**Aryam Lizette Buendía Zavala A01659465**

**Actividad Apuntadores y Memeoria**

**Caso 1: Stack**

Compilando y ejecutando el Código como está, salió esto: Texto

Descripción generada automáticamente

Tiene sentido porque si al inicio del programa el MAX\_SIZE es 100000, el cout del MAX\_SIZE/2 es 50000.

**Prueba 1:**

Después le agregué al MAX\_SIZE un 0 más y yo no ejecutó.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Ya no corrió el programa porque el stack ya está lleno.

**Prueba 2:** Probé ahora quitar ceros JAJAJA. A ver si me funcionaba.

Texto, Logotipo, Sitio web

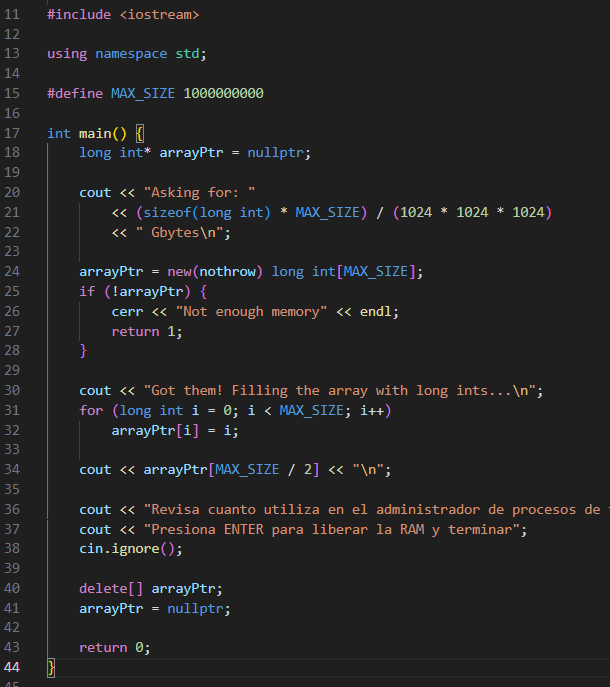
Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente

Con 10000 sí funcionó todo bien, y si quitaba ceros también sin problema.

**Caso 2: Heap**

****

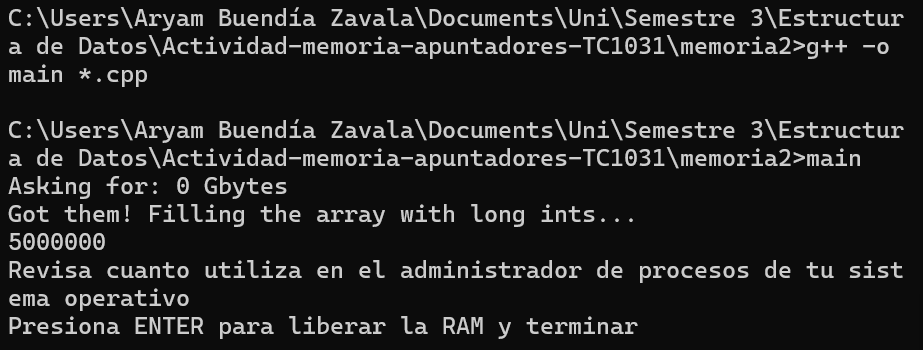
Al inicio, compilé y corrí el programa así como estaba y salió esto:

**Texto

Descripción generada automáticamente**

Ni siquiera me dejaba ejecutarlo, así que decidií bajar el MAX SIZE 2 ceros hasta que me dejara ejecutarlo.





Aquí ya vi que sí se ejecutó, así que a partir de aquí empecé a hacer los cambios solicitados.

Cuando cambié el long int por string me salió esto. No sale ningún número.

Texto

Descripción generada automáticamente

Regresé el string a long int y agregué un cero más. 

De plano con 100000000 es cuando funcionaba, con más ceros marcaba error.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Quité el nowthrow de new, compilé y ejecuté y pasó esto:

Texto

Descripción generada automáticamente

Lo mismo, muy grande el MAX SIZE.

Texto

Descripción generada automáticamente

Le quité un cero.

Texto

Descripción generada automáticamente

No sé qué pasa, desde la vez pasada no me sale nada kjasnkfs. Pero debería salir que no hay memoria, porque finalmente no me deja ejecutarlo por eso. Y con o sin el nothrow sale lo mismo. El nothrow nos ayuda a no lanzar excepciones (evitar usar el try y catch), y nos ayuda a mejorar la seguridad del código (a que se nos vaya un try o catch, vaya).

**Caso 2c:**

**Texto

Descripción generada automáticamente**

Al compilar y ejecutar el código de memoria2c, sale esto. Al inicio del código se declara un apuntador a un entero (int) y se está inicializando con el nullptr, esto para evitar que tenga basura o que haya confusión después. El \* indica que es apuntador y sin apuntador es la ubicación de memoria. Con el new(nothrow) se le está asignando memoria (new) y se están evitando lanzar excepciones (nothrow). Si el apuntador (ptr) es nullptr lanza un mensaje de que no hay memoria. Si no lanza error, significa que la memoria fue asignada correctamente y se le da el valor de 5 a la ubicación de memoria que apunta ptr. Los couts de los valores son:

Texto

Descripción generada automáticamente

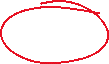
ptr (la dirección de memoria como tal)

\*ptr (el valor que está en esa dirección de memoria)

Si lo ejecuto otra vez, lo que cambia es la dirección de memoria.

