Instrucciones:
Programar cada uno de los segmentos de código mostrados en cada ejercicio; para cada uno de ellos hacer una ejecución y realizar una captura de pantalla del resultado además de explicar el resultado obtenido.

1. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a;
int b = 45;
cout <<"a: " <<a << endl <<"b: " <<b <<endl;
```

a: Sector de Memoria en a b: 45

Variable a: La variable a es un puntero a un entero (int*), pero no se le ha asignado una dirección de memoria válida antes de intentar acceder a su valor. Al imprimir a, se mostrará una dirección de memoria aleatoria que puede variar en cada ejecución del programa.
Variable b: La variable b se inicializa con el valor 45. Al imprimir b, se mostrará el valor 45. En este caso, b se comportará de manera predecible y mostrará el valor con el que fue inicializado.

2. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 45;
cout <<"a: " <<a << endl <<"b: " <<b <<endl;
```

a: Sector de Memoria en a b: 45

Variable a: Se asigna memoria dinámica para un entero con el valor 45 y se guarda la dirección de memoria en a. Al imprimir a, se mostrará la dirección de memoria donde se encuentra almacenado el valor 45.
Variable b: La variable b se inicializa con el valor 45. Al imprimir b, se mostrará el valor 45. En este caso, b se comportará de manera predecible y mostrará el valor con el que fue inicializado.

3. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 45;
cout <<"a: " <<a << endl <<"&b: " <<&b <<endl;
```

a: Sector de Memoria en a &b: Sector de Memoria en b

Variable a: Se asigna memoria dinámica para un entero con el valor 45 y se guarda la dirección de memoria en a. Al imprimir a, se mostrará la dirección de memoria donde se encuentra almacenado el valor 45.
Dirección de memoria de la variable b: Al imprimir &b, se mostrará la dirección de memoria de la variable b. La dirección de memoria de b es la ubicación en la memoria donde se almacena el valor de b.

4. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 45;
cout <<"*a: " <<*a << endl <<"&b: " <<&b <<endl;
```

*a: 45 &b: Sector de Memoria en b

Contenido de la variable apuntada por a: Al imprimir *a, se mostrará el valor almacenado en la dirección de memoria apuntada por a. En este caso, al imprimir *a, se mostrará el valor 45, ya que a apunta a un entero con el valor 45 que se asignó dinámicamente.
Dirección de memoria de la variable b: Al imprimir &b, se mostrará la dirección de memoria de la variable b. La dirección de memoria de b es la ubicación en la memoria donde se almacena el valor de b.

5. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 45;
cout <<"&*a: " <<&*a << endl <<"*&b: " <<*&b <<endl;
```

&*a: Sector de Memoria en a *&b: 45

Expresión &*a: *a accede al valor almacenado en la dirección de memoria apuntada por a, que en este caso es 45. &*a obtiene la dirección de memoria de *a, es decir, la dirección de memoria donde se encuentra almacenado el valor 45 al que apunta a.
Expresión *&b: &b obtiene la dirección de memoria de la variable b, es decir, la ubicación en la memoria donde se almacena el valor 45. *&b accede al valor almacenado en la dirección de memoria de b, que es el valor 45.

6. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 33;
a = &b;
cout <<"*a: " <<*a << endl <<"b: " <<b <<endl;
```

*a: 33 b: 33

Asignación de la dirección de memoria de b a a: Después de la asignación a = &b;, el puntero a ahora apunta a la dirección de memoria de la variable b. Esto significa que a ya no apunta a la memoria dinámica asignada previamente con el valor 45, sino que ahora apunta a la dirección de memoria de b.
Contenido de la variable apuntada por a: Al imprimir *a, se accede al valor almacenado en la dirección de memoria apuntada por a, que en este caso es la dirección de memoria de b.
Valor de la variable b: Al imprimir b, se mostrará el valor de la variable b, que es 33. Dado que a ahora apunta a la dirección de memoria de b, cualquier cambio en b también se reflejará al acceder a través de a.

7. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 33;
a = &b;
b = 55;
cout <<"*a: " <<*a << endl <<"b: " <<b <<endl;
```

*a: 55 b: 55

Asignación de la dirección de memoria de b a a: Después de la asignación a = &b;, el puntero a ahora apunta a la dirección de memoria de la variable b. Esto significa que a ya no apunta a la memoria dinámica asignada previamente con el valor 45, sino que ahora apunta a la dirección de memoria de b.
Cambio del valor de b a 55: Al realizar b = 55;, se cambia el valor de la variable b de 33 a 55.
Contenido de la variable apuntada por a: Al imprimir *a, se accede al valor almacenado en la dirección de memoria apuntada por a, que en este caso es la dirección de memoria de b. Dado que a apunta a la dirección de memoria de b, al imprimir *a, se mostrará el valor de b, que es 55 después de haber sido modificado.
Valor de la variable b:Al imprimir b, se mostrará el valor actual de la variable b, que es 55.

8. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 33;
a = &b;
b = 55;
*a = 27;
cout <<"*a: " <<*a << endl <<"b: " <<b <<endl;
```

