UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA
FACULTAD DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Nombre Completo	Carné
Kevin Geovanny Alvarado Coj	1290-14-6380

UNIDAD 1

Lenguaje en C++

C++ es un lenguaje de programación de alto nivel que fue diseñado por Bjarne Stroustrup en 1979 como una extensión del lenguaje C desarrollado por Dennis Ritchie. Se considera a C++ como un lenguaje de programación multiparadigma, lo que significa que admite varios estilos de programación, incluyendo la programación estructurada y la programación orientada a objetos.

La intención detrás del diseño de C++ fue combinar la eficiencia y el control de bajo nivel del lenguaje C con características de programación orientada a objetos. Al agregar conceptos como clases, objetos, herencia y polimorfismo, C++ permitió a los programadores abordar problemas de manera más modular y facilitó la reutilización de código.

Además de ser un lenguaje orientado a objetos, C++ también ofrece soporte para otros paradigmas de programación, como la programación genérica a través de las plantillas y la programación funcional mediante el uso de funciones lambda.

A lo largo de los años, C++ ha evolucionado y se ha convertido en uno de los lenguajes de programación más utilizados en la industria del software. Su amplia adopción se debe a su capacidad para desarrollar aplicaciones de alto rendimiento y su compatibilidad con sistemas operativos y arquitecturas de hardware diversos.

En resumen, C++ es un lenguaje de programación de alto nivel derivado de C, diseñado por Bjarne Stroustrup. Es multiparadigma y combina las características de programación estructurada y orientada a objetos. Con su

flexibilidad y eficiencia, C++ se ha convertido en una herramienta poderosa para el desarrollo de software en diversos entornos.

Herramientas de desarrollo

Existen diferentes tipos de herramientas de desarrollo utilizadas para C++. Las siguientes son los diferentes tipos y los más utilizados:

I. Compiladores de C++: Los compiladores son herramientas esenciales para traducir el código fuente de C++ a un código ejecutable. Algunos compiladores populares de C++ son:

- GCC (GNU Compiler Collection): Es un conjunto de compiladores de código abierto y ampliamente utilizado, que incluye el compilador de C++ g++.
- Clang: Es otro compilador de código abierto que se ha vuelto popular en los últimos años debido a su enfoque en la calidad del diagnóstico de errores.
- Microsoft Visual C++: Es el compilador oficial de Microsoft para Windows y es parte del entorno de desarrollo integrado (IDE) Visual Studio.

II. Entornos de desarrollo integrado (IDE): Los IDE proporcionan un conjunto completo de herramientas para el desarrollo de software, que incluyen características como resaltado de sintaxis, depuración, administración de proyectos y compilación integrada. Algunos IDEs populares para C++ son:

- Visual Studio: Es un IDE desarrollado por Microsoft que ofrece una amplia gama de herramientas y características para el desarrollo en C++.
- Visual Studio: Es un IDE desarrollado por Microsoft que ofrece una amplia gama de herramientas y características para el desarrollo en C++.
- Code::Blocks: Es un IDE de código abierto y multiplataforma que es ampliamente utilizado para el desarrollo en C++.

III. Editores de texto avanzados: Además de los IDEs, muchos programadores de C++ prefieren utilizar editores de texto más ligeros pero potentes, con características como resaltado de sintaxis y autocompletado. Algunos editores populares para C++ son:

- Visual Studio Code: Es un editor de texto de código abierto y altamente personalizable que es ampliamente utilizado por su amplia gama de extensiones

para el desarrollo en C++.

- Sublime Text: Es un editor de texto liviano y altamente personalizable que también es popular entre los programadores de C++.
- Vim y Emacs: Son editores de texto avanzados y altamente configurables que son preferidos por algunos desarrolladores de C++ debido a su flexibilidad y potencia.

IV. Sistemas de control de versiones: Para el desarrollo de software colaborativo y el seguimiento de cambios en el código fuente, es común utilizar sistemas de control de versiones. Algunos sistemas populares son:

- Git: Es un sistema de control de versiones distribuido ampliamente utilizado que permite el seguimiento de cambios, la colaboración y la administración del código fuente.
- Subversion (SVN): Es otro sistema de control de versiones ampliamente utilizado que proporciona características similares a Git, pero sigue un enfoque centralizado.

Estas son solo algunas de las herramientas más populares utilizadas en el desarrollo de C++. La elección de la herramienta depende de las preferencias personales y los requisitos específicos del proyecto.

Tipos de datos nativos en C++

En C++, existen varios tipos de datos nativos incorporados en el lenguaje que se utilizan para representar diferentes tipos de valores. Estos se pueden dividir en: enteros, de punto flotante, de caracteres, booleano y void. Los siguientes son ejemplos de estos datos los cuales son utilizados en este lenguaje:

<u>Tipos de datos enteros:</u>

- int: Representa números enteros con signo.
- unsigned int: Representa números enteros sin signo. - short: Representa números enteros cortos con signo.
- unsigned short: Representa números enteros cortos sin signo. - long: Representa números enteros largos con signo.
- unsigned long: Representa números enteros largos sin signo.
- long long: Representa números enteros largos con signo de mayor tamaño.

- unsigned long long: Representa números enteros largos sin signo de mayor tamaño.

Tipo de dato de punto flotante:

- float: Representa números de punto flotante de precisión simple.
- double: Representa números de punto flotante de precisión doble.
- long double: Representa números de punto flotante de mayor precisión.

Tipo de dato de caracteres:

- char: Representa un único carácter.
- wchar_t: Representa un carácter de ancho extendido. - char16_t: Representa un carácter Unicode de 16 bits. - char32_t: Representa un carácter Unicode de 32 bits.

Tipo de dato booleano:

- bool: Representa un valor booleano que puede ser verdadero (true) o falso (false).

Tipo de dato void:

- void: Indica la ausencia de tipo o valor. Se utiliza principalmente en funciones que no devuelven ningún valor o en punteros genéricos.

Además de estos tipos de datos nativos, C++ también permite a los programadores definir sus propios tipos de datos personalizados utilizando clases y estructuras.
Es importante tener en cuenta que los tamaños y rangos de estos tipos de datos nativos pueden variar según la plataforma y el compilador utilizados.

Bibliotecas en C++

C++ cuenta con una amplia gama de bibliotecas estándar y bibliotecas de terceros que proporcionan funcionalidades adicionales y facilitan el desarrollo de aplicaciones. A continuación, se mencionan algunas de las bibliotecas más populares en C++:

- Biblioteca Estándar de C++ (STL): Es la biblioteca estándar proporcionada por C++ y ofrece una amplia gama de clases y funciones para manipular

contenedores, algoritmos y manipulación de cadenas, entrada y salida, manejo de excepciones, entre otros.

- Boost: Es una biblioteca de C++ de alta calidad y ampliamente utilizada que proporciona soluciones para una variedad de tareas, como programación genérica, manipulación de archivos y directorios, procesamiento de imágenes, concurrencia, entre otros.

- Qt: Es una biblioteca multiplataforma ampliamente utilizada para el desarrollo de interfaces gráficas de usuario (GUI). Proporciona una amplia gama de herramientas y widgets para crear aplicaciones con una apariencia y funcionalidad profesional.

