UNIVERSIDAD MARIANO GÁLVEZ

INGENIERÍA EN SISTEMAS DE INFORMACIÓN Y CIENCIAS DE LA COMPUTACIÓN

Alumnos: Víctor Fernando Minas Salazar, Robert Alexander Cardona Berg.

Número de carné: 1290-22-12396, 1290-22-19999.

Sección: A.

Catedrático: ING. Augusto Armando Cardona Paiz.

Fecha de entrega: Domingo 28 de mayo.

Curso: Programación 1.

TAREA FINAL

Unidad I – Conceptos Básicos

Lenguaje C++

El lenguaje de programación C++ es un lenguaje informático de alto nivel, que se utiliza ampliamente en el desarrollo de software de alta calidad. Es una extensión del lenguaje de programación C, y se caracteriza por su capacidad de programación orientada a objetos, su eficiencia y su flexibilidad.

El lenguaje C++ se ha convertido en uno de los lenguajes de programación más populares y conocidos en el mundo de la informática. Fue creado en los años 80 por Bjarne Stroustrup, quien estaba buscando una manera de mejorar y ampliar el lenguaje C.

C++ es un lenguaje compilado, lo que significa que se traduce directamente a código de máquina antes de ser ejecutado. Esta característica le otorga a C++ tiempos de ejecución más rápidos y una mayor eficiencia en comparación con otros lenguajes interpretados.

La programación orientada a objetos es una de las características más notables de C++. Permite a los programadores crear objetos y clases que encapsulan datos y comportamientos para su reutilización en diferentes partes de un programa. Además, C++ también permite la programación genérica, brindando una amplia variedad de opciones para la creación de plantillas de código que pueden ser reutilizables.

C++ es utilizado en una variedad de aplicaciones informáticas, como el desarrollo de videojuegos, aplicaciones de escritorio y software de sistemas operativos. También es ampliamente utilizado en la programación de sistemas embebidos y dispositivos electrónicos.

Una de las desventajas de C++ es que la escritura de código no es tan simple comparada con otros lenguajes de programación de alto nivel, lo que puede resultar complicado para programadores novatos. Por otro lado, el lenguaje de programación C++ brinda una mayor flexibilidad al programador, permitiéndole controlar el código a nivel de sistema operativo.

En conclusión, el lenguaje de programación C++ es un lenguaje potente y versátil que ha demostrado su eficacia en la programación de sistemas operativos, videojuegos y una amplia variedad de aplicaciones informáticas. Aunque puede ser complicado para los principiantes, C++ es una herramienta esencial para aquellos programadores que buscan un mayor control sobre su código.

- ### Conceptos Básicos:

C++ es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de software y juegos. Fue diseñado por Bjarne Stroustrup en 1983 como una extensión del lenguaje de programación C. La versión actual de C++ es C++20.

C++ es un lenguaje orientado a objetos, lo que significa que permite crear objetos que tienen propiedades y comportamientos definidos. Los objetos se crean mediante la definición de clases, que son plantillas que especifican las propiedades y comportamientos de los objetos. Una clase puede contener atributos, que son variables que almacenan información sobre un objeto, y métodos, que son funciones que se usan para manipular esos atributos.

C++ también admite la programación genérica, que es una técnica que permite crear código que puede adaptarse a diferentes tipos de datos. La programación genérica se logra mediante el uso de plantillas, que son bloques de código que se pueden utilizar para generar diferentes versiones de una función o clase.

Una característica importante de C++ es la gestión manual de la memoria, lo que significa que el programador debe llevar un seguimiento de cuándo se asigna y libera la memoria utilizada por una aplicación. Esto se logra

mediante el uso de punteros, que son variables que contienen direcciones de memoria. Los punteros se usan para almacenar y acceder a los objetos en la memoria.

C++ también ofrece una amplia biblioteca estándar que proporciona funciones y clases para la entrada/salida de datos, manipulación de cadenas, matemáticas, manejo de archivos y mucho más. La biblioteca estándar es independiente de la plataforma y se puede usar en diferentes sistemas operativos.

Otras características de C++ incluyen la sobrecarga de operadores, que permite definir el comportamiento de operadores como + y *, y la herencia, que permite que una clase herede las propiedades y comportamientos de otra clase. C++ también admite la programación concurrente y el paralelismo, lo que significa que se pueden crear aplicaciones que ejecuten varias tareas simultáneamente.

En resumen, los conceptos básicos de C++ incluyen la programación orientada a objetos, la programación genérica, la gestión manual de memoria, la biblioteca estándar, la sobrecarga de operadores, la herencia y la programación concurrente y el paralelismo. C++ es un lenguaje potente y versátil que se utiliza en una amplia gama de aplicaciones, desde el desarrollo de aplicaciones empresariales hasta juegos y aplicaciones en tiempo real.

- **Herramientas de Desarrollo para C++:**

- Visual Studio: Es el entorno de desarrollo integrado (IDE) más popular para programar en C++. Incluye un compilador de C++ y un depurador y cuenta con herramientas integradas para la administración de proyectos, edición de código, y depuración.

- Eclipse CDT: Esta es una plataforma de desarrollo basada en Eclipse para C++ que ofrece un completo conjunto de herramientas para programar en C++. El IDE incluye una potente herramienta de depuración, una herramienta de análisis de código y una amplia variedad de plugins para mejorar su funcionalidad.

- Qt Creator: Es un IDE de código abierto para desarrolladores C++ que proporciona un conjunto de herramientas y funcionalidades para el diseño de interfaces de usuario, el desarrollo de aplicaciones y la prueba de aplicaciones. También ofrece soporte para diferentes plataformas, incluyendo Windows, Linux y Mac.

- CodeBlocks: Es una IDE de código abierto que está diseñada para ser especialmente útil para los desarrolladores C++ y es compatible con múltiples plataformas. Ofrece una interfaz de usuario fácil de usar y muchas herramientas integradas para la creación de aplicaciones en C++.

- Dev-C++: Es un IDE de código abierto que funciona para sistemas operativos Windows. Esta herramienta proporciona una amplia variedad de herramientas para el desarrollo en C++ incluyendo depurador, editor, y soporte para múltiples compiladores.

- CLion: Es un potente IDE para programar en C++ que incluye un avanzado depurador, una completa herramienta de análisis de código y una interfaz de usuario personalizable.

