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

Nombre Completo	Carné
RUDY JOSUE QUINTANILLA SANCHEZ	1290-22-2121

Unidad 1

- **Lenguaje C++**

C++ es un lenguaje de programación de alto nivel diseñado en 1979 por Bjarne Stroustrup es un lenguaje derivado de C (Dennis Ritchie). Se considera a C++ como un lenguaje de programación multiparadigma (programación estructurada, programación orientada a objetos entre otros).

- **Herramientas de desarrollo para C++**

Un IDE o Entorno de Desarrollo Integrado es una aplicación que nos proporciona todas las herramientas para que un desarrollador pueda programar en un determinado lenguaje de programación:

- ➢ Dev-C++
- ➢ Code: : Blocks
- ➢ Visual Studio
- ➢ CLion
- ➢ Zinjal
    - **Editores para C++**

Un editor de código fuente es un editor de texto diseñado específicamente para editar el código fuente de programas informáticos. Puede ser una aplicación individual o estar incluido en un entorno de desarrollo

- ➢ Visual Studio Code
- ➢ Sublime Text
- ➢ Atom

Línea de comandos

- **Tipos de datos nativos en C++**

Tipos primitivos de datos:

Los tipos de datos más simples son los tipos de datos primitivos, también denominados datos atómicos porque no se construyen a partir de otros tipos y son entidades únicas no descomponibles en otros.

Tipos de Datos Compuestos y Agregados:

Los datos compuestos son el tipo opuesto a los tipos de datos atómicos. Los datos compuestos se pueden romper en subcampos que tengan significado.
Existen tres tipos agregados básicos:

1. Arrays (arreglos)
2. Secuencias
3. Registros.

Array o Arreglos:

Es una colección de datos de tamaño o longitud fija ejemplo:
array de enteros: [4, 6, 8, 35, 46, 0810]

Secuencia o Cadena

Es en esencia, un array cuyo tamaño puede variar en tiempo de ejecución:
Arraydinamico = "longitude variable de un array en tiempo de ejecucion"
Un Registro: puede contener elementos datos agregados y primitivos.

Registro { Dato1 Dato2 Dato3 ... }

- Bibliotecas en C++

Bibliotecas en C++

En C++, se conoce como librerías (o bibliotecas) a cierto tipo de archivos que podemos importar o incluir en nuestro programa. Estos archivos contienen las

especificaciones de diferentes funcionalidades ya construidas y utilizables que podremos agregar a nuestro programa.

- Entrada y salida de flujos

Entrada y Salida estándar en C++:

Un programa en C++ puede realizar operaciones de entrada y salida de varias formas distintas.

Salidas con cout

cout << nombre << "armando cardona\n";

\n → nueva línea

\t → tabulación horizontal.

\\ → diagonal invertida

\" → comillas dobles

Entrada con cin

cin >> nombres;

Funciones miembro o métodos:

get: permite a un programa leer un carácter de entrada y guardarlo en una variable de tipo char. Está función toma un argumento, que debe ser una variable de tipo char.

ignore: este método permite descartar caracteres existentes en el buffer de entrada.

- **Programación estructurada**

Es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while)

- **Programación orientada a objetos:**
    - **Conceptos básicos.**

Se trata de un paradigma de la programación que pretende innovar la forma en la que se obtienen los resultados y modula el código para hacerlo más:

- Reutilizable
- Organizado
- Sencillo de mantener

En este tipo de programación se organiza el código en clases y, posteriormente, se crean los objetos. Una clase es una plantilla que define de manera general cómo serán los objetos de un tipo concreto. Por ejemplo, imaginemos la clase “Bicicletas”. Todas las bicicletas tienen características comunes: dos ruedas, un sillín, un manillar, dos pedales…, lo que en programación conocemos como “Atributos”.

En programación, las clases son todos los tipos de datos que el programador establece previamente y que sirven como modelo o plantilla para cualquier objeto que se incluya posteriormente en el software.

- **Propiedades de la programación orientada a objetos**
    - **Abstracción**

Es el proceso mediante el que se definen todos los atributos y métodos de una clase.

    - **Encapsulamiento,**

Protege los datos de manipulaciones no autorizadas.

    - **Polimorfismo,**

Permite dar la misma orden a varios objetos de distintas clases.

