Carácter ancho (para Unicode)

1. Booleanos

* bool: Valores de verdad (true o false).

**Tipos de Datos Derivados**

1. Arrays

* Colección de elementos del mismo tipo.

1. Punteros

* Almacenan direcciones de memoria.

1. Referencias

* Alias para otra variable.

1. Funciones

* Bloques de código reutilizables.

**Tipos de Datos Definidos por el Usuario**

1. Estructuras (struct)

* Agrupación de diferentes tipos de datos bajo un mismo nombre.

1. Uniones (union)

* Similar a las estructuras, pero los miembros comparten la misma ubicación en memoria.

1. Enumeraciones (enum)

* Conjunto de constantes enteras.

1. Clases (class)

* Base de la programación orientada a objetos en C++.

**Tipos de Datos Vacíos**

1. void

* Indica que una función no retorna ningún valor.

**Tipos de Datos Abstractos**

1. Clases (class)

* Plantillas para crear objetos.

1. Plantillas (template)

* Permiten definir funciones y clases genéricas.

**Entrada y salida de datos en C++ CIN y COUT – cap.2**

C++

**Inclusión de la biblioteca iostream**: Esta biblioteca permite el uso de std::cin y std::cout para la entrada y salida de datos.

**Declaración de variables**: Se declaran las variables numero y nombre para almacenar los datos ingresados por el usuario.

**Entrada de datos:**

* std::cout se utiliza para mostrar mensajes al usuario.
* std::cin se utiliza para leer los datos ingresados por el usuario.

**Salida de datos**: std::cout se utiliza nuevamente para mostrar los datos ingresados por el usuario.

#include <iostream> // Incluye la biblioteca de entrada y salida estándar

int main() {

int edad;

std::string nombre;

// Solicitar el nombre del usuario

std::cout << "Por favor, ingresa tu nombre: ";

std::cin >> nombre;

// Solicitar la edad del usuario

std::cout << "Hola, " << nombre << "! Por favor, ingresa tu edad: ";

std::cin >> edad;

// Mostrar la información ingresada

std::cout << "Tu nombre es " << nombre << " y tienes " << edad << " años." << std::endl;

return 0;

}

**Operadores lógicos y aritméticos en C++**

Los operadores lógicos y aritméticos en C++ permiten realizar operaciones y comparaciones entre valores.

Operadores aritméticos + Suma, - Resta, \* Multiplicación, / División de enteros, % Resto de la división entera.

Operadores lógicos && Operador lógico AND, == Operador de igualdad, != Operador de desigualdad, <= Operador menor o igual que, >= Operador mayor o igual que.

**NOT lógico (!)**: Invierte el valor lógico del operando.  
resultado = !(a > 0); // resultado = false

**AND lógico (&&)**: Devuelve true si ambos operandos son verdaderos.

bool resultado = (a > 0) && (b > 0); // resultado = true

**OR lógico (||)**: Devuelve true si al menos uno de los operandos es verdadero.

resultado = (a > 0) || (b < 0); // resultado = true

Operadores de asignación combinados /= División y asignación, %= Módulo y asignación, &= Asignación e “y” a nivel de bit, <<= Desplazamiento bit a bit a la izquierda y asignación.

**AND lógico (&&)**: Devuelve true si ambos operandos son verdaderos.

bool resultado = (a > 0) && (b > 0); // resultado = true

**OR lógico (||)**: Devuelve true si al menos uno de los operandos es verdadero.

resultado = (a > 0) || (b < 0); // resultado = true

**NOT lógico (!)**: Invierte el valor lógico del operando.

resultado = !(a > 0); // resultado = false

Orden de evaluación

Cuando se combinan diferentes operadores, el orden de evaluación se sigue de izquierda a derecha. Para determinar con claridad qué operaciones se desean realizar en primer lugar, se pueden usar paréntesis.

**Investigar la estructura de selección**

Las estructuras de selección en C++ son if, switch, if-else if-else y if-else. Permiten ejecutar secciones de código de forma condicional.

1. If

La estructura if evalúa una condición y ejecuta un bloque de código si la condición es verdadera.

#include <iostream>

int main()

{ int numero = 10;

if (numero > 5) { std::cout << "El número es mayor que 5." << std::endl;

}

return 0;

}

1. **if-else**

La estructura if-else evalúa una condición y ejecuta un bloque de código si la condición es verdadera; de lo contrario, ejecuta otro bloque de código.

#include <iostream>

int main() {

int numero = 3;

if (numero > 5) {

std::cout << "El número es mayor que 5." << std::endl;

} else {

std::cout << "El número es 5 o menor." << std::endl;

}

return 0;

}

1. **if-else if-else**

Esta estructura permite evaluar múltiples condiciones en secuencia.

#include <iostream>

int main() {

int numero = 7; if (numero > 10) {

std::cout << "El número es mayor que 10." << std::endl;

} else if (numero > 5) {

std::cout << "El número es mayor que 5 pero menor o igual a 10." << std::endl;

} else {

std::cout << "El número es 5 o menor." << std::endl;

}

return 0;

}

1. **switch**

La estructura switch selecciona una de entre múltiples opciones basándose en el valor de una expresión.

#include <iostream>

int main() {

int opcion = 2;

switch (opcion) {

case 1:

std::cout << "Opción 1 seleccionada." << std::endl; break;

case 2:

std::cout << "Opción 2 seleccionada." << std::endl; break;

case 3:

std::cout << "Opción 3 seleccionada." << std::endl; break;

default:

std::cout << "Opción no válida." << std::endl; break;

}

return 0;

}