- Xcode: Es un IDE desarrollado por Apple que es compatible con múltiples lenguajes de programación, incluyendo C++. Es un ambiente sencillo e intuitivo que incluye herramientas para el manejo de proyectos y una herramienta de depuración.
- NetBeans: Es una plataforma de desarrollo de código abierto que es compatible con múltiples lenguajes de programación, incluyendo C++. Algunas de sus herramientas incluyen un editor de código, compilador, preferencias de formato, y una interfaz de usuario personalizable.

Editores Para C++:

1. Code::Blocks.
2. Visual Studio Code.
3. Eclipse.
4. Atom.
5. Sublime Text.
6. NetBeans.
7. Dev-C++.
8. Notepad++.
9. CodeLite.
10. Xcode (para MacOS y iOS).

Tipos de Datos Nativos en C++:

Los tipos de datos básicos que se encuentran en C++, como enteros, flotantes, caracteres, booleanos, etc.

- bool: este tipo de datos almacena valores booleanos, es decir, verdadero o falso (true o false).
- char: el tipo de datos char almacena caracteres individuales, como letras, números y símbolos.
- int: el tipo de datos int almacena números enteros, tanto positivos como negativos.
- float: este tipo de datos almacena números con coma flotante, lo que significa que puede almacenar números decimales.
- double: este tipo de datos también almacena números con coma flotante, pero con mayor precisión que el tipo de datos float.
- void: el tipo de datos void se utiliza para declarar funciones que no devuelven ningún valor.
- wchar_t: este tipo de datos se utiliza para almacenar caracteres ampliados y puede ocupar más de un byte.

En resumen, los tipos de datos nativos en C++ son utilizados para representar diferentes tipos de datos y permiten que el código se ejecute de manera eficiente.

- **Bibliotecas en C++:**

- iostream: Biblioteca estándar para entrada/salida de datos en C++. Contiene la definición de las clases cin y cout para leer y mostrar datos, respectivamente.

- string: Biblioteca para trabajar con cadenas de caracteres en C++. Define la clase string para crear y manipular strings.

- vector: Biblioteca para trabajar con arrays dinámicos en C++. Define la clase vector para crear y manipular arrays dinámicos.

- algorithm: Biblioteca para realizar operaciones sobre contenedores en C++. Incluye funciones para ordenar, buscar elementos, etc.

- map: Biblioteca para trabajar con mapas en C++. Define la clase map para crear y manipular mapas.

- set: Biblioteca para trabajar con conjuntos en C++. Define la clase set para crear y manipular conjuntos.

- stack: Biblioteca para trabajar con pilas en C++. Define la clase stack para crear y manipular pilas.

- queue: Biblioteca para trabajar con colas en C++. Define la clase queue para crear y manipular colas.

- deque: Biblioteca para trabajar con arrays doblemente enlazados en C++. Define la clase deque para crear y manipular arrays doblemente enlazados.

- iterator: Biblioteca para crear y manipular iteradores en C++. Define la clase iterator para trabajar con contenedores de forma genérica.

- **Entrada y Salida de Flujos:**

En programación, la entrada y salida de flujos se refiere a la forma en que los datos se transfieren entre un programa y su entorno externo, ya sea en un archivo o en la pantalla. La entrada de flujo es la transferencia de datos desde el entorno externo al programa, mientras que la salida de flujo es la transferencia de datos desde el programa al entorno externo.

En C++, existen varias funciones que facilitan la entrada y salida de flujos. Algunas de las funciones de entrada de flujo más comunes son:

cin: esta función se utiliza para leer datos desde la entrada estándar, que por defecto es el teclado. Por ejemplo, si queremos leer un número entero, podemos usar la siguiente función: cin >> num;

getline: esta función se utiliza para leer una línea completa de entrada, separada por el carácter de salto de línea. Por ejemplo, si queremos leer una cadena de texto, podemos usar la siguiente función: getline(cin, str);

Por otro lado, algunas de las funciones de salida de flujo más comunes son:

cout: esta función se utiliza para imprimir datos en la salida estándar, que por defecto es la pantalla. Por ejemplo, si queremos imprimir un mensaje, podemos usar la siguiente función: cout << "Hola Mundo";

cerr: esta función se utiliza para imprimir mensajes de error en la salida estándar de error. Por ejemplo, si queremos imprimir un mensaje de error, podemos usar la siguiente función: cerr << "Error: no se puede abrir el archivo.

- ## Programación Estructurada:

La programación estructurada es un paradigma de programación que busca mejorar la claridad, calidad y eficiencia de los programas mediante la adopción de un conjunto de técnicas y principios que estructuran el flujo de control y el diseño del programa.

Los principales principios de la programación estructurada son la idea de dividir el programa en módulos o bloques de código, llamados funciones o procedimientos, que realizan tareas específicas y están diseñados para interactuar con otros bloques del programa mediante una interfaz claramente definida. También se enfatiza en la estructuración del control de flujo del programa mediante el uso de estructuras de control como la secuencia, la selección y la iteración en lugar de saltos incondicionales y estructuras de control complejas.

La programación estructurada promueve el uso de una notación clara y consistente, el uso de nombres de variables significativos, la simplificación de las estructuras de decisión y el modularidad del programa. Esto permite que el código sea más fácil de entender, depurar y mantener, y contribuye a una programación más eficiente y confiable.

Entre las herramientas de programación utilizadas para implementar la programación estructurada se encuentran los lenguajes de programación estructurados, como C, Pascal y Fortran, y las técnicas de diseño estructurado de programas, como el diagrama de flujo, el pseudocódigo y el algoritmo estructurado.

La programación estructurada ha tenido un gran impacto en la programación moderna y ha influido en muchos otros paradigmas de programación, como la programación orientada a objetos y la programación funcional. Su enfoque en la claridad, simplicidad y modularidad del código sigue siendo una de las principales preocupaciones en la programación de hoy en día.

Características:

La programación estructurada es un estilo de programación que se enfoca en el uso de estructuras de control de flujo para construir programas claros, eficientes y fáciles de mantener. Las características principales de la programación estructurada son:

1. Modularidad: separación de funciones en módulos independientes y bien definidos, con interfaces claras y documentadas.

2. Estructuras de control de flujo: uso de estructuras de control de flujo (condicionales, bucles, etc.) para construir algoritmos de manera clara y eficiente.