XD

*a: 27 b: 27

9. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
int* a = new int(45);
int b = 33;
a = &b;
```

int* a = new int(45); Se crea un puntero a que apunta a un nuevo entero en el heap con el valor 45.
int b = 33; Se crea una variable entera b con el valor 33 en la pila.
a = &b; El puntero a ahora apunta a la dirección de memoria de b, por lo que a apunta a la variable b.
b = 55; Se cambia el valor de b a 55.
a = new int(27); Se asigna a a un nuevo entero en el heap con el valor 27. El entero al que apuntaba anteriormente en el heap (45) queda sin referencia y se convierte en basura.
cout <<"*a: " <<*a << endl <<"b: " <<b <<endl; Se imprime el valor al que apunta a y el valor de b.
Como a ahora apunta a un entero en el heap con valor 27, *a imprime 27.
El valor de b sigue siendo 55, ya que se modificó después de que a dejara de apuntar a b

```
    b = 55;
    a = new int(27);
    cout <<"*a: " <<*a << endl <<"b: " <<b <<endl;
```

*a: 27 b: 55

10. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
    int* a = new int(45);
    int b = 33;
    cout <<"a: " <<a << endl <<"a+1: " <<a+1 <<endl <<"b: " <<b <<endl;
```

a: Siguiente Sector de Memoria en a

int* a = new int(45); Se crea un puntero a que apunta a un nuevo entero en el heap con el valor 45.
int b = 33; Se crea una variable entera b con el valor 33 en la pila.
cout <<"a: " <<a << endl <<"a+1: " <<a+1 <<endl <<"b: " <<b <<endl;
Se imprime la dirección de memoria a la que apunta a, la dirección de memoria a la que apunta a+1 y el valor de b.
a contiene la dirección de memoria del entero en el heap (45 en este caso).
a+1 apunta a la siguiente posición de memoria después de a, que en este caso sería la siguiente posición de memoria después del entero en el heap. El valor de b es 33

10.1 ¿Qué resultará de evaluar a+2?

11. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
    int* a = new int(45);
    int b = 33;
    cout <<"*a: " <<*a << endl <<"*a+1: " <<*a+1 <<endl <<"b: " <<b <<endl;
```

*a: 45 *a+1: 46 b: 33

int* a = new int(45); Se crea un puntero a que apunta a un nuevo entero en el heap con el valor 45.
int b = 33; Se crea una variable entera b con el valor 33 en la pila.
cout <<"*a: " <<*a << endl <<"*a+1: " <<*a+1 <<endl <<"b: " <<b <<endl;
Se imprime el valor al que apunta a, el valor al que apunta a incrementado en 1, y el valor de b.
*a accede al valor al que apunta a, que es 45 en este caso.
*a+1 suma 1 al valor al que apunta a (45), por lo que se imprime 46.
El valor de b es 33.

12. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
void swap(int &a, int &b)
{
    int temporal;
    temporal = a;
    a = b;
    b = temporal;
}

int main()
{
    int c = 5;
    int d = 7;
    cout <<"Antes" <<endl <<"c: " <<c <<endl <<"d: " <<d <<endl;
    swap(c, d);
    cout <<"Después" <<endl <<"c: " <<c <<endl <<"d: " <<d <<endl;

    return 0;

}
```

Se definen las variables c y d con los valores iniciales 5 y 7 respectivamente.
Se imprime el valor de c y d antes de llamar a la función swap.
La salida será:
Antes
c: 5
d: 7
Se llama a la función swap(c, d) para intercambiar los valores de c y d.
Dentro de la función swap, se intercambian los valores de a y b utilizando una variable temporal.
Se imprime el valor de c y d después de llamar a la función swap.
La salida será:
c: 7
d: 5

antes. c: 5 d: 7 después. c: 7 d: 5

13. ¿Qué resulta de ejecutar el siguiente segmento de código?

```
void incrementa(int *a)
{
    (*a)++;
}

int main()
{
    int* numero = new int(33);

    cout <<"Antes" <<endl <<"*numero: " <<*numero <<endl;
    incrementa(numero);
    cout <<"Después" <<endl <<"*numero: " <<*numero <<endl;
    return 0;
}
```

Se crea un puntero numero que apunta a un entero en el heap con el valor 33.
Se imprime el valor al que apunta numero antes de llamar a la función incrementa.
La salida será:
*numero: 33
Se llama a la función incrementa(numero) para incrementar en 1 el valor al que apunta numero.
Dentro de la función incrementa, se incrementa en 1 el valor al que apunta el puntero a.
Se imprime el valor al que apunta numero después de llamar a la función incrementa.
La salida será:
*numero: 34

Antes. *numero: 33 Después. *numero: 34