- OpenCV: Es una biblioteca de visión por computadora y procesamiento de imágenes. Proporciona una amplia gama de funciones y algoritmos para el procesamiento de imágenes, detección de objetos, reconocimiento facial, seguimiento de objetos, entre otros.
- Eigen: Es una biblioteca de álgebra lineal de C++ de alto rendimiento. Proporciona una amplia gama de clases y funciones para realizar operaciones matemáticas eficientes en matrices y vectores.

- Poco: Es una biblioteca C++ de código abierto y multiplataforma que ofrece un conjunto de componentes y clases para desarrollar aplicaciones de red y servicios web, manejo de hilos, almacenamiento de datos y más.

- CppUnit: Es una biblioteca para realizar pruebas unitarias en C++. Proporciona un conjunto de macros y clases que permiten escribir y ejecutar pruebas automatizadas para verificar el correcto funcionamiento de las unidades de código.

Estas son solo algunas de las bibliotecas populares en C++. Hay muchas otras bibliotecas disponibles para diferentes áreas de desarrollo, como matemáticas, gráficos, procesamiento de audio, aprendizaje automático, entre otros. La elección

de una biblioteca depende de los requisitos y necesidades específicas del proyecto.

Entrada y salida de flujos

En C++, la entrada y salida de datos se realiza a través de flujos (streams). Los flujos proporcionan una interfaz para leer datos desde una fuente (entrada) o escribir datos hacia un destino (salida). La biblioteca estándar de C++ (STL) proporciona dos tipos principales de flujos: flujos de entrada (istream) y flujos de salida (ostream).

Los flujos de entrada (istream) se utilizan para leer datos de una fuente, como el teclado o un archivo. Algunas de las funciones más comunes para la entrada de datos son:

- >>: El operador de extracción se utiliza para leer datos desde el flujo de entrada en una variable. Por ejemplo, cin >> variable se utiliza para leer un valor y almacenarlo en la variable especificada.

- getline(): Esta función se utiliza para leer una línea completa de texto desde el flujo de entrada y almacenarla en una cadena (string).

Los flujos de salida (ostream) se utilizan para escribir datos en un destino, como la consola o un archivo. Algunas de las funciones más comunes para la salida de datos son:

- <<: El operador de inserción se utiliza para escribir datos en el flujo de salida. Por ejemplo, cout << “Hola, ¡mundo!" se utiliza para escribir el texto "Hola, mundo!" en la consola.

- put(): Esta función se utiliza para escribir un carácter en el flujo de salida.

- write(): Esta función se utiliza para escribir una secuencia de bytes en el flujo de salida.

Además de los flujos estándar cin y cout, también se pueden utilizar flujos de archivo para leer y escribir en archivos. Para ello, se utilizan las clases ifstream y ofstream para flujos de entrada y salida de archivo, respectivamente.

Los flujos de entrada (istream) se utilizan para leer datos desde una fuente, mientras que los flujos de salida (ostream) se utilizan para escribir datos en un destino. Los flujos estándar cin y cout se utilizan para la entrada y salida en la consola, respectivamente, y también se pueden utilizar flujos de archivo para leer y escribir en archivos.

Programación estructurada

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa utilizando únicamente subrutinas o funciones y tres estructuras: secuencial, de selección y repetitiva.

En C++ es un enfoque de programación que se basa en la organización lógica y estructurada del código. Se centra en dividir el programa en módulos más pequeños y utilizar estructuras de control como secuencias, bucles y decisiones para controlar el flujo de ejecución del programa. Aquí hay algunos conceptos clave de la programación estructurada en C++:

- Funciones: En C++, las funciones se utilizan para dividir el código en bloques más pequeños y lógicos. Cada función tiene un propósito específico y puede recibir argumentos y devolver un valor si es necesario. Las funciones ayudan a modularizar el código y mejorar la legibilidad y reutilización del mismo.
- Estructuras de control:

- Secuencia: Las instrucciones se ejecutan secuencialmente en el orden en que aparecen en el código.
- Bucles: Los bucles, como for, while y do-while, se utilizan para repetir un bloque de código varias veces hasta que se cumpla una condición específica.
- Decisiones: Las estructuras de decisión, como if, else if y switch, se utilizan para tomar decisiones basadas en condiciones específicas y ejecutar diferentes bloques de código en consecuencia.

- Variables y tipos de datos: En la programación estructurada, las variables se utilizan para almacenar y manipular datos. C++ proporciona varios tipos de datos nativos, como enteros, flotantes, caracteres, booleanos, entre otros. Las variables deben declararse antes de su uso y pueden tener un alcance local o global.

- Modularidad: La programación estructurada fomenta la modularidad al dividir el código en módulos o funciones más pequeñas y coherentes. Esto facilita la comprensión, el mantenimiento y la reutilización del código.
- Evitar saltos incondicionales: En la programación estructurada, se evita el uso de saltos incondicionales como goto. En su lugar, se utilizan estructuras de control y bucles para controlar el flujo de ejecución de manera más legible y estructurada.

La programación estructurada en C++ se basa en principios de diseño claros y estructurados, lo que facilita el desarrollo y el mantenimiento del código. Sin embargo, es importante tener en cuenta que C++ también admite otros paradigmas, como la programación orientada a objetos, que pueden ofrecer beneficios adicionales en ciertos escenarios de desarrollo.

Programación orientada a objetos y sus propiedades

La programación orientada a objetos es un paradigma de programación que usa objetos y sus interacciones para el desarrollo de aplicaciones. La POO se basa en la idea natural de un mundo lleno de objetos y que la resolución de problemas se realiza mediante el modelo de objetos.

En C++, la POO se puede implementar utilizando clases y objetos, y proporciona varios conceptos clave:

- Clases: Las clases son estructuras de datos que encapsulan datos y comportamientos relacionados. Una clase actúa como una plantilla o molde para crear objetos. Define los atributos (variables miembro) y los métodos (funciones miembro) que describen las características y acciones de los objetos.

- Objetos: Los objetos son instancias de una clase específica. Cada objeto tiene su propio conjunto de atributos y puede realizar acciones definidas por los métodos de su clase.

- Encapsulación: La encapsulación es un principio de la POO que implica ocultar los detalles internos de una clase y proporcionar una interfaz para interactuar con los objetos. Los datos y los métodos relacionados se agrupan en una clase, y el acceso a los datos se controla mediante modificadores de acceso,

como public, private y protected.

- Herencia: La herencia es un mecanismo que permite crear nuevas clases basadas en clases existentes. Permite heredar atributos y métodos de una clase base (superclase) y extender o modificar su funcionalidad en una clase derivada (subclase). La herencia facilita la reutilización de código y la organización jerárquica de las clases.
- Polimorfismo: El polimorfismo permite que objetos de diferentes clases se comporten de manera similar. Permite utilizar un objeto de una clase base para manipular objetos de diferentes clases derivadas. El polimorfismo se logra mediante el uso de funciones virtuales y el mecanismo de enlace dinámico.
- Abstracción: La abstracción implica identificar las características y comportamientos esenciales de un objeto y modelarlos en una clase. Permite simplificar y centrarse en los aspectos relevantes para el problema en cuestión, ocultando los detalles innecesarios.

