Pointer Revision Basic To Advance

Question 1:

Which of the following is true about pointers in C++?

A) A pointer can store the address of any variable, regardless of type.

B) A pointer must be initialized at the time of declaration.

C) A pointer stores the memory address of another variable.

D) A pointer can only store the address of integer variables.

▼ Solution

```
Answer: C) A pointer stores the memory address of another variable.
Explanation: A pointer stores the memory address of another variable, but
```

Question 2:

What is the size of a pointer variable in a 64-bit system?

A) 4 bytes

B) 8 bytes

C) Depends on the data type

D) None of these

▼ Solution

> Answer: B) 8 bytes
> Explanation: On a 64-bit system, pointers are 8 bytes because they store 6

Question 3:

What happens if you dereference an uninitialized pointer?

A) Undefined behavior

B) Compiler error

C) Null value is returned

D) Segmentation fault

▼ Solution

> Answer: A) Undefined behavior
> Explanation: Dereferencing an uninitialized pointer leads to undefined beh

---

Question 4:

Which operator is used to access the value stored at a memory address pointed by a pointer?

A) &

B) *

C) →

D) .

▼ Solution

> Answer: B)
> Explanation: The `*` operator is used to dereference a pointer and access

Question 5:

What will be printed by this code?

```
int x = 10;
int *ptr = &x;
cout << *ptr;
```

A) Memory address of x

B) Value of x (10)

C) Compilation error

D) Garbage value

▼ Solution

```
Answer: B) Value of x (10)
Explanation: `*ptr` dereferences the pointer and gives the value of `x`, whi
```

Question 6:

How do you declare a pointer to an integer in C++?

A) int ptr;

B) int *_ptr;_
C) int &ptr;
D) int ptr
;

▼ Solution

```
Answer: B) int *ptr;
Explanation: The correct syntax is `int *ptr;`, which declares a pointer to a
```

Question 7:

What will happen if you assign an integer value directly to a pointer?

```
int x = 10;
int *ptr = x;
```

A) Pointer will store the value of x.

B) Compilation error.

C) Pointer will point to garbage memory.

D) Pointer will store a null value.

▼ Solution

Answer: B) Compilation error.
Explanation: You must use `&x` to assign the address of `x` to the pointer.

Question 8:

What will be printed by this code?

```
int x = 5;
int *p = &x;
*p += 5;
cout << x;
```

A) 5

B) 10

C) Address of x

D) Compilation error

▼ Solution

Answer: B) 10
Explanation: `*p += 5` modifies the value of `x` to 10.

Question 9:

Which statement correctly declares and initializes two pointers to integers?

A) int *p1, p2;

B) int *p1, *p2;

C) int p1, p2;

D) int *p1 = nullptr, *p2 = nullptr;

▼ Solution

```
Answer: D) int *p1 = nullptr, p2 = nullptr;
Explanation: This properly declares two pointers and initializes them to `nu
```

Question 10:

What does this code output?

```
int x = 5, y = 10;
int *p = &x;
p = &y;
cout << *p;
```

A) 5

B) 10

C) Address of x

D) Address of y

▼ Solution

```
Answer: B) 10
Explanation: `p` is reassigned to point to `y`, so `*p` prints `10`.
```

Question 11:

What will be the output of the following code?

```
int arr[] = {10, 20, 30};
int *p = arr;
```

```
cout << *(p + 1);
```

A) 10

B) 20

C) 30

D) Compilation error

▼ Solution

Answer: B) 20
Explanation: `p + 1` points to the second element in the array, which is `20

Question 12:

Which of the following is true for pointer arithmetic?

A) Pointers can be added to integers.

B) Pointers can be divided.

C) Pointers can be multiplied.

D) You can subtract two pointers of different types.

▼ Solution

Answer: A) Pointers can be added to integers.
Explanation: Pointer arithmetic allows adding/subtracting integers to/from

Question 13:

What will this code print?

```
int x = 7;
int *p = &x;
int **q = &p;
cout << **q;
```

A) Address of x

B) Address of p

C) 7

D) Compilation error

▼ Solution

> Answer: C) 7
> Explanation: `**q` accesses the value of `x` through a pointer to a pointer.

Question 14:

Which of the following is used to declare a pointer to a pointer to an integer?

A) int **ptr;

B) int *ptr;

C) int ptr;

D) int &ptr;

▼ Solution

> Answer: A) int **ptr;
> Explanation: `int **ptr` is used to declare a pointer to a pointer to an intege

Question 15:

What does `nullptr` represent in modern C++?