3. Clasificación: utilización de tipos de datos y estructuras de datos para clasificar la información de manera organizada y accesible.

4. Énfasis en funciones: la programación estructurada se enfoca en la creación de funciones, que son bloques autónomos de código que realizan tareas específicas.

5. Enfoque en el algoritmo: la programación estructurada se centra en la creación de algoritmos eficientes para resolver problemas.

6. Legibilidad: la programación estructurada enfatiza la legibilidad del código para facilitar su comprensión y mantenimiento.

7. Modularidad: los programas se dividen en módulos que se pueden desarrollar, probar y mantener de manera independiente.

En resumen, la programación estructurada es un enfoque sistemático y disciplinado para la construcción de software que enfatiza el uso de estructuras de control de flujo, modularidad y eficiencia.

- **Programación Orientada a Objetos:**

La programación orientada a objetos (POO) es un paradigma de programación que se basa en el concepto de "objetos", los cuales son entidades que combinan datos y funciones relacionadas en una sola unidad. En lugar de enfocarse en los procedimientos y las instrucciones, la POO se centra en la interacción entre objetos, permitiendo la reutilización de código y la organización modular del programa.

La POO se basa en cuatro pilares fundamentales: encapsulamiento, herencia, polimorfismo y abstracción.

1. El encapsulamiento se refiere a la capacidad de ocultar los detalles internos de un objeto y exponer solo una interfaz para interactuar con él. Esto se logra mediante la definición de clases, que son plantillas o moldes que describen la estructura y el comportamiento de los objetos. Las clases definen los atributos (variables) y los métodos (funciones) que pueden ser utilizados por los objetos creados a partir de ellas.

2. La herencia permite la creación de nuevas clases basadas en clases existentes. Una clase derivada hereda los atributos y métodos de la clase base, lo que facilita la reutilización de código y la organización jerárquica de las clases. Además, la herencia permite agregar nuevos atributos y métodos a la clase derivada, lo que proporciona flexibilidad y extensibilidad al diseño del programa.

3. El polimorfismo se refiere a la capacidad de objetos de diferentes clases de responder de manera distinta a la misma llamada de método. Esto se logra mediante la definición de métodos con el mismo nombre en diferentes clases y la capacidad de invocarlos a través de una referencia de tipo más general. El polimorfismo permite tratar a objetos de diferentes clases de manera uniforme, lo que facilita la flexibilidad y la extensibilidad del código.

4. La abstracción es el proceso de identificar las características esenciales de un objeto y omitir los detalles irrelevantes. En la POO, la abstracción se logra mediante la creación de clases y la definición de métodos abstractos, que son métodos sin implementación. Las clases abstractas proporcionan una base común para las clases derivadas y establecen un contrato que define los métodos que deben implementar. La abstracción permite el modularidad y el diseño de programas más flexibles y mantenibles.

Además de estos pilares, la POO también se basa en otros conceptos, como la asociación, la composición, la agregación y la sobrecarga de operadores, que permiten modelar relaciones más complejas entre objetos y mejorar la expresividad del código.

En resumen, la programación orientada a objetos es un enfoque de programación que se centra en los objetos, su interacción y su reutilización. La POO se basa en la encapsulación, la herencia, el polimorfismo y la abstracción, y proporciona un marco para desarrollar programas más estructurados, flexibles y fáciles de mantener.

Conceptos Básicos de la POO

1. **Clase:** es una estructura que define las propiedades y comportamientos de un objeto. Es un modelo o plantilla que crea instancias de objetos.

2. **Objeto:** es una instancia de una clase en la POO. Representa una entidad con características y comportamientos definidos por la clase.

3. **Pilares:** son cuatro pilares fundamentales que definen un esquema en el desarrollo de software. Estos cuatro pilares son la abstracción, encapsulamiento, polimorfismo y herencia.

4. **Instanciación:** es el proceso de crear un objeto a partir de una clase. Con esto, se asigna memoria y se inicializan los atributos del objeto.

5. **Métodos:** son funciones o procedimientos que realiza una clase u objeto. Representa las acciones o comportamientos que pueden realizar los objetos de una clase.

6. **Atributos:** son las variables que almacenan los datos asociados a una clase u objeto. Representan las características o propiedades de este.

7. **Constructor:** es un método especial que se llama al crear un objeto de una clase. Normalmente se utiliza para inicializar los atributos y realizar tareas iniciales.

8. **Destructor:** es un método especial que se llama al eliminar un objeto de una clase. Es utilizado para liberar recursos y realizar tareas finales antes de la eliminación del objeto.

Clase y super clase

Dentro de C++, existen diferentes tipos de clase, los cuales son:

- **Clase:** es una plantilla o estructura que define las propiedades y comportamientos de un objeto. Representa un concepto o entidad que encapsula datos y funciones relacionadas.
- **Superclase:** es una clase de la cual se derivan otras clases. Este tipo de clase proporciona una base común de atributos y métodos que heredan sus clases derivadas. También es llamada “clase base” o “clase padre”.
- **Clase derivada:** es una clase a la cual derivan todos los métodos y atributos que provienen de una clase base. También es llamada “clase hija”.

Sobrecarga de operadores

La sobrecarga de operadores es una característica en C++ que permite definir el comportamiento de los operadores existentes para tipos de datos personalizados, tales como clases y estructuras dentro del código en C++.

En la sobrecarga de operadores interviene el operador de sobrecarga, el cual es una función que se utiliza para definir el comportamiento de un operador específico. Como ejemplo de algunos operadores sobrecargables podemos mencionar:

- **Aritméticos:** incluyen los operadores **+, -, * y /.**
- **De asignación:** incluye el operador **=.**
- **Relacionales:** incluyen los operadores **<, > y = =.**
- **Lógicos:** incluyen los operadores **&& y ||.**

Propiedades y métodos de una clase

En una clase de C++, pueden intervenir diversos conceptos que ayudan a estructurar la programación. Estos conceptos complementan cada una de las clases y le proveen de información adicional para los objetos en esa misma clase. Estos dos conceptos son las propiedades y los métodos de una clase.

- **Propiedades:** son también llamados "atributos", y son variables que pertenecen a una clase y almacenan datos relacionados a los objetos de esa misma clase.

- **Métodos:** son funciones definidas dentro de la clase y realizan operaciones específicas relacionadas con los objetos de la clase. Representan su comportamiento dentro del programa.