La programación orientada a objetos ofrece varios beneficios, como la modularidad, el reuso de código, la legibilidad, el mantenimiento y la escalabilidad del software. Permite un diseño más estructurado y flexible, facilitando la colaboración entre equipos de desarrollo. C++ combina la programación orientada a objetos con características de programación estructurada, lo que le da un amplio rango de aplicaciones y flexibilidad en el diseño y desarrollo de sistemas de software.

Clase y super clase

En C++, una clase es una estructura de programación que actúa como un plano o plantilla para crear objetos. Define los atributos y los métodos que describen las características y el comportamiento de los objetos creados a partir de ella. Una clase encapsula datos y funciones relacionados en una única entidad.

Una superclase, también conocida como clase base o clase padre, es una clase de la cual se derivan otras clases. Proporciona una base común de atributos y comportamientos que se heredan por las clases derivadas. En C++, se implementa la herencia para lograr la relación entre una superclase y una subclase.

En resumen, una clase en C++ define una estructura de programación que representa un objeto, mientras que una superclase es una clase base de la cual se derivan otras clases, permitiendo la herencia de atributos y comportamientos. La herencia en C++ facilita la construcción de jerarquías de clases y la reutilización de código.

Sobrecarga de operadores

La sobrecarga de operadores en C++ es una característica que permite proporcionar una implementación personalizada para los operadores existentes, lo que permite que los objetos de una clase se comporten de manera similar a los tipos de datos básicos. Esto significa que puedes definir cómo se comportan los operadores aritméticos, de asignación, de comparación, entre otros, para los objetos de tu propia clase.

La sobrecarga de operadores se realiza mediante la definición de funciones miembro o funciones no miembro especiales llamadas "operadores sobrecargados". Estas funciones tienen una sintaxis especial que indica qué operador se está sobrecargando.

La sobrecarga de operadores es una herramienta la cual nos permite personalizar el comportamiento de los operadores para trabajar con objetos de nuestra propia clase. Podemos sobrecargar una amplia variedad de operadores, como aritméticos, de asignación, de comparación, de indexación, entre otros, según tus necesidades específicas.

Propiedades y métodos de una clase

Las propiedades y métodos de una clase en C++ son los elementos fundamentales que definen el estado y el comportamiento de los objetos creados a partir de esa clase.

Las propiedades y métodos de una clase en C++ permiten encapsular datos y comportamientos relacionados en una única entidad. Esto ofrece varios beneficios, como la modularidad, el ocultamiento de datos, la reutilización de código y la organización estructurada del programa. También facilita la interacción entre objetos a través de métodos que se comunican y operan en las propiedades

de los objetos.

Propiedades (o atributos):

- Las propiedades representan los datos asociados a un objeto de la clase. Son variables miembro declaradas dentro de la clase. Pueden ser de diferentes tipos de datos, como enteros, flotantes, caracteres, punteros, objetos de otras clases, etc.
- Las propiedades encapsulan el estado de un objeto y definen sus características. Pueden ser públicas, privadas o protegidas, lo que determina el alcance y la accesibilidad de las propiedades desde fuera de la clase.
- Las propiedades se inicializan en el constructor de la clase y se pueden acceder y modificar utilizando los métodos de la clase.

Métodos (o funciones miembro):

- Los métodos definen el comportamiento y las acciones que un objeto de la clase puede realizar. Son funciones miembro de la clase que se definen dentro de la clase y operan en los datos de la clase (las propiedades).
- Los métodos pueden tener parámetros y devolver valores. Pueden realizar operaciones sobre las propiedades del objeto, interactuar con otros objetos o realizar cálculos.
- Los métodos pueden ser públicos, privados o protegidos, lo que determina su accesibilidad desde fuera de la clase. Los métodos públicos se pueden llamar desde fuera de la clase, mientras que los métodos privados solo se pueden llamar desde dentro de la clase.
- Los métodos pueden ser definidos en la propia declaración de la clase (dentro de la definición de la clase) o fuera de la declaración (definidos por separado en la implementación de la clase).

Es importante tener en cuenta que el diseño adecuado de las propiedades y métodos de una clase es esencial para crear una estructura lógica y coherente. Los nombres de las propiedades y métodos deben ser descriptivos y representar adecuadamente la funcionalidad y el propósito de la clase. Además, es importante considerar los principios de encapsulación y cohesión al definir las propiedades y métodos para lograr una implementación eficiente y fácil de

mantener.

Creación de objetos

La creación de objetos en C++ implica instanciar una clase para crear una entidad específica con su propio conjunto de propiedades y comportamiento. Los siguientes son conceptos importantes para poder crear dichos objetos:

Declaración de clase:

- Antes de crear objetos, debes tener una clase definida. La declaración de clase establece la estructura y las características de los objetos que se pueden crear a partir de ella.
- La declaración de clase incluye la definición de propiedades (atributos) y métodos que representan las características y el comportamiento de los objetos.

Constructor:

- Un constructor es un método especial dentro de una clase que se invoca automáticamente cuando se crea un objeto.
- El constructor se utiliza para inicializar las propiedades del objeto y realizar cualquier configuración inicial necesaria.
- Puedes definir un constructor predeterminado sin parámetros o también puedes crear constructores con parámetros para permitir una inicialización personalizada.

Sintaxis de creación de objetos:

- La sintaxis para crear un objeto en C++ implica utilizar el operador new seguido del nombre de la clase y, opcionalmente, pasar argumentos al constructor si se define uno.
- Por ejemplo, para crear un objeto de una clase llamada MiClase, puedes usar: **MiClase* objeto = new MiClase();**
- Esta sintaxis crea un objeto de MiClase en el montón (heap) y devuelve un puntero a ese objeto.

Destrucción de objetos:

- Después de crear un objeto, es importante liberar la memoria asignada cuando ya no se necesita.

- En C++, se utiliza el operador delete para liberar la memoria ocupada por el objeto creado con new.
- Por ejemplo, para liberar el objeto objeto creado anteriormente, se puede usar: **delete objeto;**
- La liberación de memoria con delete también puede llevarse a cabo en destructores personalizados definidos dentro de la clase.

Ciclo de vida del objeto:

- El ciclo de vida de un objeto en C++ comienza con su creación y finaliza con su destrucción.
- Durante el ciclo de vida, puedes acceder a las propiedades y llamar a los métodos del objeto para realizar operaciones y manipulaciones.
- Los objetos pueden tener una vida útil local dentro de una función o una vida útil extendida si se almacenan en el montón (heap) y se administran manualmente.

La creación de objetos en C++ permite la instanciación de clases y la creación de entidades individuales con su propio estado y comportamiento.Es importante gestionar adecuadamente el ciclo de vida de los objetos para evitar pérdidas de memoria y liberar recursos de manera adecuada cuando ya no sean necesarios.