- **Herencia**

  Las clases pueden heredar los atributos y métodos partiendo de una clase base hacia una clase derivada.

- **Clase y super clase**

  Sería la clase que tiene todos los atributos y métodos en común, y las subclases van a ser las que heredan de la primera, con ello establecemos un sistema de jerarquía de clases, donde a mayor especificación se van bajando los niveles y no hay límite, podemos tener un clase que herede de otra clase que a su vez ha heredado de otra más, con ello podemos ir traspasando atributos comunes a una clase inferior que será muy específica pero que puede compartir con otros objetos una serie de funcionalidades.

- **Sobrecarga de operadores**

  La sobrecarga de operadores es la forma en que se pueden implementar operadores en tipos definidos por el usuario. Utiliza una lógica personalizada basada en el tipo y la cantidad de argumentos que pasamos.

- **Propiedades y métodos de una clase**

  Un **método** es una *Procedure* escrita específicamente para manipular los objetos encontrados en la clase.

  Una **propiedad** es un elemento de clase que puede ser utilizado directamente por su nombre como miembro y para el cual las operaciones de asignación de valor y recuperación desencadenan la ejecución de una Process.

- **Creación de objetos**

  Mediante la sintaxis literal de objetos esto es realmente sencillo. Todo lo que tienes que hacer es lanzar tus pares clave-valor separados por ':' dentro de un conjunto de llaves ({ }) y tu objeto está listo para ser servido (o consumido), como abajo: const person = { firstName: 'testFirstName', lastName: 'testLastName' };

Unidad 2

- **Operadores lógicos**

  Son los operadores utilizados para realizar operaciones lógicas. Los operadores lógicos devuelven un resultado booleano, y son tres: ***AND, OR y NOT.***

  And → Se representa como "**&&**" y dice que si ambos son verdaderos, entonces el resultado es **verdadero.**
  Or → Se representa como"||**"** y el resultado sera verdadero si uno de los dos es verdadero.

  Not/ Negación → Se representa como "**!variable**" y siempre devolverá una negación de la variable. variable = **falso.**

- **Estructuras de control**

  Estas nos permiten modificar el flujo en el que se ejecutan todas las instrucciones de un programa. Las estructuras de control son tres: de Secuencia, de Selección y de repetición.

  Las estructuras de secuencia ejecutan todas las instrucciones de forma lineal, una después de otra en el orden que se encuentran en el programa. Es la que tenemos por defecto en un programa.

  Como parte de las estructuras de selección tenemos:

  - **IF**

  Si la condición es verdadera realiza una acción, en cambio si es falsa la evita. Es una estructura de selección simple.

  - **IF anidados,**

  Son ciclos **IF** uno dentro de otro, haciendo que si la condición sea verdadera, vuelva a revisar otra condición dentro de la condición inicial.

- **SWITCH (Break y continue)**

Realiza una de entre varias acciones distintas, dependiendo del valor de una expresión. Es similar al ciclo if, pero con la diferencia que es una estructura de selección múltiple, porque se puede escoger entre muchas opciones.

Como parte de las estructuras de control del tipo de repetición nos encontramos con las siguientes:

- **FOR**

Ejecutar un grupo de sentencias un número determinado de veces.

- **WHILE**

Ejecutar un grupo de sentencias sólo cuando se cumpla una condición.

- **Diferencia entre operadores de igualdad y de asignación**

Los operadores de igualdad y asignación son conceptos diferentes en la programación y se utilizan para propósitos distintos.

El operador de igualdad se utiliza para comparar dos valores y determinar si son iguales. En la mayoría de los lenguajes de programación, el operador de igualdad se representa mediante el símbolo "==". Por ejemplo:

```
x = 5

y = 10


if x == y:

    print("x es igual a y")

else:
```

```
print("x no es igual a y")
```

En este ejemplo, el operador de igualdad "==" compara los valores de las variables "x" y "y". Si los valores son iguales, se ejecuta el código dentro del bloque "if"; de lo contrario, se ejecuta el código dentro del bloque "else".

- **Recursividad**

  Es una técnica de programación que consiste en que una función se llame a sí misma.

- **Manejo de excepciones**

  Es la manera en la que tratamos de capturar y tratar los errores o excepciones que se nos pueden presentar durante la ejecución y compilación de un programa.