También es de gran importancia que dentro de estos dos conceptos también pueden intervenir los cuatro pilares de la POO.

Creación de objetos

Dentro de C++, la creación de objetos es el proceso de instanciar una clase para crear un objeto en memoria. Para este proceso, se realiza utilizando la palabra clave "new" seguida del nombre de la clase y los paréntesis para invocar el constructor.

Durante este proceso se utilizan diversos conceptos de la POO, además de sus pilares y algunas otras como los constructores, destructores, asignación de objetos, copia de objetos, punteros, entre otros.

Constructores

Los constructores en C++ son métodos especiales que forman parte de las clases en C++ donde son utilizados para inicializar los objetos de esa clase. Generalmente se llaman de manera automática al crear un objeto.

Existen algunos tipos de constructores, los cuales son:

- **Constructor por defecto:** es el constructor que no posee parámetros o cuyos parámetros poseen valores predeterminados.
- **Constructor con parámetros:** es el constructor que acepta uno o más argumentos para inicializar los atributos del objeto.
- **Constructor de copia:** es un constructor especial que crea un objeto nuevo basado en otro objeto existente de la misma clase.
- **Constructor explícito:** es un constructor que se utiliza para evitar conversiones implícitas y de esa manera limitar la forma en que se puede construir un objeto.
- **Constructor por omisión y eliminados:** son constructores especiales que se definen para controlar el comportamiento de la creación de objetos.
- **Constructor en herencia:** sucede en la herencia de clases, cuando los constructores de la clase base se llaman automáticamente al crear un objeto de la case derivada.

Unidad II

Operadores Lógicos

Los operadores lógicos en C++ conforman una parte importante en la implementación del código en C++. Son utilizados para realizar operaciones de lógica booleana en expresiones condicionales, permitiendo combinar y evaluar condiciones booleanas (verdadero o falso). Entre los operadores lógicos en C++ encontramos:

1. **Operador lógico AND (&&):** evalúa dos condiciones booleanas y devuelve *verdadero* si ambas condiciones son verdaderas. De lo contrario devuelve *falso*.
2. **Operador lógico OR (||):** evalúa dos condiciones booleanas y devuelve *verdadero* si al menos una de las condiciones es verdadera. Solo devuelve *falso* si ambas condiciones son falsas.
3. **Operador lógico NOT (!):** es un operador que invierte el valor de una condición booleana. Si la condición es *verdadera*, devuelve *falso* y viceversa.

- ## Estructuras de Control:

Las estructuras de control son construcciones en lenguajes de programación que permiten controlar el flujo de ejecución de un programa. Estas estructuras permiten tomar decisiones y repetir acciones según ciertas condiciones. A continuación, se presentan las principales estructuras de control y sus definiciones:

1. Estructura de control condicional (if-else): Permite ejecutar un bloque de código si se cumple una condición, y otro bloque de código si no se cumple esa condición. La condición se evalúa como verdadera o falsa y determina qué bloque se ejecutará.

2. Estructura de control iterativa (bucles o loops):

- Bucle while: Permite repetir un bloque de código mientras se cumpla una condición. La condición se verifica antes de cada repetición.
- Bucle do-while: Similar al bucle while, pero la condición se verifica después de cada repetición, por lo que el bloque de código se ejecuta al menos una vez.
- Bucle for: Permite repetir un bloque de código un número específico de veces. Se especifica una variable de control, una condición de finalización y un incremento o decremento para la variable en cada repetición.

3. Estructura de control de selección múltiple (switch-case): Permite seleccionar un bloque de código para ejecutar entre varios posibles casos, según el valor de una expresión. Cada caso representa un valor posible de la expresión y se ejecutará el bloque correspondiente a ese valor.

Estas estructuras de control permiten que un programa tome decisiones y se ajuste dinámicamente según las condiciones o requerimientos del problema a resolver. Al combinar estas estructuras de control, se pueden construir algoritmos más complejos y poderosos. Es importante comprender y utilizar adecuadamente estas estructuras para controlar el flujo de ejecución de los programas de manera eficiente y lógica

If, If Anidados, For, While, Switch:

Las estructuras de control son construcciones en lenguajes de programación que permiten controlar el flujo de ejecución de un programa. Estas estructuras permiten tomar decisiones, repetir acciones y seleccionar entre varias opciones. A continuación, se presentan las definiciones de algunas estructuras de control comunes:

- Estructura de control condicional (if): Permite ejecutar un bloque de código si se cumple una condición específica. Si la condición es verdadera, se ejecuta el bloque asociado al if; de lo contrario, se omite y continúa con la ejecución del programa.

- Estructuras de control condicional anidadas (if anidados): Consisten en la utilización de múltiples bloques if dentro de otros bloques if. Esto permite tomar decisiones más complejas y anidadas, donde cada bloque if evalúa una condición específica y ejecuta su correspondiente bloque de código si la condición es verdadera.

- Estructura de control iterativa (for): Permite repetir un bloque de código un número específico de veces. Se utiliza un contador para controlar el número de repeticiones, y se especifica una condición de finalización. En cada repetición, se ejecuta el bloque de código y se actualiza el valor del contador.

- Estructura de control iterativa (while): Permite repetir un bloque de código mientras se cumpla una condición específica. La condición se verifica antes de cada repetición. Si la condición es verdadera, se ejecuta el bloque de código; de lo contrario, se finaliza la iteración.

- Estructura de control de selección múltiple (switch): Permite seleccionar una opción de varias posibles, según el valor de una expresión. Se evalúa la expresión y se compara con los distintos casos especificados en el switch. Si se encuentra una coincidencia, se ejecuta el bloque de código asociado a ese caso. Es común utilizar las instrucciones "break" y "continue" dentro de los casos para controlar el flujo de ejecución dentro del switch.

Estas estructuras de control son fundamentales en la programación, ya que permiten construir programas flexibles y adaptables. Cada estructura tiene su propia sintaxis y forma de uso, por lo que es importante comprender y utilizar correctamente cada una de ellas para controlar el flujo de ejecución de un programa de manera efectiva.

Diferencia entre operadores de igualdad y de asignación:

Los operadores de igualdad y de asignación son dos tipos de operadores que se utilizan en la programación, pero tienen propósitos diferentes. A continuación, se explica la diferencia entre ellos:

- Operador de igualdad (==): Este operador se utiliza para comparar dos valores y determinar si son iguales. Devuelve un valor booleano, es decir, verdadero (true) si los valores son iguales, y falso (false) si son diferentes. La comparación se realiza únicamente en base al valor de los operandos, sin considerar su tipo de dato.