Constructores

Los constructores en C++ son métodos especiales dentro de una clase que se utilizan para inicializar los objetos de esa clase. Un constructor se invoca automáticamente al crear un objeto y tiene el mismo nombre que la clase. Varios constructores pueden ser definidos en una clase, con diferentes listas de parámetros, lo que se conoce como constructores sobrecargados. Los constructores se utilizan para establecer los valores iniciales de las propiedades del objeto y realizar cualquier configuración necesaria. Pueden tener parámetros para permitir una inicialización personalizada y utilizan listas de inicialización para asignar valores directamente a las propiedades. Los constructores en C++ son una parte fundamental en la creación de objetos, ya que permiten una flexibilidad en la inicialización y configuración de las propiedades de los objetos.

UNIDAD 2

Operadores Lógicos

Los operadores lógicos en C++ son utilizados para realizar operaciones de lógica booleana en expresiones condicionales. Estos operadores permiten combinar o evaluar condiciones y producir un resultado basado en la veracidad o falsedad de las mismas. Los operadores más utilizados son los siguientes:

Operador AND lógico (&&):

Este operador devuelve true si ambas expresiones que lo rodean son verdaderas, y false en caso contrario.

Operador OR lógico (||):

Este operador devuelve true si al menos una de las expresiones que lo rodean es verdadera, y false si ambas son falsas.

Operador NOT lógico (!):

Este operador devuelve el valor opuesto de la expresión que lo sigue. Si la expresión es true, el operador NOT devuelve false, y viceversa.

Estos operadores se utilizan principalmente en estructuras de control condicionales, como las sentencias if, while, for, entre otras, para tomar decisiones basadas en condiciones lógicas. También se pueden combinar entre sí y con operadores de comparación para formar expresiones más complejas.

Es importante tener en cuenta que los operadores lógicos siguen reglas de evaluación de cortocircuito. Esto significa que si el resultado de la expresión se puede determinar por la evaluación de una parte de la expresión, el resto de la expresión no se evalúa. Esto es útil para optimizar el rendimiento y evitar evaluaciones innecesarias en ciertos casos. Estos operadores son esenciales para el control del flujo de un programa y la toma de decisiones basadas en condiciones lógicas.

Estructuras de control

Las estructuras de control en C++ son construcciones que permiten controlar el flujo de ejecución de un programa. Estas estructuras determinan qué instrucciones se ejecutan y en qué orden, dependiendo de las condiciones y criterios establecidos.

La estructura de control condicional **if** permite ejecutar un bloque de código si se cumple una condición determinada. Puede estar seguida de estructuras **else if** y un bloque **else** opcional para manejar múltiples condiciones.

La estructura de control **switch** permite seleccionar un camino de ejecución entre varios posibles, según el valor de una expresión.

Las estructuras de control iterativas, como **while, do-while** y **for**, permiten repetir un bloque de código mientras se cumpla una condición específica. La diferencia principal entre ellas radica en el momento en que se evalúa la condición. Estas estructuras de control en C++ son fundamentales para controlar el flujo del programa, tomar decisiones basadas en condiciones y repetir bloques de código según las necesidades.

Diferencia entre operadores de igualdad y de asignación

Los operadores de igualdad (==) y de asignación (=) tienen funciones diferentes en C++.

El operador de igualdad se utiliza para comparar dos valores y verificar si son iguales. Devuelve true si los valores son iguales y false si son diferentes. Este operador se emplea en expresiones condicionales y de comparación para tomar decisiones basadas en la igualdad o diferencia entre valores. Por ejemplo, se puede utilizar para verificar si dos variables tienen el mismo valor.

Por otro lado, el operador de asignación se utiliza para asignar un valor a una variable. Toma el valor a la derecha y lo asigna a la variable a la izquierda. Este operador modifica el contenido de la variable, reemplazándolo con el nuevo valor asignado. Es importante destacar que el operador de asignación no verifica la igualdad de los valores, sino que simplemente realiza la asignación.

Es fundamental distinguir entre ambos operadores, ya que confundirlos puede llevar a errores lógicos en el programa. La utilización correcta de los operadores de igualdad y de asignación es crucial para realizar comparaciones y asignaciones de manera precisa y adecuada en C++.

Recursividad

La recursividad en C++ es un concepto que permite a una función llamarse a sí

misma dentro de su propio cuerpo. Es decir, una función recursiva se invoca a sí misma para resolver un problema de manera repetitiva, dividiéndolo en subproblemas más pequeños.

Debemos considerar los siguientes factores al utilizar la recursividad:

- Caso base: La recursividad en C++ es un concepto que permite a una función llamarse a sí misma dentro de su propio cuerpo. Es decir, una función recursiva se invoca a sí misma para resolver un problema de manera repetitiva, dividiéndolo en subproblemas más pequeños.
- Caso recursivo: Es la parte de la función donde se llama a sí misma para abordar un subproblema más pequeño. Al realizar esta llamada recursiva, la función trabaja gradualmente hacia el caso base, resolviendo los subproblemas y combinando sus resultados para obtener el resultado final.

Es importante tener en cuenta algunos aspectos al utilizar la recursividad. Cada llamada recursiva crea una nueva instancia de la función con su propio conjunto de variables locales y estado de ejecución. Es crucial asegurarse de que el caso base se alcance eventualmente para evitar una recursión infinita y un desbordamiento de la pila. Además, se debe garantizar la correcta manipulación de los parámetros y variables en cada llamada recursiva para evitar errores y resultados inesperados. La recursividad puede ser útil para resolver problemas que se pueden descomponer en subproblemas más pequeños y similares.

Manejo de excepciones

El manejo de excepciones en C++ es una técnica que permite detectar y responder a condiciones excepcionales o errores durante la ejecución de un programa. Cuando se produce una excepción, el flujo normal del programa se interrumpe y se busca un bloque de código que pueda manejarla.

Cuando ocurre una excepción, el programa busca el bloque "catch" correspondiente que pueda manejarla. Si se encuentra un bloque "catch" compatible, se ejecuta el código dentro de ese bloque. Si no se encuentra ningún bloque "catch" adecuado, el programa termina y se generará un mensaje de error.

El manejo de excepciones en C++ permite detectar y manejar errores de manera

más controlada, evitando que el programa se bloquee o se cierre inesperadamente. Permite una mayor robustez y capacidad de recuperación en el código, facilitando la identificación y resolución de problemas.

Procesamiento de archivos

El procesamiento de archivos y el uso de flujos en C++ permiten manipular la información almacenada en archivos. Los archivos son recursos externos que pueden contener datos o información que se necesita leer, escribir o modificar en un programa. Los flujos en C++ actúan como interfaces que permiten la comunicación entre el programa y los archivos. Utilizando flujos de entrada y salida, se pueden abrir archivos, leer su contenido, escribir nuevos datos, realizar modificaciones y cerrar los archivos correctamente. El procesamiento de archivos y el uso de flujos en C++ son fundamentales para el intercambio de información entre programas y archivos, lo que permite el almacenamiento persistente de datos y la manipulación de archivos en diversas operaciones y aplicaciones.

UNIDAD 3

Arreglos y vectores

Los arreglos y los vectores en C++ son estructuras de datos que permiten almacenar y manipular conjuntos de elementos del mismo tipo.