A) A random memory location

B) An uninitialized pointer

C) A null pointer constant

D) An integer zero

▼ Solution

> Answer: C) A null pointer constant
> Explanation: `nullptr` was introduced in C++11 to represent a null pointer c

Question 16:

What is the result of this code?

```
int a = 5;
int *p = &a;
*p = *p + 1;
cout << a;
```

A) 5

B) 6

C) Garbage

D) Compilation error

▼ Solution

Answer: B) 6
Explanation: `*p = *p + 1;` increments `a` by 1, resulting in `6`.

Question 17:

Which operation is invalid for pointers?

A) Addition of pointer and integer

B) Subtraction of two pointers of same type

C) Multiplication of two pointers

D) Comparison of two pointers

▼ Solution

Answer: C) Multiplication of two pointers
Explanation: Multiplying two pointers is not allowed in C++.

Question 18:

What is a dangling pointer?

A) A pointer pointing to a valid memory location

B) A pointer not initialized yet

C) A pointer pointing to memory that has been freed

D) A pointer pointing to another pointer

▼ Solution

```
Answer: C) A pointer pointing to memory that has been freed
Explanation: Dangling pointers point to memory that is no longer valid, lea

int *ptr = new int(10);  // Allocate memory
delete ptr;              // Memory is freed
cout << *ptr;            // Dangling pointer! (Undefined Behavior)
```

Question 19:

What is the use of `&` in pointer declaration?

A) It defines the size of a pointer

B) It is used to dereference the pointer

C) It is used to access the address of a variable

D) It multiplies the pointer

▼ Solution

```
Answer: C) It is used to access the address of a variable
Explanation: `&` is used to get the memory address of a variable.
```

Question 20:

Which statement best describes the relationship between arrays and pointers?

A) Arrays and pointers are completely unrelated.

B) Arrays store pointers internally.

C) The name of the array is a constant pointer to its first element.

D) Pointers are arrays in disguise.

▼ Solution

Answer: C) The name of the array is a constant pointer to its first element.
Explanation: In most contexts, an array name is treated as a pointer to its f

Question 21:

What will this code print?

```
int a = 100;
int *p = &a;
int **q = &p;
int ***r = &q;
cout << ***r;
```

A) 100

B) Address of a

C) Address of p

D) Garbage value

▼ Solution

Answer: A) 100
Explanation: `***r` dereferences three levels of pointers to get the value of

Question 22:

Which of the following is **not** a valid pointer type in C++?

A) `int *`

B) `float **`

C) `void *`

D) `string &*`

▼ Solution

Answer: D) `string &*`
Explanation: `&*` is not a valid combination for pointer declaration.

Question 23:

What will happen if you try to dereference a null pointer?

A) It will return 0

B) It will compile successfully and print garbage

C) It will cause a segmentation fault/runtime error

D) It will print NULL

▼ Solution

Answer: C) It will cause a segmentation fault/runtime error
Explanation: Dereferencing `nullptr` leads to a runtime crash.

Question 24:

Which keyword is used to dynamically allocate memory in C++?

A) malloc

B) alloc

C) new

D) create

▼ Solution

Answer: C) new
Explanation: `new` is the C++ keyword for dynamic memory allocation.

Question 25:

What will this code print?

```
int *p = new int(10);
cout << *p;
delete p;
```

A) 0

B) 10

C) Garbage

D) Error

▼ Solution

> Answer: B) 10
> Explanation: `new int(10)` creates an integer with value `10`. It's deleted aft

Question 26:

Which operator is used to deallocate memory allocated using `new`?

A) delete

B) free

C) remove

D) dispose

▼ Solution

> Answer: A) delete
> Explanation: `delete` is used in C++ to free memory allocated with `new`.

Question 27:

Which of the following is **true** about `void *` in C++?

A) It cannot be assigned any other pointer.

B) It can store the address of any data type.

C) It must be dereferenced directly.

D) It is not allowed in C++.

▼ Solution

> Answer: B) It can store the address of any data type.
> Explanation: A `void *` is a generic pointer that can store any type's addre

Question 28:

If `int a = 50; int *p = &a;`, what does `p + 1` represent?

A) The next integer value

B) The address of the next integer variable (not value)

C) a + 1

D) Compilation error

▼ Solution

> Answer: B) The address of the next integer variable (not value)
> Explanation: `p + 1` points to the next integer-sized memory location.

Question 29:

Which of the following is used to pass an array to a function using pointers?

A) Pass the base address

B) Pass the address of the array name

C) Pass the first element using reference

D) All of the above

▼ Solution

> Answer: D) All of the above
> Explanation: All these methods can be used depending on syntax.

Question 30:

What will be the output?

```
int a = 5, b = 10;
int *p1 = &a, *p2 = &b;
*p1 = *p2;
cout << a;
```

A) 5

B) 10

C) Garbage

D) Address of b

▼ Solution

```
Answer: B) 10
Explanation: `*p1 = *p2` assigns the value of `b` (10) to `a`. So `a` becomes
```

Advance Pointer Questions:

Question 1:

What happens when you increment a pointer pointing to an array?