- **Procesamiento de archivos**

  Se refiere a la lectura y escritura de datos en archivos utilizando flujos de archivo.

  - **Archivos y flujos**

  Para poder trabajar con ficheros o archivos en C++ debemos utilizar algunas librerías como ***fstream, ifstream, ofstream.*** Los archivos son muy utilizados en C++, sobre todo para almacenar y recuperar datos de forma permanente en el sistema de almacenamiento en nuestra computadora.

  Los flujos nos ayudan a manejar los archivos. Esto es posible gracias a que nos proporcionan las herramientas necesarias para leer y escribir datos *desde y hacia* los archivos. Entonces podemos encontrar los tres principales los cuales son: ***Flujos de entrada, de salida*** y ***Flujos de Entrada/Salida***.

  Flujos de entrada: Nos permiten poder leer los datos desde un archivo hacia el programa. Osea le da acceso al programa para entrar en el archivo.

  Flujos de Salida: Este Flujo por el contrario que el anterior, nos permite modificar y escribir datos para posteriormente mandarlos desde el programa hacia un archivo.

  Flujos de Entrada/salida: Es la combinación de los dos anteriores, para poder leer y escribir sobre un archivo.

Unidad 3

- **Arreglos y vectores**

  Se dividen en 2 grupos, los vectores y las matrices. Los vectores son arreglos que contienen una sola dimensión y las matrices 2 o más dimensiones. Los Arreglos se utilizan para almacenar un conjunto de variables, que sean del mismo tipo de dato, y todas estas bajo un mismo nombre.

- **Declaración y creación de arreglos**

  Los arreglos se crean con la palabra new, para crearlo el programador debe especificar el tipo y el número de elementos que almacenará. La declaración y la creación del arreglo se puede hacer en una sola línea, de la siguiente manera String Nombres [] = new String [5];

  En c++, se pueden crear string <nombre> [<tamaño arreglo>];

- **Ejemplos del uso de arreglos**

  Los arreglos sirven para muchas cosas pero se pueden utilizar para guardar información dentro de la memoria temporal, se utilizan los loops para recorrer la información o guardarla en sus distintos espacios depende de cuanto se haya programado.

- **Arreglos a funciones**

  Se pueden pasar arreglos a funciones de diferentes maneras:

  por ejemplo, se puede pasar por medio de punteros

```
void funcion(int* arreglo, int tamano) {
   // Acceder a los elementos del arreglo usando notación de punteros
   for (int i = 0; i < tamano; i++) {
      cout << arreglo[i] << " ";
   }
}
```

```
int main() {
    int arreglo[] = {1, 2, 3, 4, 5};
    int tamano = sizeof(arreglo) / sizeof(arreglo[0]);
    funcion(arreglo, tamano);
    return 0;
}
```

Puntero constante

```
void funcion(const int* arreglo, int tamano) {
    // Acceder a los elementos del arreglo usando notación de punteros
    for (int i = 0; i < tamano; i++) {
        cout << arreglo[i] << " ";
    }
}

int main() {
    int arreglo[] = {1, 2, 3, 4, 5};
    int tamano = sizeof(arreglo) / sizeof(arreglo[0]);
    funcion(arreglo, tamano);
    return 0;
}
```

Referencia a un arreglo

```
void funcion(int (&arreglo)[5]) {
    // Acceder a los elementos del arreglo directamente
    for (int i = 0; i < 5; i++) {
        cout << arreglo[i] << " ";
    }
}

int main() {
```

```
    int arreglo[] = {1, 2, 3, 4, 5};
    funcion(arreglo);
    return 0;
}
```

- **Búsqueda de datos en arreglos**

  Se pueden buscar por medio de loops ya sea por ciclos for, while, do while.

  Estos se recorren por medio de una iteración por ejemplo, se declara una variable i. Está inicializada a 0.

```
int numeros[100];
for(int i=0; i<0; i++){
    if(numero[i]==1){
        cout<<"se encuentra 1 en el arreglo";
}
};
```

- **Ordenamiento de arreglos**

  El ordenamiento de arreglos es el proceso de reorganizar los elementos de un arreglo en un orden específico, ya sea ascendente o descendente, de acuerdo con algún criterio definido. El objetivo principal del ordenamiento es facilitar la búsqueda, la recuperación y el procesamiento eficiente de los datos.