Los arreglos son colecciones de elementos contiguos en memoria, donde cada elemento se accede a través de un índice numérico. Tienen un tamaño fijo y se definen en tiempo de compilación. Los arreglos en C++ pueden contener cualquier tipo de dato, como enteros, caracteres, objetos, etc. Se accede a los elementos utilizando el nombre del arreglo seguido de corchetes y el índice deseado.

Por otro lado, los vectores son contenedores dinámicos que se ajustan automáticamente para almacenar elementos. Son implementados como plantillas de la biblioteca estándar de C++. Los vectores ofrecen flexibilidad en términos de tamaño, ya que pueden crecer o reducirse según sea necesario durante la ejecución del programa. Además, proporcionan varias operaciones y funciones útiles, como agregar elementos, eliminar elementos, acceder a elementos por índice, obtener su

tamaño, entre otros.

Tanto los arreglos como los vectores son ampliamente utilizados en C++ para almacenar y manipular conjuntos de datos. Sin embargo, los vectores ofrecen una mayor flexibilidad y funcionalidad en comparación con los arreglos estáticos. La elección entre utilizar arreglos o vectores depende de los requisitos específicos del programa, como el tamaño conocido o desconocido del conjunto de datos y las operaciones requeridas.

Declaración y creación de arreglos

En C++, se pueden declarar y crear arreglos de dos formas: estática y dinámica. En la declaración estática, se especifica el tipo de datos seguido del nombre del arreglo y su tamaño entre corchetes. Los elementos del arreglo se inicializan automáticamente con valores predeterminados según el tipo de datos. Por otro lado, en la declaración dinámica, se utiliza el operador `new` para asignar memoria en tiempo de ejecución al arreglo. La memoria se asigna en el montón (heap) y debe liberarse manualmente utilizando `delete[]`. En la declaración dinámica, los elementos del arreglo no se inicializan automáticamente y se deben inicializar manualmente si es necesario. La elección entre la declaración estática y dinámica de arreglos depende de las necesidades del programa, como el tamaño conocido en tiempo de compilación o la necesidad de un tamaño variable en tiempo de ejecución.

Ejemplos del uso de arreglos

Hay diferentes formas en las cuales los arreglos nos pueden servir, por ejemplo para almacenar y acceder valores son de gran importancia, tambien los podemos utilizar para encontrar el elemento máximo o al crear matrices. El siguiente es un ejemplo de como funcionaria el código de matrices utilizando arreglos.

```
int matrix[3][3];  // Declaración estática de una matriz 3x3 de enteros
matrix[0][0] = 1;  // Asignar un valor a un elemento específico de la matriz
int element = matrix[1][2];  // Acceder a un elemento específico de la matriz
```

Estos son solo algunos ejemplos básicos del uso de arreglos en C++. Los arreglos pueden ser utilizados para una amplia variedad de aplicaciones, como almacenar datos, realizar cálculos, implementar algoritmos y manipular estructuras de datos más complejas.

Arreglos a funciones

En C++, los arreglos pueden ser pasados como argumentos a funciones, lo que permite realizar operaciones y manipulaciones en los datos contenidos en ellos. Al pasar un arreglo a una función, se puede acceder a sus elementos y modificarlos dentro del ámbito de la función. Esto proporciona una forma conveniente de trabajar con conjuntos de datos estructurados. Además, los arreglos pueden ser pasados como parámetros por valor o por referencia, dependiendo de si se desea modificar el arreglo original o no. Al utilizar arreglos como argumentos de funciones, se puede modularizar el código y reutilizarlo para diferentes conjuntos de datos, lo que promueve una proramación más eficiente y organizada.

Búsqueda de datos en arreglos

La búsqueda de datos en arreglos es una operación común en la programación. En C++, hay varias formas de buscar datos en un arreglo, dependiendo de los requisitos y la estructura de los datos. Algunos métodos comunes de búsqueda son:

1. Búsqueda lineal: Consiste en recorrer secuencialmente el arreglo desde el principio hasta el final, comparando cada elemento con el valor buscado. Si se encuentra una coincidencia, se devuelve la posición del elemento o se realiza alguna acción específica. Este método es simple pero puede ser lento para arreglos grandes.

2. Búsqueda binaria: Este método requiere que el arreglo esté ordenado previamente. Se divide el arreglo por la mitad y se compara el valor buscado con

el
elemento central. Si son iguales, se devuelve la posición del elemento. Si el valor buscado es menor, se realiza la búsqueda en la mitad inferior del arreglo; si es mayor, en la mitad superior. El proceso se repite hasta encontrar el elemento o determinar que no está presente. La búsqueda binaria es más eficiente que la búsqueda lineal, pero solo se puede utilizar en arreglos ordenados.

3. Algoritmos de búsqueda de la biblioteca estándar: C++ proporciona algoritmos de
búsqueda en la biblioteca estándar, como `std::find` y `std::binary_search`. Estos algoritmos simplifican la implementación de la búsqueda y funcionan con contenedores como arreglos y vectores. Pueden ser útiles cuando se necesita realizar búsquedas más complejas o utilizar comparadores personalizados.

La elección del método de búsqueda depende del tamaño del arreglo, si está ordenado o no, y los requisitos de rendimiento. Es importante considerar la eficiencia y la complejidad temporal de cada método para seleccionar el más adecuado para cada situación.

Ordenamiento de arreglos

El ordenamiento de arreglos es una operación fundamental en la programación que permite organizar los elementos de un arreglo en un determinado orden. En C++, existen varios algoritmos de ordenamiento disponibles para este propósito. Algunos de los algoritmos de ordenamiento comunes son:

1. Ordenamiento de burbuja (Bubble Sort): Este algoritmo compara repetidamente pares adyacentes de elementos y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que el arreglo esté completamente ordenado. Aunque
es fácil de implementar, el ordenamiento de burbuja no es eficiente para arreglos grandes.

2. Ordenamiento por inserción (Insertion Sort): En este algoritmo, se divide

el
arreglo en una parte ordenada y una parte desordenada. Se toma un elemento de la
parte desordenada y se inserta en la posición correcta en la parte ordenada. Este proceso se repite hasta que todos los elementos estén en su lugar. El ordenamiento por inserción es eficiente para arreglos pequeños o casi ordenados.

3. Ordenamiento por selección (Selection Sort): En cada iteración de este algoritmo, se busca el elemento más pequeño (o más grande) en el arreglo y se coloca en la posición correcta. El proceso se repite hasta que el arreglo esté ordenado por completo. El ordenamiento por selección también es más eficiente para arreglos pequeños.

4. Ordenamiento rápido (Quick Sort): Es un algoritmo de dividir y conquistar. Se
selecciona un elemento llamado "pivote" y se reorganizan los elementos del arreglo de manera que los elementos menores al pivote estén antes de él, y los mayores estén después. Luego, se aplica recursivamente el mismo proceso a las subpartes del arreglo. El ordenamiento rápido es eficiente en promedio y ampliamente utilizado en la práctica.

5. Ordenamiento por mezcla (Merge Sort): Es un algoritmo recursivo que divide el arreglo en mitades más pequeñas, las ordena por separado y luego las combina en un arreglo ordenado.