Texto

Descripción generada automáticamente



**Caso 2d:**

**Imagen que contiene Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente**

Este programa nos muestra apuntadores en arreglos. Primero definimos el valor de MAX SIZE a 10 (el tamaño que tendrá el arreglo, (0 a 9)). Inicializamos el apuntador con int\* ptr = nullptr. Y lo mismo del ejemplo de arriba, inicializarlo y asignarle memoria. Con el ciclo for vaños llenando el arreglo (del 10 [0] al 1 [9]). En los couts teneos 4. El primero nos dice la dirección de memoria del apuntador, el segundo nos dice EL VALOR del 6to elemento del arreglo (que es 4), el tercero nos dice el primer valor del arreglo + 6 (10 + 6 = 16), y el último imprime ptr + 6 (o sea ptr[6]) que es 4. Si hubiera sido ptr + 9, sería ptr[9] y se imprimiría el 1. Aquí la demostración.

**Texto

Descripción generada automáticamente**

Aquí vemos que al ejecuturlo varias veces, obviamente cambia la dirección de memoria, pero los valores de los apuntadores son los mismo.s.

Ejemplo del ptr + 9

****

****

**Caso 3:**

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

En este código podemos ver que estamos creando una clase, en los atributos públicos tenemos al constructor y al destructor. Estos constructores y destructores tienen un cout que indican si se está llamando al constructor (sale Persona Creada), y al destructor (Persona destruida).

En el main tenemos un apuntador a un objeto de la clase Persona, podemos ver esto gracias al \*, esto nos dice que es un apuntador y no un objeto. Este está apuntando a la dirección de memoria del objeto Persona. En la línea de abajo estamos inicializando el apuntador y si falla esta asignación, se manda un mensaje indicnado que no hay memoria. Si la memoria fue asignada correctamente, se manda a llamar al constructor que imprime su respectivo mensaje. Luego tenemos el cout de “antes de liberar memoria” y con el delete ptr; ptr = nullptr; estamos liberando la memoria (delete) y es aquí donde el destructor de se manda a llamar.

**Caso 4:**

**Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente**

Lo ejecuté 3 veces y salió lo mismo, “Persona creada “y “Persona destruida”. Tenemos clase Persona, con su constructor y destructor. Tenemos diferentes variables, por así decirlo, una global y una local. crearPersona de la función es SOLO de la función, y la del main es por aparte. Se inicializa el apuntador con nullptr, se le asigna memoria, y si no fue asignada correctamente, lanza el mensaje de que no hay espacio. En el main el ptr = crearPersona(); manda llamar a la función, aquí se asigna memoria al objeto y le da un apuntador a él. Si el apuntador es nulo, la memoria se libera (delete).

**Caso 5:**

**Texto

Descripción generada automáticamente Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente**

Primero debemos ver si un objeto está en el stack o el heap. Tenemos lo mismo de los ejemplos anteriores, la clase Persona, constructor y destructor. Después tenemos la función de asignación de memoria, solo que tenemos un ptr->nombre = “Juan”; Cuando cambiamos ptr->nombre = “Juan”; a ptr.nombre = “Juan”; nos manda error.

Texto

Descripción generada automáticamente

Cuando es un apuntador a un objeto se utiliza -> para acceder a los miembros del objeto apuntado. El punto es para el objeto.

**Caso 6:**

**Pantalla de computadora con letras

Descripción generada automáticamente con confianza media Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente**

Para mandar a llamar un método a través de un apuntador se usa el ->

Con este operador podemos acceder a los miembros de los datos y métodos. El ptr es un apuntador de tipo Persona para el objeto Persona y así podemos llamar a los setter y getters.

**Caso 6b:**

**Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente**

Este código tiene más objetos, es un arreglo de objetos con apuntador. Aquí asignamos memoria al arreglo. Cada vez que se crea una persona, sale el constructor, y cada vez que sale el destructor sale el nombre (objeto), El MAX SIZE es 4, por lo que el arreglo de objetos tiene 4 objetos.

**Caso 7:**

**Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente**

**Imagen que contiene botella, foto, firmar, teléfono

Descripción generada automáticamente**

No se lee el mensaje, ni el nombre. Aquí no sé por qué para eso, no sé por qué no se lee el último mensaje.

**Caso 8:**

**Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente**

Al correr el código, no hubo problema, a comparación del ejemplo anterior, según yo en el otro como que muestra el constructor y destructor, el delete va antes como de la función, vaya. Porque no se muestra la persona o algo. Pero en este lo único que pasa como “raro” es que el constructor se manda a llamar 2 veces, pero justo es porque hay 2 delete.

Caso 9:

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

En este código tenemos los deletes en condicionales. Estamos asegurándonos que si ya liberamos la memoria, no la volvamos a liberar.

Caso 10:

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

No se llamaron los 2 destructores, según yo solo se muestra LA SEGUNDA asignación de Pedro y no la de Juan. Como p se asigna a Juan y esa misma se reasigna a Pedro, es por eso es que solo se muestra Pedro al final. Sí salen los 2 constructores, pero solo 1 destructor.

**Caso 11:**

Texto

Descripción generada automáticamente

Aquí sale el mensaje de persona creada y destruida, pero después se me cuatropeó todo. Aquí todavía no entiendo por qué, justo pasó lo mismo la vez pasada pero no entiendo bien qué pasa.