**ASIGNACIÓN DE MEMORIA DINÁMICA EN C++**

**Conceptos básicos**

La estructura de la memoria principal de la computadora se divide **esencialmente en: memoria estática ocupada para las variables** y la memoria dinámica que se gestiona con los operadores **new** y **delete,** y que se denomina también **montículo (heap).**

La memoria dinámica funciona mediante la reserva y liberación de trozos de memoria voluntad del programador. En ausencia de un “recolector automático de basura” o **garbage recolector**, la memoria dinámica permanece ocupada hasta que el programador la libera explícitamente, devolviéndola a la zona de memoria dinámica libre, donde puede volver a ser utilizada.

La asignación de memoria dinámica consiste en asignar la cantidad de memoria necesaria para almacenar un objeto durante la ejecución del programa, en vez de hacerlo en el momento de la compilación de este.

La memoria dinámica en C++ se refiere a la asignación y liberación de memoria durante la ejecución de un programa en lugar de en tiempo de compilación. Esto permite a un programa reservar la cantidad exacta de memoria que necesita en un momento dado y liberarla cuando ya no la necesita.

En C++ puedes trabajar con memoria dinámica utilizando operadores como new y delete.

La reserva de memoria (tamaño) se realiza con new y la liberación cuando ya no se necesita se realiza con el operador delete, que deja libre el espacio ocupado para una posterior utilización.

De igual forma new asigna memoria para un objeto y delete destruye y libera la memoria ocupada por el objeto.

Esto implica declarar un puntero a un tipo de datos igual al tipo de objeto que se quiere asignar dinámicamente. Por ejemplo, si queremos asignar memoria dinámicamente a un entero.

Int \*p = new int; // Reserva memoria para un entero

\*p = 42; // Asigna un valor a la memoria reservada

Delete p; // Libera la memoria asignada a través de ´new´.

---Ejemplo---

**Verificación de errores**

Es importante verificar si la asignación de memoria dinámica tuvo éxito, new puede lanzar una excepción **std::bad\_aloc** si no se puede asignar memoria, para ello deberá usar un bloque try-catch para manejar una excepción.

---ejemplo----

**Uso seguro de punteros**

Asegúrate de usar los punteros correctamente y evitar problemas como punteros colgantes (danling pointers) y fugas de memoria. Puedes considerar el uso de punteros inteligentes, como **std::shared\_ptr** y **std::unique\_ptr,** para ayudar en la gestión segura de la memoria dinámica.

**Puntero colgante**

Es un puntero en el código que conduce al bloque de memoria incorrecto o a algún destino no deseado. En muchos casos, esto se debe a que el objeto original al que se apunta el puntero se ha eliminado, movido o reemplazado.

**Liberación de memoria**

Siempre libera la memoria dinámica cuando ya no la necesites para evitar fugas de memoria. Puedes hacerlo manualmente con delete o utilizar punteros inteligentes para que la memoria se libere automáticamente cuando ya no haya referencias a ella.

Es importante recordar que el uso incorrecto de la memoria dinámica puede causar problemas de seguridad y estabilidad en tu programa, como fugas de memoria o corrupción de datos. Por lo tanto, es esencial ser cuidadoso al trabajar con memoria dinámica en C++.

**Ventajas de la asignación dinámica de memoria en C++**

C++ ofrece un nuevo y mejor método para gestionar la asignación dinámica de memoria, los operadores **new y delete**, que asignan y liberan la memoria de una zona de memoria llamada almacén libre.

Los operadores **new y delete** son más versátiles que **malloc ()** y **free ()**,ya que ellos pueden asociar la asignación de memoria con el medio que lo utiliza. Son más fiables, ya que el compilador realiza verificación de tipos cada vez que un programa asigna memoria con new.

**New** y **delete** se implementan como operadores y no como funciones. Esto significa que se construyen en el propio lenguaje de modo que los programas pueden utilizar sin incluir ningún archivo de cabecera.

**New y delete** nuca requieren moldeado (conversión forzosa de tipos) de tipos y eso hace que sean más fáciles de utilizar que **malloc () y free ().**

Operadores new y delete

**New**

C++ proporciona un método para obtener bloques de memoria: el operador new. El operador new asigna un bloque de memoria que es d<el tamaño del tipo de dato.

El dato u objeto dato puede ser un int, un float, una estructura, un array o cualquier otro tipo de dato. El operador new devuelve un puntero que es la dirección del bloque asignado de memoria. El puntero se utiliza para referenciar el bloque de memoria.

**Delete**

El operador delete asegura un uso seguro y eficiente de la memoria, libera la memoria reservada con new.

…

**No se puede utilizar delete para liberar memoria creada por declaración de variables ordinarias.**

* Int \*p = new int; // correcto
* Delete p; // correcto
* Delete p; // no es correcto, ducplicar
* Int num = 10; // correcto
* Int \*pn = & num; // correcto
* Delete pn; // incorrecto, memoria no asignada por new

**ASIGNACIÓN Y LIBERACIÓN DE MEMORIA DINÁMICA**

**CLASE NODO EN C++**

Una clase nodo en C++ se utiliza comúnmente en estructuras de datos como listas enlazadas, arboles grafos y otras estructuras similares.

La clase nodo representa un elemento individual en una estructura de datos y generalmente contiene dos partes principales: un valor o un dato y un puntero o referencia al siguiente nodo en la secuencia (en el caso de listas enlazadas).

En algunas ocasiones se requiere de un segundo apuntador, como en el caso de las estructuras de datos no lineales como árboles y grafos.