Cada algoritmo de ordenamiento tiene sus ventajas y desventajas en términos de rendimiento, complejidad y eficiencia en diferentes situaciones. La elección del algoritmo de ordenamiento adecuado depende del tamaño del arreglo, la distribución de los datos y los requisitos específicos del programa.

Arreglos multidimensionales

Los arreglos multidimensionales, también conocidos como matrices, son estructuras
de datos en C++ que permiten almacenar y manipular datos en más de una
dimensión. A diferencia de los arreglos unidimensionales, que tienen una sola fila de elementos, los arreglos multidimensionales tienen filas y columnas (y

posiblemente más dimensiones). Se declaran especificando el tipo de datos, el nombre del arreglo y las dimensiones en corchetes. Los elementos de una matriz se acceden utilizando índices que representan las filas y columnas correspondientes. Los bucles anidados se utilizan comúnmente para recorrer y procesar los elementos de una matriz. Los arreglos multidimensionales son útiles para representar y manipular datos estructurados en una estructura de rejilla, como matrices numéricas, tableros de juegos, imágenes y datos en formato de tabla.

Algoritmos de búsqueda y ordenamiento

Los algoritmos de búsqueda y ordenamiento son fundamentales en la programación,
ya que permiten encontrar elementos específicos en una colección de datos y organizarlos de manera ascendente o descendente según un criterio establecido. En C++, existen varios algoritmos disponibles para realizar estas operaciones de manera eficiente y conveniente.

Los algoritmos de búsqueda se utilizan para encontrar un valor particular en una
colección de datos. Algunos ejemplos de algoritmos de búsqueda comunes incluyen la búsqueda lineal, que recorre secuencialmente los elementos hasta encontrar una coincidencia, y la búsqueda binaria, que divide repetidamente la colección en dos mitades y determina en cuál mitad se encuentra el valor buscado. Estos algoritmos son útiles para buscar elementos en arreglos ordenados y no ordenados, respectivamente.

Por otro lado, los algoritmos de ordenamiento permiten organizar los elementos de una colección en un orden específico. Algunos algoritmos de ordenamiento ampliamente utilizados en C++ son el ordenamiento de burbuja, el ordenamiento por selección, el ordenamiento por inserción, el ordenamiento rápido y el ordenamiento
por mezcla. Cada algoritmo tiene su propia estrategia para comparar y reorganizar los elementos, y su eficiencia varía según el tamaño de la colección y las características de los datos. Algunos algoritmos son más adecuados para conjuntos pequeños o casi ordenados, mientras que otros son más eficientes para conjuntos grandes o desordenados.

La elección del algoritmo de búsqueda u ordenamiento adecuado depende del

contexto y los requisitos específicos del programa. Es importante considerar la

eficiencia, la estabilidad y la complejidad de cada algoritmo para seleccionar el más apropiado. Además, C++ proporciona una biblioteca estándar que ofrece implementaciones optimizadas de estos algoritmos, lo que facilita su uso en proyectos y aplicaciones.

Motores de bases de datos

Un motor de base de datos es un software que permite almacenar, gestionar y recuperar grandes cantidades de datos de manera eficiente y segura. En el ámbito de la programación, los motores de bases de datos son esenciales para la persistencia de datos y el acceso a ellos de forma rápida y confiable.

Existen diversos motores de bases de datos, cada uno con características y enfoques específicos. Algunos ejemplos populares son: MySQL, PostgreSQL, MongoDB, SQ Lite, entre otros.

Cada motor de base de datos tiene sus ventajas y desventajas, y la elección adecuada depende de los requisitos del proyecto, el volumen de datos, el rendimiento esperado y otros factores. Es importante considerar aspectos como la capacidad de escalabilidad, la seguridad, la compatibilidad con el lenguaje de programación y las necesidades específicas de la aplicación al seleccionar un motor de base de datos. Las siguientes son las diferencias entre cada motor, asi como sus ventajas y desventajas:

MySQL:

Enfoque: Relacional.

Licencia: Código abierto (versión comunitaria) y comercial (versión empresarial).

Escalabilidad: Soporta grandes volúmenes de datos y alta concurrencia. Soporte SQL: Amplio soporte para SQL estándar.

Flexibilidad: Admite múltiples motores de almacenamiento y ofrece opciones de configuración.

Comunidad: Gran comunidad de usuarios y abundante documentación.

PostgreSQL: Enfoque: Relacional.

Licencia: Código abierto.
Escalabilidad: Maneja grandes volúmenes de datos y permite la replicación y la partición de tablas.
Soporte SQL: Ofrece un soporte sólido para SQL, incluyendo características avanzadas.
Flexibilidad: Permite la definición de tipos de datos personalizados y ofrece una amplia gama de extensiones.
Características: Mayor énfasis en la integridad y la consistencia de los datos.

MongoDB:
Enfoque: NoSQL (almacenamiento de documentos). Licencia: Código abierto.
Escalabilidad: Escalable horizontalmente y diseñado para grandes volúmenes de datos.
Modelo de datos: Almacena datos en documentos flexibles en formato JSON-like (BSON).
Flexibilidad: No requiere un esquema fijo y puede adaptarse a cambios en la estructura de datos.
Rendimiento: Búsqueda rápida en documentos y soporte para operaciones complejas.

SQLite:
Enfoque: Relacional y embebido.
Licencia: Dominio público.
Tamaño y portabilidad: Pequeño y no necesita un servidor separado, adecuado para aplicaciones locales y de dispositivos móviles.
Rendimiento: Acceso rápido a los datos y transacciones ACID.
Simplificado: No requiere configuración compleja y no tiene configuraciones de servidor.

UNIDAD 4

SQL (DML y DDL)

SQL (Structured Query Language) es un lenguaje utilizado para interactuar

con bases de datos relacionales. Se divide en dos categorías principales: DML (Data Manipulation Language) y DDL (Data Definition Language).

DML se utiliza para manipular los datos almacenados en la base de datos. Algunas de las instrucciones más comunes de DML son: SELECT, INSERT, UPDATE y DELETE.

Estas instrucciones de DML son fundamentales para realizar operaciones de consulta, inserción, actualización y eliminación de datos en una base de datos.

Por otro lado, DDL se utiliza para definir y modificar la estructura de la base de datos y los objetos relacionados. Algunas de las instrucciones de DDL más utilizadas son: CREATE, ALTER, DROP y TRUNCATE.

Estas instrucciones de DDL son esenciales para diseñar y mantener la estructura de la base de datos, así como para definir las relaciones entre las tablas y asegurar la integridad de los datos.

En resumen, DML se utiliza para manipular los datos dentro de una base de datos, mientras que DDL se utiliza para definir y modificar la estructura de la base de datos
y los objetos relacionados. Ambos aspectos son fundamentales para trabajar con
bases de datos relacionales y administrar los datos de manera eficiente.