Ejemplo:

```
x = 5
y = 7
if x == y:
    print("Los valores son iguales")
else:
    print("Los valores son diferentes")
```

En este ejemplo, la condición x == y evalúa si los valores de x y y son iguales. Si se cumple la condición, se imprimirá "Los valores son iguales".

- Operador de asignación (=): Este operador se utiliza para asignar un valor a una variable. Asigna el valor de la expresión o variable del lado derecho al nombre de la variable del lado izquierdo. Es importante destacar que el operador de asignación no compara valores, sino que realiza una asignación de valor.

Ejemplo:

```
x = 5
y = x
print(y)
```

En este ejemplo, el valor de x (que es 5) se asigna a la variable y mediante el operador de asignación (=). Luego, se imprime el valor de y, que será 5.

En resumen, la diferencia principal entre los operadores de igualdad (==) y de asignación (=) radica en su función. El operador de igualdad se utiliza para comparar dos valores y determinar si son iguales, mientras que el operador de asignación se utiliza para asignar un valor a una variable. Es importante utilizar el operador correcto según el objetivo que se desea lograr en el programa.

- ## Recursividad:

La recursividad es un concepto en programación que se refiere a la capacidad de una función para llamarse a sí misma de forma directa o indirecta. En otras palabras, una función recursiva es aquella que se puede definir en términos de sí misma.

Al utilizar la recursividad, una función puede dividir un problema en subproblemas más pequeños y resolverlos de manera recursiva hasta llegar a un caso base que no requiere más llamadas recursivas. Cada llamada recursiva resuelve un subproblema más pequeño, y los resultados se combinan para obtener la solución del problema original.

La estructura básica de una función recursiva consta de dos partes principales:

- Caso base: Es la condición que indica cuándo se debe detener la recursividad y retornar un resultado concreto. Es fundamental definir correctamente el caso base para evitar que la recursividad se ejecute infinitamente.

- Caso recursivo: Es la parte en la que se invoca la función recursivamente, generalmente con argumentos modificados o más pequeños. Esta llamada a la función se repite hasta que se alcance el caso base.

A continuación, se muestra un ejemplo de una función recursiva que calcula la factorial de un número:

```python
def factorial(n):
    if n == 0:  # Caso base: factorial de 0 es 1
        return 1
    else:
        return n * factorial(n-1)  # Caso recursivo: n * factorial(n-1)

resultado = factorial(5)
print(resultado)  # Imprime 120
```

En este ejemplo, la función factorial se llama a sí misma en el caso recursivo, multiplicando el número actual (n) por la factorial del número anterior (n-1). La recursividad continúa hasta que se alcance el caso base, que es cuando n es igual a 0, en cuyo caso se devuelve 1.

Es importante tener en cuenta que la recursividad puede ser una técnica poderosa, pero también es necesario tener cuidado al utilizarla. Es fundamental asegurarse de que se cumpla el caso base y que las llamadas recursivas converjan hacia él. Además, una implementación ineficiente o incorrecta de la recursividad puede llevar a problemas de rendimiento o a una ejecución infinita del programa.

- ## Manejo de Excepciones:

El manejo de excepciones es un mecanismo en la programación que permite controlar y responder adecuadamente a situaciones excepcionales o errores durante la ejecución de un programa. Una excepción es un objeto que representa un error o una condición anómala que ocurre durante la ejecución de un programa.

El manejo de excepciones se basa en bloques de código llamados "try-catch" o "try-except", donde se coloca el código que puede generar una excepción dentro del bloque "try", y las respuestas a las excepciones se definen en los bloques "catch" o "except". A continuación, se presenta un ejemplo básico del manejo de excepciones:

Ejemplo:

```
try:
    # Código que puede generar una excepción
    resultado = dividir(10, 0)  # Intento dividir 10 entre 0
    print(resultado)
except ZeroDivisionError:
    # Manejo de la excepción específica (división entre cero)
    print("Error: No se puede dividir entre cero")
```

En este ejemplo, el código que realiza la división se coloca dentro del bloque "try". Si ocurre una excepción de tipo "ZeroDivisionError" (división entre cero), se ejecuta el bloque "except" correspondiente, donde se define cómo manejar la excepción. En este caso, simplemente se imprime un mensaje de error.

El manejo de excepciones permite capturar y tratar diferentes tipos de excepciones de manera específica, lo que permite al programa recuperarse de errores y seguir ejecutándose en lugar de detenerse abruptamente. Además, es posible utilizar bloques "except" adicionales para manejar diferentes tipos de excepciones y proporcionar respuestas específicas para cada una.

Además de los bloques "try" y "except", también se puede utilizar un bloque "finally" opcional para definir un código que siempre se ejecutará, independientemente de si ocurre una excepción o no. Esto es útil para liberar recursos o realizar tareas de limpieza.

El manejo de excepciones es una práctica importante para mejorar la robustez y la confiabilidad de los programas, ya que permite anticiparse y manejar situaciones excepcionales de manera controlada en lugar de que el programa falle o produzca resultados incorrectos.

- ## Procesamiento de Archivos:

El procesamiento de archivos es una tarea común en la programación, que implica leer y escribir datos en archivos almacenados en el sistema de archivos de la computadora. Los archivos pueden contener información estructurada o sin formato, como texto, números, imágenes, etc. A continuación, se presentan los conceptos básicos y las operaciones principales del procesamiento de archivos:

- Apertura de archivos: Para trabajar con un archivo, primero debe abrirse. La apertura de un archivo crea una conexión entre el programa y el archivo en el sistema de archivos. Durante la apertura, se especifica el modo de acceso al archivo, que puede ser lectura, escritura o ambos.

- Lectura de archivos: Una vez que un archivo está abierto para lectura, se puede acceder a su contenido. Las operaciones de lectura permiten recuperar datos del archivo, ya sea en su totalidad o en partes. Algunas funciones comunes de lectura incluyen leer una línea de texto, leer un carácter, leer una cantidad específica de bytes o leer todo el contenido del archivo.