  Hay una variedad de algoritmos de ordenamiento disponibles, cada uno con su propia complejidad y eficiencia. Aquí tienes algunos conceptos clave relacionados con el ordenamiento de arreglos:

  Elementos comparables: Para ordenar un arreglo, los elementos deben ser comparables entre sí. Esto significa que se debe definir una relación de orden o una función de comparación que pueda determinar si un elemento es mayor, menor o igual a otro.

Orden ascendente y descendente: El orden ascendente implica que los elementos se organizan de menor a mayor, mientras que el orden descendente implica que se organizan de mayor a menor.

Estabilidad: Un algoritmo de ordenamiento es estable si preserva el orden relativo de elementos con claves iguales. Es decir, si hay dos elementos con la misma clave y aparecen en un orden específico en el arreglo original, el algoritmo de ordenamiento estable mantendrá ese orden en el arreglo ordenado.

Complejidad del tiempo: La complejidad del tiempo se refiere al tiempo de ejecución necesario para ordenar un arreglo. Se mide generalmente en términos de mejor caso, peor caso y caso promedio. La complejidad del tiempo puede ser expresada en notación Big O, como O(n^2) para algoritmos de ordenamiento cuadráticos o O(n log n) para algoritmos más eficientes.

Algoritmos de ordenamiento populares: Hay una amplia gama de algoritmos de ordenamiento, cada uno con sus propias ventajas y desventajas. Algunos de los algoritmos más conocidos incluyen: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort y Heap Sort.

- **Arreglos multidimensionales**

  Puede crear un arreglo multidimensional creando una matriz de dos dimensiones en primer lugar y, después, ampliándola. Por ejemplo, defina primero una matriz de 3 por 3 como la primera página de un arreglo 3D.

- **Búsqueda y ordenamiento.**
  - **Algoritmos de búsqueda**

La búsqueda es un concepto fundamental en ciencias de la computación y se utiliza para encontrar un elemento específico dentro de una colección de datos. Existen diferentes

algoritmos de búsqueda que varían en su eficiencia y en la forma en que acceden y comparan los elementos de la colección.

1. **Búsqueda lineal:** Este algoritmo recorre secuencialmente cada elemento de la colección hasta encontrar el elemento buscado o llegar al final. Es sencillo de implementar, pero puede ser lento para grandes conjuntos de datos.

2. **Búsqueda binaria:** Este algoritmo requiere que la colección esté ordenada previamente. Comienza comparando el elemento buscado con el elemento del medio de la colección. Si son iguales, se encontró el elemento. Si el elemento buscado es menor, se realiza una búsqueda en la mitad inferior de la colección; de lo contrario, se realiza una búsqueda en la mitad superior.

   Este proceso se repite hasta encontrar el elemento o determinar que no está presente. La búsqueda binaria es eficiente para conjuntos de datos ordenados, ya que reduce el espacio de búsqueda a la mitad en cada iteración.

3. **Búsqueda hash**: Este enfoque utiliza una función de hash para calcular la ubicación del elemento buscado dentro de una estructura de datos especial llamada tabla hash. Si la función de hash está bien diseñada y la tabla hash tiene un buen factor de carga, la búsqueda puede ser muy eficiente, con un tiempo de búsqueda promedio constante. Sin embargo, la búsqueda hash requiere que los elementos sean *hasheables* y puede haber colisiones si dos elementos diferentes consiguen la misma ubicación en la tabla hash.

   - **Algoritmos de ordenamiento.**

   Los algoritmos de ordenamiento son algoritmos utilizados para organizar una colección de elementos en un orden específico, como de forma ascendente o descendente. Ordenar los elementos es una tarea común en ciencias de la computación y es fundamental para muchas aplicaciones.

1. ***Ordenamiento de burbuja (Bubble Sort):*** Compara repetidamente pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que la lista esté completamente ordenada.

2. ***Ordenamiento por selección (Selection Sort):*** Encuentra repetidamente el elemento mínimo de la lista y lo coloca al principio. El proceso se repite para el resto de la lista hasta que esté completamente ordenada.

3. ***Ordenamiento por inserción (Insertion Sort):*** Construye una lista ordenada insertando elementos uno a uno en la posición correcta. En cada iteración, se toma un elemento de la lista no ordenada y se inserta en su posición correcta dentro de la lista ordenada.