UNIDAD 5

Memoria Dinámica

Declaración e inicialización de punteros:
La declaración e inicialización de punteros se realiza mediante la especificación del
tipo de dato seguido del asterisco (*), indicando que se trata de un puntero. Los punteros pueden ser inicializados asignándoles la dirección de memoria de

una variable existente utilizando el operador de dirección (&), o mediante el uso del operador "new" para asignar memoria dinámicamente. La inicialización de punteros también puede realizarse asignándoles el valor nulo (nullptr) cuando no se requiere que apunten a una dirección de memoria específica. Los punteros son útiles para acceder y manipular datos en memoria, especialmente en escenarios donde se necesita gestionar la memoria de manera dinámica. Sin embargo, es importante tener precaución al trabajar con punteros para evitar errores de acceso a memoria no válida y asegurarse de liberar correctamente la memoria asignada dinámicamente para evitar fugas de memoria.

Arrays de punteros:

Un array de punteros es una estructura que contiene múltiples punteros como elementos de un array. Cada elemento del array es un puntero que puede apuntar a una dirección de memoria específica. La declaración de un array de punteros se realiza indicando el tipo de dato seguido del asterisco (*). Posteriormente, se pueden inicializar los elementos del array asignándoles direcciones de memoria utilizando el operador de dirección (&) o asignando memoria dinámicamente con el operador "new". Los arrays de punteros pueden ser útiles en situaciones donde se necesita almacenar y acceder a múltiples direcciones de memoria, como en la implementación de estructuras de datos complejas o en el manejo de matrices multidimensionales. Cada elemento del array puede utilizarse para acceder y manipular los datos almacenados en la dirección de memoria correspondiente, lo que proporciona flexibilidad y capacidad de gestión de la memoria en el programa.

Sin embargo, es importante tener precaución al utilizar arrays de punteros para evitar errores de acceso a memoria no válida y asegurarse de liberar correctamente la memoria asignada dinámicamente para evitar fugas de memoria.

Aritmética de punteros:

La aritmética de punteros permite realizar operaciones matemáticas con punteros, lo que facilita la manipulación de datos en memoria. Al operar con punteros, se pueden realizar sumas y restas para desplazarse a través de los elementos de

un array, accediendo a posiciones de memoria contiguas. Por ejemplo, al sumar un valor entero a un puntero, se desplaza la posición de memoria apuntada hacia adelante en función del tamaño del tipo de dato al que apunta el puntero. De manera similar, al restar un valor entero de un puntero, se desplaza hacia atrás en memoria. Esto resulta especialmente útil al trabajar con arrays, ya que permite recorrer sus elementos de manera eficiente. Es importante tener en cuenta que la aritmética de punteros debe realizarse dentro de los límites válidos del array para evitar acceder a posiciones de memoria no asignadas. Además, es esencial tener precaución al realizar operaciones de aritmética de punteros, ya que cualquier error puede llevar a resultados inesperados o comportamiento indefinido.

Operador Sizeof:

El operador sizeof se utiliza para determinar el tamaño en bytes de un tipo de dato, una variable o una expresión. Proporciona información sobre la cantidad de memoria que ocupa un objeto en la memoria del sistema.

El resultado de sizeof es un valor entero no negativo, expresado en bytes. Por ejemplo, sizeof(int) devolverá el tamaño en bytes de un entero, y sizeof(float) devolverá el tamaño en bytes de un número de punto flotante.

El operador sizeof es útil en varias situaciones, como en la asignación de memoria dinámica, el cálculo de la longitud de un array o la verificación del tamaño de un tipo de dato para garantizar la portabilidad del código.

Es importante tener en cuenta que el tamaño devuelto por sizeof puede variar en

función del sistema operativo, la arquitectura y el compilador utilizado. Además, sizeof no evalúa la longitud de cadenas de caracteres terminadas en null (\0) o arreglos dinámicos, sino que devuelve el tamaño del puntero que los representa.

Relación entre apuntadores y arreglos:

Existe una gran relación entre los apuntadores y los arreglos. Cuando se declara un arreglo, el nombre del arreglo se comporta como un puntero constante que

apunta al primer elemento del arreglo. Esto significa que se puede utilizar el nombre del arreglo para acceder al contenido de sus elementos utilizando la aritmética de punteros.

Esta relación permite acceder a los elementos de un arreglo utilizando tanto la notación de corchetes como la aritmética de punteros. Por un lado, se puede acceder a los elementos del arreglo utilizando la notación de corchetes, donde arr[i] representa el elemento en la posición i del arreglo. Por otro lado, se puede utilizar la aritmética de punteros, donde *(arr + i) o arr[i] representan el mismo elemento.

Esta relación entre apuntadores y arreglos brinda flexibilidad al trabajar con arreglos, ya que permite recorrerlos y acceder a sus elementos de manera eficiente. Además, esta relación es útil cuando se pasa un arreglo como argumento a una función, ya que los arreglos se pasan automáticamente por referencia, y al utilizar un apuntador en la función, se puede acceder y modificar los elementos del arreglo original.

Apuntadores a funciones:

En C++, los apuntadores a funciones son variables que contienen la dirección de memoria de una función en particular. Permiten tratar a las funciones como datos, lo que brinda flexibilidad y capacidad para manipularlas y utilizarlas de diversas formas en un programa.

Al utilizar apuntadores a funciones, es posible pasar funciones como argumentos a otras funciones, devolver funciones desde funciones y asignar diferentes funciones a un mismo apuntador, lo que facilita la implementación de algoritmos genéricos y el diseño modular del código.

Los apuntadores a funciones son útiles en situaciones donde se requiere una selección dinámica de funciones o cuando se necesita definir comportamientos personalizados para una operación determinada.

Es importante tener en cuenta que los apuntadores a funciones deben

ser compatibles en cuanto a su tipo de retorno y tipos de parámetros con la función a la que apuntan. Además, si una función se sobrecarga, es necesario especificar el tipo de función exacto al declarar y asignar un apuntador a esa función.

Asignación de memoria dinámica:

La asignación de memoria dinámica se refiere a la creación y gestión de memoria en tiempo de ejecución utilizando los operadores new y delete. Permite reservar y liberar bloques de memoria según sea necesario durante la ejecución del programa.

La asignación de memoria dinámica es especialmente útil cuando se necesita crear estructuras de datos de tamaño variable o cuando se requiere una cantidad de memoria que no se puede determinar en tiempo de compilación. Al asignar memoria dinámicamente, se puede controlar el ciclo de vida de los objetos y liberar la memoria cuando ya no se necesita, lo que ayuda a evitar el desperdicio de recursos.

Al asignar memoria dinámica, se debe tener cuidado de liberar la memoria cuando ya no sea necesaria para evitar fugas de memoria. Además, es esencial evitar el acceso a memoria liberada o liberar la misma memoria más de una vez, ya que esto puede llevar a comportamientos indefinidos en el programa.

Introducción a las listas enlazadas:

Las listas enlazadas son estructuras de datos fundamentales en programación y se
utilizan para almacenar y organizar elementos de manera dinámica. A diferencia de los arreglos estáticos, las listas enlazadas permiten la inserción y eliminación eficiente de elementos en cualquier posición, incluso durante la ejecución del programa.

En una lista enlazada, cada elemento se conoce como nodo y consta de dos componentes principales: el valor o dato que almacena y un puntero que apunta al siguiente nodo en la secuencia. Esta estructura de nodos enlazados permite construir una secuencia de elementos enlazados de forma secuencial.

