Parte I – Análisis de Memoria

Dado el siguiente Código:

```
#include <iostream>
using namespace std;

int main() {

   int x = 8;
   int* p = &x;
   int* q = p;

   *q = 20;

   cout << x << endl;
   cout << *p << endl;
   cout << *q << endl;

   return 0;
}
```

Responda:

1. **¿Cuál es la salida del programa?**

   20
   20
   20

2. **¿Cuántos espacios de memoria distintos existen en el stack?**

   No hay espacios en memoria porque f

3. **¿Existe aliasing? Explique.**

   Si porque p y q direcciona a la memoria de x y hay 2 punteros accediendo al mismo espacio de la memoria

4. Dibuje el esquema de memoria (stack y heap).

- Stack:
    - x = 20
    - p → dirección de x
    - q → dirección de x
- Heap:
    - Vacío

Parte II – Memoria Dinámica (0.5 puntos)

Escriba un programa que:

1. Reserve memoria dinámica para un entero.
2. Asigne el valor 50 utilizando des referenciación.
3. Imprima:
    - Dirección almacenada en el puntero.
    - Valor almacenado en esa dirección.
    - Dirección del puntero.
4. Libere correctamente la memoria.
5. Asigne nullptr después del delete.

```
#include <iostream>

using namespace std;


int main() {

   int* p = new int;
   *p = 50;


   cout << "Direccion almacenada en p: " << p << endl;
   cout << "Valor almacenado en esa direccion: " << *p << endl;
   cout << "Direccion del puntero p: " << &p << endl;


   delete p;         // liberación correcta
   p = nullptr;       // evitar dangling pointer
   return 0;

}
```

Parte III – Análisis de Error

Analice el siguiente código:

```
int* p = new int(15);
int* q = p
delete p;
cout << *q << endl;
```

Explique:

1. **¿Qué tipo de error conceptual existe?**

Existe un dangling pointer**,** después de ejecutar delete p, la memoria queda liberada, pero q sigue apuntando a esa dirección.

2. ¿Por qué puede funcionar aparentemente?

Porque la memoria liberada **no se borra físicamente de inmediato**.
Si el sistema aún no reutiliza ese bloque de memoria, el valor puede seguir siendo 15 temporalmente.

Sin embargo, el comportamiento es **indefinido**.

3. ¿Qué concepto del estándar de C++ se está violando?

Se viola la regla de **acceso a memoria válida**.

Según el estándar de C++, acceder a memoria después de haber sido liberada produce un comportamiento indefinido

El programa ya no tiene garantía de resultado correcto.

Parte IV – Memory Leak (0.5 puntos)

Dado el código:

```
int* p = new int(10);
p = new int(30);
```

Responda:

1. **¿Existe memory leak?**

Si existe memoria leak

2. **¿Por qué ocurre?**

Se reserva memoria con `new int(10)`.

Luego el puntero `p` se reasigna con `new int(30)`.

Se pierde la referencia al primer bloque de memoria.

Esa memoria nunca se libera.

3. **Reescriba el código para evitar el leak**

```
int* p = new int(10);

delete p; // liberar antes de reasignar

p = new int(30);

delete p;
p = nullptr;
```