- Escritura en archivos: Cuando un archivo está abierto para escritura, se puede agregar contenido al archivo. Las operaciones de escritura permiten guardar datos en el archivo. Se pueden escribir líneas de texto, valores numéricos o cualquier otro tipo de datos admitido por el lenguaje de programación.

- Cierre de archivos: Después de finalizar las operaciones en un archivo, es importante cerrarlo. El cierre de un archivo libera los recursos asociados y asegura que los datos se guarden correctamente. Además, cerrar un archivo permite que otros programas accedan a él.

El procesamiento de archivos también puede involucrar otras operaciones adicionales, como posicionamiento en un archivo, búsqueda y manipulación de la posición del puntero de lectura/escritura, y manejo de errores al abrir o escribir en un archivo.

Es importante tener en cuenta que al trabajar con archivos, se deben seguir prácticas adecuadas de manejo de errores y garantizar que se realicen las operaciones de apertura, lectura y escritura de manera segura y eficiente. Además, se deben considerar aspectos de seguridad, como la validación de la entrada del usuario y la protección de los archivos contra accesos no autorizados.

El procesamiento de archivos es una herramienta poderosa para almacenar y recuperar información persistente. Se utiliza en una amplia variedad de aplicaciones, desde el procesamiento de datos hasta la gestión de archivos multimedia y la manipulación de configuraciones y archivos de texto.

- **Archivos y Flujos:**

Los archivos y los flujos son conceptos relacionados en el procesamiento de datos y se utilizan para leer y escribir información en diferentes medios de almacenamiento, como discos duros, unidades de almacenamiento externas, redes, etc. A continuación, se explica cada concepto:

- Archivos: Un archivo es una unidad lógica de información almacenada en un medio de almacenamiento. Puede contener cualquier tipo de datos, como texto, números, imágenes, audio, etc. Los archivos se organizan en una estructura de directorios y subdirectorios en el sistema de archivos de la computadora. Cada archivo tiene un nombre único y una extensión que indica su tipo de contenido. Los archivos pueden ser creados, leídos, modificados y eliminados mediante operaciones de entrada/salida.

- Flujos: Un flujo (stream) es una secuencia de datos que fluye de una fuente a un destino. Los flujos se utilizan para leer y escribir datos desde y hacia archivos u otros dispositivos de entrada/salida, como la consola o una conexión de red. Un flujo se considera una abstracción que permite el procesamiento de datos de manera uniforme, independientemente de su origen o destino.

Existen dos tipos principales de flujos:

- Flujo de entrada (input stream): Proporciona datos desde una fuente hacia el programa. Se utiliza para leer datos de archivos o dispositivos externos. Por ejemplo, se puede leer un archivo de texto línea por línea o leer bytes de un archivo binario.

- Flujo de salida (output stream): Recibe datos desde el programa y los envía a un destino. Se utiliza para escribir datos en archivos o enviar datos a dispositivos externos. Por ejemplo, se puede escribir texto en un archivo o enviar bytes a través de una conexión de red.

Los flujos se pueden asociar con archivos específicos, como flujos de archivo, o con otros medios de entrada/salida, como flujos de consola o flujos de red. Los flujos proporcionan métodos y funciones para realizar operaciones de lectura y escritura de datos de manera eficiente y conveniente.

En resumen, los archivos representan la unidad lógica de almacenamiento en el sistema de archivos, mientras que los flujos son una abstracción que permite la lectura y escritura de datos desde y hacia archivos y otros

dispositivos de entrada/salida. Los flujos se utilizan para manipular la información contenida en los archivos y realizar operaciones de entrada/salida de manera uniforme.

UNIDAD 3

Arreglos y vectores

Los arreglos y vectores son estructuras de datos en C++ que permiten almacenar múltiples valores del mismo tipo.

Los arreglos (también llamados arrays) se utilizan para almacenar varios valores en una sola variable, en lugar de declarar variables separadas para cada valor. Para declararlo, hay que definir el nombre de la matriz seguido de corchetes y especificar especificar el número de elementos que debe almacenar.

Los vectores son un tipo de arreglo de una sola dimensión y forman parte de la amplia variedad de estructuras de datos que ofrece C++.

Declaración y creación de arreglos

Los arreglos se declaran indicando el tipo de dato y el nombre del arreglo. Se crean asignando valores a cada elemento utilizando índices. Para esto, se define el tipo de variable seguido de corchetes y el número de elementos a almacenar.

Hay que tomar en cuenta que el primer elemento del arreglo tiene índice 0 y el último tiene índice (tamaño – 1).

Ejemplos del uso de arreglos

Los arreglos se utilizan para almacenar y manipular conjuntos de datos. Pueden utilizarse en algoritmos de búsqueda, ordenamiento, procesamiento de imágenes, entre otros. Estos elementos se acceden mediante su índice. Un ejemplo en C++ de un arreglo sencillo puede ser:

```
#include <iostream>

using namespace std;

int main() {
   // Declaración y creación de un arreglo de enteros
   int numeros[5] = {2, 4, 6, 8, 10};

   // Acceso a elementos del arreglo
   cout << "Primer elemento: " << numeros[0] << endl;
   cout << "Tercer elemento: " << numeros[2] << endl;

   // Modificación de un elemento del arreglo
   numeros[1] = 12;
```

```
    cout << "Segundo elemento modificado: " << numeros[1] << endl;

    // Recorrido y muestra de todos los elementos del arreglo
    cout << "Elementos del arreglo: ";
    for (int i = 0; i < 5; i++) {
        cout << numeros[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Arreglos a funciones

Los arreglos pueden ser pasados como argumentos a funciones. Esto permite trabajar con los elementos del arreglo dentro de la función y realizar modificaciones. Al pasar un arreglo a una función, se pasa por referencia, lo que permite modificar el contenido del arreglo en la función.

Para pasar un arreglo a una función, el nombre del arreglo deberá aparecer sin corchetes o índices, es decir, como un argumento actual dentro de la llamada de la función.

Búsqueda de datos en arreglos

La búsqueda de datos en arreglos consiste en encontrar la posición de un valor específico dentro del arreglo. Entre estos tipos de búsqueda encontramos:

- **Búsqueda secuencial o lineal:** consiste en recorrer secuencialmente un arreglo desde el primer elemento hasta el último y comprobar si alguno de los elementos del arreglo contiene el vector buscado, o sea, comparar el elemento del arreglo con el valor a buscar.
- **Búsqueda binaria:** consiste en analizar en primer lugar el elemento central del vector. Si el elemento buscado es menor, se busca por el tramo inferior del vector utilizando la misma técnica. De lo contrario, por el tramo superior.

Ordenamiento de arreglos

El ordenamiento en los arreglos implica reorganizar los elementos en un orden específico, como ascendente o descendente. Los algoritmos de ordenamiento más comunes son el ordenamiento de burbuja, inserción y ordenamiento rápido.

- **Ordenamiento por burbuja:** se basa en comparar pares de elementos adyacentes de un vector.

- **Ordenamiento por inserción:** se basa en el recorrido por la lista seleccionando en cada iteración un valor como clave y compararlo con el resto.

- **Ordenamiento por selección:** consiste en encontrar el menor de todos los elementos del vector e intercambiarlo con el que está en primera posición.

Arreglos multidimensionales

Son arreglos que tienen más de una dimensión, tales como matrices o tablas. Se declaran indicando el tamaño de cada dimensión y se acceden a los elementos utilizando la cantidad de índices que se especificaron en el tamaño de la dimensión.

A continuación se presenta un ejemplo de un arreglo de dos dimensiones:

```
#include <iostream>

using namespace std;

int main() {
    // Declaración y creación de un arreglo de dos dimensiones (3x3)
    int matriz[3][3] = {{1, 2, 3},
                        {4, 5, 6},
                        {7, 8, 9}};

    // Acceso a elementos del arreglo multidimensional
    cout << "Elemento en la posición (0, 0): " << matriz[0][0] << endl;
    cout << "Elemento en la posición (1, 2): " << matriz[1][2] << endl;

    // Modificación de un elemento del arreglo multidimensional
    matriz[2][1] = 10;
    cout << "Elemento modificado en la posición (2, 1): " << matriz[2][1] << endl;

    // Recorrido y muestra de todos los elementos del arreglo multidimensional
    cout << "Elementos de la matriz:" << endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matriz[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Búsqueda y ordenamiento

Para la búsqueda y ordenamiento se plantean los siguientes algoritmos:

- **Algoritmos de búsqueda:** los algoritmos de búsqueda en arreglos incluyen los de búsqueda lineal (recorrer secuencialmente un array) y búsqueda binaria (analizar el dato por los extremos inferior y superior).
- **Algoritmos de ordenamiento:** incluyen el ordenamiento burbuja (comparar pares de elementos), por inserción (recorrido utilizando valores como clave) y selección (intercambio del menor con el de la primera posición).

Motores de bases de datos

Los sistemas de gestión de bases de datos (SGDB) son programas que permiten introducir y almacenar datos, ordenarlos y manipularlos. Cada motor de bases de datos posee sus propias características, rendimiento, capacidad de escalabilidad y soporte de características específicas.

Entre estos motores encontramos MySQL, MS SQL Server, PostgreSQL, entre otros.

Comparaciones características

Al comparar motores de bases de datos, se evalúan características como la facilidad de uso, la escalabilidad, el rendimiento, seguridad, disponibilidad de soporte y la compatibilidad con los estándares.

Para comparar y ver sus características de algunos motores de bases de datos, podemos analizar la siguiente tabla:

Tabla comparativa de los distintos SGDB.

SGBD	Características	Ventajas	Inconvenientes
ACCESS	Perteneciente a Microsoft. Es muy gráfico. Métodos simples y directos, con formularios, para trabajar con la información.	Asequible para personas con poco manejo con las bases de datos. Crea varias vistas para una misma información.	No es multiplataforma. No funciona con bases de datos grandes, tanto para registros como para usuarios.
SQLite	Los tipos de datos se asignan a valores individuales y no a la columna como la mayoría de los SGBD.	Multiplataforma. No requiere configuración. Acceso muy rápido. No requiere servidor.	El dinamismo de los datos hace que no se portable a otras bases de datos. Falta de clave foráneas.
SQL SERVER	Software propietario. El lenguaje es TSQL.	Multiplataforma, aunque pertenezca a Microsoft. Transacciones.	Utiliza mucha RAM. Tamaño de página fijo y pequeño. Relación calidad/precio inferior a Oracle.
MYSQL	Pertenece a Oracle. Licencia GPL/Licencia comercial.	Agrupación de transacciones. Distintos motores de almacenamiento. Instalación sencilla.	No tiene soporte. Capacidad limitada.
POSTGRESQL	Tiene la extensión POSTGIS para bases de datos espaciales.	Código abierto y gratuito, multiplataforma. Gran volumen de datos. Transacciones, disparadores y afirmaciones.	Respuesta lenta. Requiere hardware. No es intuitivo.
ORACLE	Dispone de su propio lenguaje PL/SQL. Soporta bases de datos de gran tamaño.	Es el más usado a nivel mundial. Multiplataforma. Es intuitiva y fácil de usar.	Precio muy elevado. Elevado coste de la información, tratado por trabajadores formados por Oracle.

UNIDAD 4

SQL (DML y DDL)

DDL (Data Definition Languaje)

DDL (Lenguaje de Definición de Datos) es una categoría del lenguaje SQL que permite definir y modificar la estructura de la base de datos. Entre sus sentencias encontramos:

- **CREATE:** crea objetos en la base de datos.
- **ALTER:** modifica la estructura de la base de datos.
- **DROP:** borra objetos de la base de datos.
- **TRUNCATE:** elimina todos los registros de la tabla.

DML (Data Manipulation Languaje)

DML (Lenguaje de Manipulación de Datos) es una categoría del lenguaje SQL que permite realizar operaciones de manipulación y consulta de datos, pudiendo recuperar, insertar, eliminar y modificar la información almacenada. Entre sus sentencias encontramos:

- **SELECT:** obtiene datos de una base de datos.
- **INSERT:** inserta datos a una tabla.
- **UPDATE:** modifica datos existentes dentro de una tabla.
- **DELETE:** elimina todos los registros de la tabla.

UNIDAD 5

Memoria dinámica

La memoria dinámica permite asignar y liberar memoria durante la ejecución de un programa. Se utiliza para trabajar con estructuras de datos flexibles y para evitar limitaciones de tamaño de los arreglos estáticos.

La reserva de memoria dinámica se hace en tiempo de ejecución después de leer los datos y de conocer el tamaño exacto del problema a resolver.