4. ***Ordenamiento por mezcla (Merge Sort):*** Divide recursivamente la lista en sublistas más pequeñas, las ordena por separado y luego las combina en una lista ordenada más grande. Utiliza el enfoque "divide y conquista".

5. ***Ordenamiento rápido (Quick Sort):*** Selecciona un elemento como pivote y reorganiza la lista de manera que los elementos más pequeños que el pivote estén antes y los elementos más grandes estén después. Luego, se aplica recursivamente este proceso a las sublistas antes y después del pivote.

   Estos son solo algunos ejemplos de algoritmos de ordenamiento. Cada algoritmo tiene su propia complejidad y características, lo que los hace adecuados para diferentes situaciones. La elección del algoritmo de ordenamiento dependerá del tamaño de los datos, el tiempo de ejecución esperado, los recursos disponibles y otros factores específicos del problema.

   - **Motores de bases de datos**

   MySQL

   Es un sistema de gestión de bases de datos relacional de código abierto. Es ampliamente utilizado y conocido por su confiabilidad, rendimiento y facilidad de uso. MySQL es compatible con múltiples plataformas y es utilizado en una amplia gama de aplicaciones, desde sitios web pequeños hasta grandes sistemas empresariales.

PostgreSQL

También es un sistema de gestión de bases de datos relacional de código abierto. Se destaca por su capacidad de manejar grandes volúmenes de datos y su soporte para características avanzadas, como integridad referencial, transacciones ACID y consultas complejas. PostgreSQL es altamente personalizable y es utilizado en aplicaciones tanto pequeñas como grandes.

Oracle Database

Es un sistema de gestión de bases de datos relacional desarrollado por Oracle Corporation. Es conocido por su escalabilidad, seguridad y capacidad de manejar grandes cargas de trabajo empresariales. Oracle Database ofrece una amplia gama de características y opciones avanzadas, lo que lo convierte en una opción popular para aplicaciones de misión crítica.

Microsoft SQL Server

Es un sistema de gestión de bases de datos relacional desarrollado por Microsoft. Está diseñado para entornos Windows y se integra estrechamente con otras tecnologías de Microsoft, como .NET Framework. Microsoft SQL Server ofrece herramientas de administración y desarrollo robustas, así como capacidades de análisis de datos y business intelligence.

MongoDB

Es una base de datos NoSQL orientada a documentos. En lugar de utilizar tablas y filas, MongoDB almacena datos en documentos flexibles tipo JSON con esquemas dinámicos. Es altamente escalable y puede manejar grandes volúmenes de datos no estructurados. MongoDB es popular en aplicaciones web, análisis de datos y casos de uso de tiempo real.

- **Comparaciones características**

MySQL:

Modelo de datos: Relacional.
Lenguaje de consulta: SQL.
Licencia: Código abierto (Community Edition) y licencia comercial (Enterprise Edition).
Escalabilidad: Buena escalabilidad vertical y horizontal.
Soporte de transacciones: Sí, con soporte para transacciones ACID.
Replicación: Soporte para replicación maestro-esclavo y multidireccional.
Flexibilidad: Admite una variedad de sistemas operativos y lenguajes de programación.

PostgreSQL:

Modelo de datos: Relacional.
Lenguaje de consulta: SQL.
Licencia: Código abierto (PostgreSQL License).
Escalabilidad: Buena escalabilidad vertical y horizontal.
Soporte de transacciones: Sí, con soporte para transacciones ACID.
Replicación: Soporte para replicación maestro-esclavo y multidireccional.
Características avanzadas: Integridad referencial, consultas complejas, tipos de datos personalizados y funciones almacenadas.

Oracle Database:
Modelo de datos: Relacional.
Lenguaje de consulta: SQL.
Licencia: Comercial.
Escalabilidad: Excelente escalabilidad vertical y horizontal.
Soporte de transacciones: Sí, con soporte para transacciones ACID.
Replicación: Soporte para replicación maestro-esclavo y multidireccional.
Características avanzadas: Alta disponibilidad, particionamiento, seguridad avanzada, análisis de datos y rendimiento optimizado.