El primer nodo de la lista se denomina nodo cabeza o nodo inicial, y el último nodo apunta a un valor nulo o a un nodo centinela que indica el final de la lista. Esto proporciona una forma conveniente de recorrer la lista y realizar operaciones como la inserción y eliminación de elementos.

Las listas enlazadas se pueden clasificar en diferentes tipos, como listas enlazadas simples, listas enlazadas dobles y listas enlazadas circulares, según el número y la dirección de los punteros que conectan los nodos.

Las listas enlazadas ofrecen varias ventajas y desventajas. Por un lado, permiten una inserción y eliminación eficiente de elementos en cualquier posición, incluso en listas de gran tamaño. Además, no requieren una asignación de memoria continua, lo que facilita la gestión de la memoria en aplicaciones dinámicas.

UNIDAD 6

Introducción a la estructura de datos en C++

La estructura de datos es una parte fundamental de la programación que se encarga de organizar y almacenar datos de manera eficiente. En C++, existen varias estructuras de datos incorporadas que proporcionan diferentes formas de almacenamiento y manipulación de datos.

Algunas de las estructuras de datos básicas en C++ incluyen arreglos, listas enlazadas, pilas, colas, árboles, grafos y conjuntos. Estas estructuras de datos tienen características y funcionalidades específicas que se adaptan a diferentes necesidades de programación.

Los arreglos son estructuras de datos estáticas que almacenan elementos del mismo tipo de forma contigua en la memoria. Proporcionan acceso aleatorio a los elementos y son útiles cuando se conoce de antemano el tamaño y la cantidad de elementos a almacenar.

Las listas enlazadas son estructuras de datos dinámicas que consisten en nodos enlazados, donde cada nodo contiene un valor y un puntero al siguiente nodo. Estas estructuras permiten una inserción y eliminación eficiente en cualquier posición,

lo que las hace adecuadas para situaciones donde el tamaño de la lista puede cambiar.

Las pilas y las colas son estructuras de datos que siguen el principio de "último en
entrar, primero en salir" (LIFO) y "primero en entrar, primero en salir" (FIFO),
respectivamente. Las pilas permiten agregar y eliminar elementos solo en un extremo, mientras que las colas permiten agregar elementos al final y eliminar elementos del principio.

Los árboles y los grafos son estructuras de datos no lineales que se utilizan para representar relaciones jerárquicas o conexiones entre elementos. Los árboles tienen una estructura jerárquica con un nodo raíz y varios nodos hijos, mientras que los grafos pueden tener conexiones arbitrarias entre nodos.

Los conjuntos son estructuras de datos que almacenan elementos únicos sin un orden específico. Proporcionan operaciones eficientes para insertar, buscar y eliminar elementos sin permitir duplicados.

Estas son solo algunas de las estructuras de datos básicas disponibles en C++. Cada estructura tiene sus propias ventajas y desventajas, y su elección depende de los requisitos y las necesidades específicas del problema que se está resolviendo. El conocimiento de las estructuras de datos en C++ es esencial para diseñar y desarrollar programas eficientes y escalables.

Clases auto referenciadas

Las clases auto referenciadas, también conocidas como clases que contienen punteros a sí mismas, son un concepto importante en programación orientada a
objetos. En C++, una clase puede tener miembros que son punteros a objetos de la misma clase. Esto permite crear relaciones complejas y recursivas entre objetos de la misma clase.

Las clases auto referenciadas son especialmente útiles cuando se desea modelar estructuras de datos o relaciones que contienen referencias mutuas. Un ejemplo común es la implementación de estructuras de datos como árboles, listas enlazadas o grafos, donde cada nodo o elemento tiene una referencia a otros nodos de la misma estructura.

Para lograr esto, se utiliza un puntero miembro en la clase que apunta a un objeto de la misma clase. Este puntero se puede utilizar para acceder y manipular los miembros y métodos del objeto referenciado. Al crear y trabajar con objetos de una clase auto referenciada, es importante tener en cuenta el manejo adecuado de la memoria y evitar bucles infinitos o referencias nulas.

Asignación dinámica de memoria y estructura de datos

La asignación dinámica de memoria es un concepto fundamental en la estructura de datos en C++. Permite reservar y liberar memoria durante la ejecución del programa, lo que es especialmente útil cuando se trabaja con estructuras de datos de tamaño variable o cuando la cantidad de datos no se conoce de antemano.

La asignación dinámica de memoria se realiza utilizando los operadores new y delete. El operador new se utiliza para asignar memoria dinámicamente, mientras que el operador delete se utiliza para liberar la memoria asignada.

Al trabajar con estructuras de datos dinámicas, como listas enlazadas, árboles o grafos, es común utilizar la asignación dinámica de memoria para crear y manipular nodos o elementos de la estructura. Esto permite una gestión eficiente de la memoria y la capacidad de adaptarse a cambios en la estructura de datos durante la ejecución del programa.

La asignación dinámica de memoria también se utiliza en combinación

con
estructuras de datos estáticas, como arreglos. Por ejemplo, se puede asignar dinámicamente un arreglo de tamaño variable según la entrada del usuario o según las necesidades del programa.

En resumen, la asignación dinámica de memoria es una herramienta poderosa en la estructura de datos en C++. Permite reservar y liberar memoria durante la ejecución del programa, lo que facilita la implementación de estructuras de datos de tamaño variable y proporciona flexibilidad en la gestión de la memoria. Sin embargo, se debe tener cuidado para evitar fugas de memoria y garantizar una correcta liberación de la memoria asignada dinámicamente.

Pilas y colas

Las pilas y colas son estructuras de datos fundamentales utilizadas en programación para organizar y manipular conjuntos de elementos de manera eficiente.

Una pila es una estructura de datos que sigue el principio de "último en entrar,
primero en salir" (LIFO, por sus siglas en inglés). Esto significa que el último elemento insertado en la pila es el primero en ser eliminado. Se asemeja a una pila de objetos físicos, donde el último objeto colocado en la pila es el primero en ser retirado.

Las pilas se utilizan en una variedad de escenarios, como la implementación de funciones recursivas, la evaluación de expresiones matemáticas y el seguimiento de la ejecución de un programa en forma de llamadas a funciones

Por otro lado, una cola es una estructura de datos que sigue el principio de "primero en entrar, primero en salir" (FIFO, por sus siglas en inglés). En una cola, el primer elemento insertado es el primero en ser eliminado. Esto se asemeja a una

cola de personas esperando en un mostrador, donde la persona que llega primero es la primera en ser atendida.

Las colas se utilizan en situaciones donde se requiere procesar elementos en el mismo orden en que llegaron, como en sistemas de gestión de tareas, planificación de procesos y simulaciones.

Tanto las pilas como las colas se pueden implementar utilizando arreglos estáticos, listas enlazadas u otras estructuras de datos subyacentes. Las operaciones básicas en una pila incluyen la inserción de un elemento en la parte superior (push) y la eliminación del elemento superior (pop). En una cola, las operaciones básicas son la inserción de un elemento al final (enqueue) y la eliminación del elemento del frente (dequeue)