Conceptuales

1. ¿Qué es un apuntador en C++?
   a) Una variable que almacena datos
   b) Una variable que almacena direcciones de memoria
   c) Un tipo especial de estructura
   d) Una función especial
2. ¿Qué operador se usa para obtener la dirección de memoria de una variable?
   a) *
   b) ->
   c) &
   d) %
3. ¿Qué operador se usa para acceder al valor almacenado en la dirección de un apuntador?
   a) &
   b) *
   c) ->
   d) .
4. ¿Cuál es el valor inicial recomendado de un apuntador si no apunta a nada?
   a) 1
   b) 0
   c) NULL
   d) undefined
5. ¿Qué diferencia hay entre *p y p en un apuntador?
   a) Ninguna
   b) p es el valor apuntado, *p es la dirección
   c) p es la dirección, *p es el valor
   d) Son equivalentes siempre
6. ¿Qué pasa si intentamos acceder a través de un apuntador no inicializado?
   a) El programa corre normal
   b) Se produce comportamiento indefinido
   c) Siempre lanza error de compilación
   d) Devuelve 0
7. ¿Cuál es la forma correcta de declarar un apuntador a entero?
   a) int p;
   b) int* p;

c) *int p;
d) pointer int p;

8. ¿Qué es nullptr en C++ moderno?
   a) Una constante entera
   b) Un literal especial para apuntadores nulos
   c) Un tipo de memoria dinámica
   d) Un error de compilación
9. ¿Qué significa el operador -> en apuntadores a estructuras o clases?
   a) Es igual que . pero para apuntadores
   b) Sirve para multiplicar
   c) Devuelve la dirección
   d) Sirve para borrar memoria
10. ¿Qué ocurre al usar memoria dinámica sin liberar con delete o delete[]?
    a) Nada
    b) Error de compilación
    c) Fuga de memoria
    d) Se reinicia el programa

Memoria dinámica

11. ¿Qué instrucción reserva memoria dinámica para un entero?
    a) int* p = malloc(4);
    b) int* p = new int;
    c) int p = new int;
    d) malloc(int);
12. ¿Qué instrucción libera memoria de un solo entero reservado con new?
    a) delete p;
    b) free(p);
    c) delete[] p;
    d) remove(p);
13. ¿Qué instrucción se usa para liberar memoria de un arreglo dinámico?
    a) delete p;
    b) delete[] p;
    c) free(p);
    d) remove[] p;

14. ¿Qué sucede si se hace delete dos veces al mismo apuntador?
    a) Nada
    b) Comportamiento indefinido
    c) Se libera dos veces
    d) Error de compilación
15. ¿Qué instrucción crea un arreglo dinámico de 10 enteros?
    a) int p[10];
    b) int* p = new int[10];
    c) int* p = malloc(10);
    d) int* p = new[10];
16. ¿Cómo se debe liberar un arreglo creado con new int[10];?
    a) delete p;
    b) delete[] p;
    c) free(p);
    d) clear(p);
17. ¿Qué instrucción reserva memoria dinámica para un número decimal (double)?
    a) double* p = new double;
    b) double p = new double;
    c) new double p;
    d) malloc(double);
18. ¿Cuál es el riesgo de no usar delete en un programa que usa memoria dinámica constantemente?
    a) El programa será más rápido
    b) Se producirán fugas de memoria
    c) No compila
    d) La memoria se autolibera
19. ¿Qué función de C se usaba para reservar memoria dinámica antes de new?
    a) malloc
    b) calloc
    c) alloc
    d) create
20. ¿Qué función de C libera memoria reservada con malloc?
    a) free
    b) delete
    c) release
    d) exit

Ejemplos prácticos de código

21. Si int x = 5; int* p = &x;, ¿qué imprime *p?
    a) Dirección de x
    b) 5
    c) Nada
    d) Error
22. ¿Qué hace el siguiente código?

```
int* p = new int;
*p = 10;
delete p;
```

a) Asigna 10 y libera memoria
b) Genera error de compilación
c) Crea un arreglo
d) Nada

23. Si int* p = new int[5];, ¿cuántos enteros se reservan en memoria?
    a) 1
    b) 4
    c) 5
    d) 10
24. ¿Qué hace el siguiente código?

```
int a = 7;
int* p = &a;
cout << p;
```

a) Imprime el valor de a
b) Imprime la dirección de a
c) Error de compilación
d) Nada

25. En int* p = nullptr;, ¿qué valor tiene p?
    a) 0
    b) No apunta a ninguna dirección válida

c) Una dirección aleatoria
d) 1

26. ¿Qué ocurre si no usamos delete[] con un arreglo dinámico?
    a) El programa se reinicia
    b) Se produce una fuga de memoria
    c) Error de compilación
    d) Nada
27. ¿Qué significa que los apuntadores permiten **aritmética de punteros**?
    a) Se pueden sumar y restar direcciones
    b) Se pueden convertir en enteros
    c) Se pueden dividir
    d) No se puede hacer nada
28. ¿Qué ocurre si haces int* p; *p = 5; sin inicializar p?
    a) Guarda 5 en p
    b) Error de compilación
    c) Comportamiento indefinido
    d) Imprime 5
29. ¿Qué instrucción reserva memoria para una estructura llamada Persona?
    a) Persona p;
    b) Persona* p = new Persona;
    c) Persona p = new Persona;
    d) malloc(Persona);
30. ¿Cuál es la forma correcta de liberar memoria de una estructura creada con new?
    a) delete p;
    b) delete[] p;
    c) free(p);
    d) remove(p);