Microsoft SQL Server:

Modelo de datos: Relacional.
Lenguaje de consulta: SQL.
Licencia: Comercial (varias ediciones disponibles).
Escalabilidad: Buena escalabilidad vertical y horizontal.
Soporte de transacciones: Sí, con soporte para transacciones ACID.
Replicación: Soporte para replicación maestro-esclavo y multidireccional.
Integración con tecnologías de Microsoft: Estrecha integración con .NET Framework, herramientas de desarrollo y plataformas de Microsoft.

MongoDB:

Modelo de datos: NoSQL (orientado a documentos).
Lenguaje de consulta: MongoDB Query Language (MQL).
Licencia: Código abierto (Server Side Public License) y licencia comercial (MongoDB Enterprise Advanced).
Escalabilidad: Excelente escalabilidad horizontal.
Soporte de transacciones: Sí, con soporte para transacciones ACID en colecciones específicas.
Flexibilidad de esquema: Esquema flexible y dinámico.
Características avanzadas: Alta disponibilidad, particionamiento automático, índices y consultas rápidas en documentos JSON.

Unidad 4

- SQL (DML y DDL)

DML

Su nombre significa *Lenguaje de manipulación de datos* y es un sub-lenguaje de consulta y manipulación de datos. Se entenderá por manipulación de datos a la:

1. Recuperación de información.
2. Inserción de nueva información.
3. Eliminado de información existente.
4. Modificación de Información Almacenada.

Select: Para obtener datos de una base de datos. Ejemplo:

SELECT columna1, columna2 **FROM** tabla **WHERE** condición**;**

Insert: Para insertar datos en una tabla. Ejemplo:

INSERT INTO tabla **(**c1, c2**) VALUES (**v1, v2**);**

Update: Para modificar datos existentes. Por ejemplo:

UPDATE tabla **SET** columna = valor **WHERE** condición**;**

Delete: Permite eliminar datos en una tabla. Ejemplo:

DELETE FROM tabla **WHERE** condición**;**

DDL

Significa *lenguaje de definición de datos* y es el archivo que se consulta cada vez que se consulta toda vez se quiere leer, modificar, o eliminar los datos de una base de datos. Se utilizan para crear y modificar la estructura de las tablas así como otros objetos de la base de datos.

Se utilizan las sentencias:

Create: Para crear objetos en la base de datos. **Ejemplo:**

```
CREATE TABLE tabla (
    columna1 tipo_de_dato,
    columna2 tipo_de_dato,
```

...
);

Alter: Para modificar algún registro de una tabla. **Ejemplo:**

ALTER TABLE tabla ADD COLUMN nueva_columna tipo_de_dato;

Drop: Para eliminar una tabla. **Ejemplo:**

DROP TABLE tabla;

Truncate: Eliminar todos los registros de una tabla. **Ejemplo:**

TRUNCATE TABLE nombre_tabla;

Unidad 5

- **Memoria dinámica**
    - **Declaración e inicialización de punteros**

      Un puntero es una variable que almacena la dirección de memoria de otro objeto. Para declarar un puntero, se utiliza el tipo de dato al que apunta, seguido del símbolo de asterisco (*). La inicialización de un puntero implica asignarle la dirección de memoria de otro objeto utilizando el operador de dirección (&).

    - **Arrays de punteros**

      Un array de punteros es una estructura que contiene varios punteros. Cada elemento del array es un puntero que puede apuntar a una dirección de memoria diferente. Esto permite almacenar y manipular múltiples direcciones de memoria en un solo array.

    - **Aritmética de punteros**

      La aritmética de punteros se refiere a la capacidad de realizar operaciones matemáticas en los punteros. Por ejemplo, se puede incrementar o decrementar un puntero, lo que se traduce en moverse a través de las direcciones de memoria contiguas.

    - **Operador Sizeof**

      El operador sizeof se utiliza para determinar el tamaño en bytes de un tipo de dato o una variable. Se puede aplicar tanto a tipos de datos estáticos como a punteros, y su resultado depende de la arquitectura del sistema.

    - **Relación entre apuntadores y arreglos**

      En C y C++, los arreglos se pueden tratar como punteros debido a su estrecha relación. El nombre de un arreglo es un puntero constante que apunta a la primera posición de memoria del arreglo. Esto permite acceder a los elementos del arreglo utilizando aritmética de punteros.

