**Estudiante:** Daniela Tellez Cobo

**Informe gestión de memoria en C++**

**Actividad 1:**

* **Implementación:** Se declaró una variable entera. Se obtuvo su dirección de memoria usando el operador &. Se utilizó un puntero para modificar indirectamente el valor de la variable original.
* **Concepto Clave:** Demostró el ciclo de vida de una variable básica en la memoria **Stack**. Estableció que cada variable tiene una dirección única y que los punteros son variables especiales diseñadas para *almacenar* esas direcciones, permitiendo la manipulación indirecta de datos.

**Actividad 2:**

* **Implementación:** Se utilizó un puntero para modificar una variable (como en la Actividad 1). Luego, se creó una referencia a esa misma variable y se usó para modificar su valor. Finalmente, se imprimieron las direcciones de memoria.
* **Concepto Clave:** Se demostró la diferencia fundamental entre punteros y referencias:
  + Un **Puntero** es una *variable separada* que ocupa su propio espacio en memoria y almacena una dirección. Puede ser nulo y puede ser "reapuntado" a otras variables.
  + Una **Referencia** es un *alias* para una variable existente. No ocupa su propia dirección de memoria (al imprimir &ref, se obtiene la dirección de la variable original). Debe ser inicializada al momento de su creación y no puede ser nula.

**Actividad 3:**

* **Implementación:** Se declaró un array de enteros. Se asignó un puntero al inicio del array. Se utilizó la **aritmética de punteros** para acceder y modificar elementos (ej. \*(ptr + 2) = 99; para cambiar el tercer elemento).
* **Concepto Clave:** Demostró la estrecha relación entre arrays y punteros en C++. El nombre de un array "decae" (se convierte) en un puntero a su primer elemento. Se comprobó que numeros[i] es sintácticamente equivalente a \*(numeros + i).

**Actividad 4:**

* **Implementación:** Se creó una matriz 2D de tamaño dinámico (definido en tiempo de ejecución). Esto requirió un puntero-a-puntero.
  1. Se usó new int\*[filas] para crear un array de punteros en el **Heap**.
  2. Se iteró ese array y se usó new int[columnas] para crear cada fila individual en el **Heap**.
  3. Se llenó la matriz con datos.
  4. Se liberó la memoria en orden inverso: primero cada fila (delete[] matriz[i];) y luego el array de punteros (delete[] matriz;).
* **Concepto Clave:** Esta fue la implementación central, introduciendo la memoria dinámica (**Heap**). A diferencia del Stack (automático), el Heap debe ser gestionado manualmente por el programador.

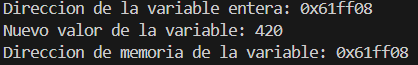
**Análisis General: Stack vs. Heap**

La progresión de las actividades ilustra perfectamente los dos principales segmentos de memoria en C++:

1. **El Stack (La Pila):**
   * **Dónde se usó:** Actividades 1, 2 y 3. También se usó en la Actividad 4 para almacenar las variables filas, columnas y el propio puntero int\*\* matriz.
   * **Características:** Es una memoria rápida, pequeña y **automática**. Las variables locales se crean aquí y se destruyen automáticamente cuando salen de su ámbito (scope).
2. **El Heap (El Montón):**
   * **Dónde se usó:** Exclusivamente en la Actividad 4.
   * **Características:** Es una gran reserva de memoria, más lenta pero mucho más flexible. Es **manual**. El programador es 100% responsable de solicitar memoria (new) y, crucialmente, de **liberarla** (delete).

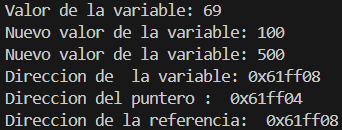
**Análisis de las salidas del programa**

**Actividad 1**

****

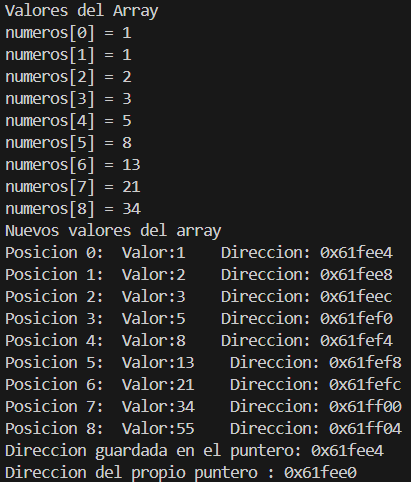
Se observa que la dirección de memoria es igual antes (0x61ff08) y después (0x61ff08) de la modificación. Esto demuestra que el puntero no creó una variable nueva. "Viajó" a la dirección de memoria original y cambió el valor (420) que estaba *dentro* de ese contenedor, sin mover el contenedor.

**Actividad 2**

****

La dirección de la variable original (0x61ff08) y la dirección de la referencia (0x61ff08) son **exactamente la misma**. Esto prueba visualmente que una referencia **es un alias** (un "apodo"). No es una variable nueva, es solo otro nombre para la variable original. La dirección del puntero (0x61ff04) es **diferente**. Esto prueba que el puntero (int\* ptr) es una **variable separada** que ocupa su propio espacio en la memoria (en la dirección 0x61ff04), y su *contenido* es la dirección 0x61ff08.

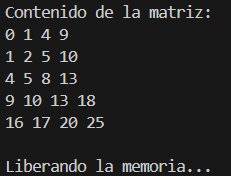
**Actividad 3**



**Las** direcciones del array van de 0x61fee4 a 0x61fee8 a 0x61feec. Esto prueba que un array es un **bloque de memoria único y contiguo**. El sistema sabe que un int ocupa 4 bytes (en tu máquina), por lo que para ir al siguiente elemento, simplemente suma 4 a la dirección anterior.

La Direccion guardada en el puntero (0x61fee4) es idéntica a la dirección de la Posicion 0 (0x61fee4). Esto confirma que cuando asignas un array a un puntero (int\* ptr = numeros;), el puntero simplemente almacena la dirección del **primer elemento** del array.

**Actividad 4**

****

La matriz se imprimió correctamente. Esto confirma que la lógica para crear el "array de punteros" (new int\*[filas]) y luego crear cada "fila" (new int[columnas]) funcionó perfectamente.

El programa imprimió "Liberando la memoria..." y luego **terminó limpiamente**. No se "colgó" (crasheó) ni imprimió la memoria basura que vimos en la conversación anterior. Esto indica que tus bucles delete[] liberaron correctamente toda la memoria.