Declaración e inicialización de punteros

Los punteros son variables que almacenan direcciones de memoria. Se declaran utilizando el tipo de dato seguido de un asterisco (*) y se inicializan con la dirección de memoria de otra variable.

Los punteros se utilizan para acceder y manipular datos a través de la memoria dinámica. Un ejemplo de la inicialización de punteros puede ser:

```
#include <iostream>

using namespace std;

int main() {
    int numero = 10;
    int* puntero = &numero;

    cout << "Valor de la variable: " << numero << endl;
    cout << "Dirección de memoria de la variable: " << &numero << endl;
    cout << "Valor del puntero: " << puntero << endl;
    cout << "Valor al que apunta el puntero: " << *puntero << endl;

    return 0;
}
```

Arrays de punteros

En el array de punteros, se dice que elemento de un arreglo es un puntero. Suelen utilizarse para almacenar múltiples punteros y acceder a ellos mediante índices. Un ejemplo podría ser:

```
#include <iostream>

using namespace std;

int main() {
    int numero1 = 10;
    int numero2 = 20;
    int numero3 = 30;

    int* arregloPunteros[3];
    arregloPunteros[0] = &numero1;
    arregloPunteros[1] = &numero2;
    arregloPunteros[2] = &numero3;

    for (int i = 0; i < 3; i++) {
        cout << "Valor del elemento " << i + 1 << ": " << *arregloPunteros[i] << endl;
    }
```

```
    return 0;
}
```

Aritmética de punteros

La aritmética de punteros consiste en la realización de operaciones matemáticas en los punteros, tales como incrementar o decrementar su valor. Conforma el conjunto de operaciones en las cuales podemos realizar en un puntero su creación para un arreglo y así manipularlo.

Para esto se basa en el tamaño del tipo de dato al que apunta el puntero.

Operador Sizeof

El operador sizeof devuelve el tamaño en bytes de un tipo de dato o una variable. Se utiliza para calcular la cantidad de memoria asignada a una estructura de datos o un tipo de dato.

Además, este operador informa del tamaño de almacenamiento utilizado por cualquier objeto, así este sea un tipo básico o incluso derivado.

Relación entre apuntadores y arreglos

Los arreglos en C++ se basan en punteros, donde el nombre del arreglo es un puntero al primer elemento de este. Los punteros se utilizan para acceder a los elementos del arreglo y realizar operaciones en ellos.

Un apuntador a una función contendrá la dirección que tiene la función en la memoria.

Apuntadores a funciones

Los punteros a funciones son punteros que almacenan la dirección de memoria de una función. Se utilizan para llamar a una función a través del puntero, lo que permite la flexibilidad en la llamada a funciones en tiempo de ejecución.

Asignación de memoria dinámica

La asignación de memoria dinámica permite reservar espacio en memoria durante la ejecución del programa. Se utiliza el operador “new” para asignar memoria y el operador “delete” para liberarla cuando ya no se necesita.

La reserva de memoria dinámica añade una gran flexibilidad a los programas porque permite al programador la posibilidad de reservar la cantidad de memoria exacta en el preciso instante en el que se necesite, sin tener que realizar una reserva por exceso en prevención a la que pueda llegar a necesitar.

Dado que los punteros se pueden aplicar a cualquier tipo de variable se puede entonces realizar una asignación de memoria dinámica para cualquier variable.

Introducción a listas enlazadas (conceptos)

Las listas enlazadas son estructuras de datos dinámicas que permiten almacenar y manipular una secuencia de elementos enlazados mediante punteros. Cada elemento de la lista contiene un dato y un puntero que apunta al siguiente elemento de la lista.

Existen diversos métodos para construir una lista enlazada individualmente, entre ellos encontramos:

- **Método ingenuo** (asignar memoria para todos los nodos individuales)
- **Línea única** (función de newNode())
- **Método genérico** (recorrido del contenedor)
- **Entre otras.**

UNIDAD 6

Introducción a la estructura de datos en C++

Las estructuras de datos en C++ son formas organizadas de almacenar y manipular datos para su eficiente acceso y modificación.

C++ proporciona clases y bibliotecas estándar para implementar estructuras de datos como listas, pilas, colas y árboles. Las estructuras de datos permiten almacenar de manera ordenada una serie de valores dados en una misma variable.

Clases auto referenciadas

Las clases auto referenciadas son clases que contienen miembros que son punteros a la misma clase. Esto permite la creación de estructuras de datos complejas como listas enlazadas, donde cada nodo de la lista es un objeto de la misma clase.

Listas enlazadas

Las listas enlazadas son estructuras de datos dinámicas que consisten en una secuencia de nodos enlazados mediante punteros.

Cada nodo contiene un dato y un puntero que apunta al siguiente nodo de la lista. Existen diversos métodos para construir una lista enlazada individualmente, los cuales vimos en la introducción a las listas enlazadas.

Asignación dinámica de memoria y estructura de datos

La asignación dinámica de memoria se utiliza para crear estructuras de datos de tamaño variable durante la ejecución del programa. Permite la gestión eficiente de la memoria y la adaptabilidad de las estructuras de datos a medida que cambian los requisitos del programa.

Pilas y colas

Las pilas y colas son estructuras de datos lineales que permiten el acceso y la modificación de los elementos siguiendo un orden específico. Para esto veremos brevemente qué son cada una de ellas:

- **Pilas:** funcionan según el principio LIFO (Last-In-First-Out), donde el último elemento insertado es el primero en ser eliminado.
- **Colas:** funcionan según el principio FIFO (First-In-First-Out), donde el primer elemento insertado es el primero en ser eliminado.

Un ejemplo de las pilas y colas en C++:

```
#include <iostream>
#include <stack>
#include <queue>

using namespace std;
```

```
int main() {
    // Ejemplo de uso de una pila
    stack<int> pila;

    pila.push(10);
    pila.push(20);
    pila.push(30);

    cout << "Elementos de la pila: ";
    while (!pila.empty()) {
        cout << pila.top() << " ";
        pila.pop();
    }
    cout << endl;

    // Ejemplo de uso de una cola
    queue<string> cola;

    cola.push("Hola");
    cola.push("Mundo");
    cola.push("!");

    cout << "Elementos de la cola: ";
    while (!cola.empty()) {
        cout << cola.front() << " ";
        cola.pop();
    }
    cout << endl;

    return 0;
}
```