- **Apuntadores a funciones**

  Los punteros a funciones son variables que almacenan direcciones de funciones en lugar de datos. Pueden ser utilizados para invocar funciones a través del puntero, lo que brinda flexibilidad y la posibilidad de cambiar la función que se ejecuta en tiempo de ejecución.

- **Asignación de memoria dinámica**

  La asignación de memoria dinámica permite reservar espacio en la memoria durante la ejecución de un programa. Se utiliza la función malloc o new (en C++) para asignar memoria dinámica y obtener un puntero que apunta a ella. Es responsabilidad del programador liberar la memoria cuando ya no se necesita utilizando la función free o delete (en C++).

- **Introducción a listas enlazadas (Conceptos)**

  Las listas enlazadas son estructuras de datos dinámicas que consisten en nodos enlazados entre sí. Cada nodo contiene un valor y un puntero que apunta al siguiente nodo de la lista. Esto permite la creación de estructuras de datos flexibles que pueden crecer o reducirse dinámicamente según sea necesario.

Unidad 6

- **Introducción a la estructura de datos en C++**

  Una estructura de datos es una combinación de elementos en la que cualquiera de estos representa datos y sus operaciones relacionadas. Por ejemplo: Cualquier número de tipo entero o un número flotante almacenado en mi computadora, es una estructura de datos.

- **Clases auto referenciadas**

  Las clases auto referenciadas, también conocidas como clases que se referencian a sí mismas o clases recursivas, son clases en las que un miembro de la clase es un puntero a un objeto de la misma clase. Esto permite que un objeto contenga referencias a otros objetos de su mismo tipo.

  La capacidad de una clase para referenciarse a sí misma es una característica poderosa y flexible en la programación orientada a objetos, y se utiliza en diversas estructuras de datos y algoritmos. Un ejemplo común de una clase auto referenciada es la estructura de datos de una lista enlazada.

  La lista enlazada es una estructura de datos dinámica que consta de nodos enlazados entre sí. Cada nodo contiene un dato y un puntero que apunta al siguiente nodo de la lista. En este caso, la clase nodo se refiere a sí misma, ya que tiene un miembro puntero que apunta a un objeto de la misma clase (el siguiente nodo en la lista).

  - **Listas enlazadas**

  Las listas enlazadas son estructuras de datos dinámicas utilizadas para almacenar y organizar datos de manera flexible. A diferencia de los arreglos estáticos, las listas enlazadas pueden crecer o reducirse en tamaño durante la ejecución del programa, lo que las hace ideales para manejar conjuntos de datos cambiantes.

Una lista enlazada está compuesta por una serie de nodos, donde cada nodo contiene un valor y un puntero que apunta al siguiente nodo en la lista. Cada nodo se representa como una estructura que contiene dos componentes principales: el dato o valor almacenado y el puntero al siguiente nodo.

La ventaja principal de las listas enlazadas radica en su capacidad para insertar y eliminar elementos de manera eficiente en cualquier posición de la lista. Esto se debe a que no es necesario desplazar o reorganizar los elementos contiguos como en los arreglos estáticos. En cambio, se ajustan los punteros para redirigir el flujo de la lista.

- **Asignación dinámica de memoria y estructura de datos**

  Las estructuras de datos consisten básicamente en cualquier tipo de dato, como un número, una variable o una cadena de texto.

  La asignación de memoria dinámica trata acerca de la gestión de la memoria del sistema mediante el uso de estructuras de datos dinámicas. Permite reservar y liberar memoria durante la ejecución de un programa, lo cual es útil cuando se necesita gestionar cantidades variables de datos o cuando se requiere almacenar datos de forma dinámica.

  La asignación dinámica de memoria es especialmente útil cuando se trabaja con estructuras de datos dinámicas, como listas enlazadas, pilas, colas o árboles, ya que estas estructuras pueden crecer o reducirse según sea necesario durante la ejecución del programa.

- **Pilas y colas**
  - **Pilas**

Es una estructura de datos de tipo LIFO. Es decir que el último elemento en entrar es el primero en salir. Los datos interactúan por medio del mismo extremo.

- **Colas**

Una cola es una lista lineal en la cual los datos se insertan por un extremo y

se extraen por el otro extremo. Dicho en otras palabras es una estructura FIFO → Primero en entrar, primero en salir.