(A) It points to the next byte in memory.

(B) It points to the next element of the array based on the data type size.

(C) It points to the previous element of the array.

(D) It remains unchanged.

▼ Solution

```
Answer: (B) It points to the next element of the array based on the data typ
Explanation: Pointer arithmetic is type-aware - incrementing moves by siz
```

Question 2:

What happens when you access an array using a pointer?

(A) The pointer must be incremented manually to access each element.

(B) The pointer automatically points to each element in sequence.

(C) The array is converted into a pointer.

(D) The pointer becomes an array.

▼ Solution

> Answer: (A) The pointer must be incremented manually to access each element.
> Explanation: Arrays don't auto-traverse - you must explicitly move the pointer.

Question:

What will be the output of this code?

```
int arr[] = {1, 2, 3};
int *ptr = arr + 2;
cout << *ptr;
```

(A) 1

(B) 2

(C) 3

(D) Undefined Behavior

▼ Solution

> Answer: (C) 3
> Explanation: arr + 2 calculates address offset for 2 elements (2*sizeof(int)).

Question 3:

What will be the output of this code?

```
int arr[] = {1, 2, 3};
int *ptr = arr;
cout << *(ptr + 1) << " " << *(ptr + 2);
```

(A) 1 2

(B) 1 3

(C) 2 3

(D) Error

▼ Solution

Answer: (C) 2 3
Explanation: Pointer arithmetic respects array bounds in this case.

Question 4:

What is the purpose of the new operator in C++?

(A) To allocate memory dynamically.

(B) To deallocate memory dynamically.

(C) To declare variables.

(D) To initialize variables.

▼ Solution

Answer: (A) To allocate memory dynamically.
Explanation: new allocates heap memory, delete frees it.

Question 5:

What does this code do?

```
int *ptr = new int;
*ptr = 10;
```

(A) Allocates memory for an integer and assigns it the value 10.

(B) Allocates memory for an integer but does not assign a value.

(C) Compilation error.

(D) Garbage value.

▼ Solution

Answer: (A) Allocates memory for an integer and assigns it the value 10.
Explanation: Demonstrates basic dynamic memory allocation.

Question 6:

What will happen if you forget to use delete after new?

(A) Memory leak.

(B) Compilation error.

(C) Runtime error.

(D) None of these.

▼ Solution

Answer: (A) Memory leak.
Explanation: Unfreed heap allocations cause memory leaks.

Question 7:

What does this statement mean?

```
int *ptr = new int[10];
```

(A) Allocates memory for a single integer.

(B) Allocates memory for an array of 10 integers.

(C) Allocates memory for a pointer to an integer.

(D) None of these.

▼ Solution

Answer: (B) Allocates memory for an array of 10 integers.
Explanation: new[] allocates contiguous memory for arrays.

Question 8:

What will be the output of this code?

```
int *ptr = new int[5];
for (int i = 0; i < 5; i++) {
    ptr[i] = i * 10;
}
for (int i = 0; i < 5; i++) {
    std::cout << ptr[i] << " ";
}
delete[] ptr;
```

(A) 0 10 20 30 40

(B) 10 20 30 40 50

(C) Compilation error.

(D) Garbage value.

▼ Solution

Answer: (A) 0 10 20 30 40
Explanation: Shows proper array allocation/initialization/deletion.

Smart Pointers

Notes: Smart Pointers

Question 10:

What is the difference between unique_ptr and shared_ptr?

(A) unique_ptr allows shared ownership, while shared_ptr does not.

(B) unique_ptr does not allow shared ownership, while shared_ptr does.

(C) unique_ptr is faster than shared_ptr.

(D) shared_ptr is faster than unique_ptr.

▼ Solution

Answer: (B) unique_ptr does not allow shared ownership, while shared_ptr does.

Explanation: Key difference in ownership semantics.

Question 11:

What does this code do?

```
#include <memory>
int main() {
unique_ptr<int> ptr(new int(10));
cout << *ptr;
}
```

(A) Prints 10.

(B) Prints the memory address of ptr.

(C) Compilation error.

(D) Garbage value.

▼ Solution

Answer: (A) Prints 10.
Explanation: Demonstrates basic unique_ptr usage.

Question 12:

What will happen if you try to copy a unique_ptr?

(A) Compilation error.

(B) Runtime error.

(C) The copy operation succeeds.

(D) Memory leak.

▼ Solution

Answer: (A) Compilation error.
Explanation: unique_ptr cannot be copied (only moved).

Question 13:

What does this statement mean?

```
shared_ptr<int> ptr(new int(10));
```

(A) Creates a shared pointer to an integer with the value 10.

(B) Creates a unique pointer to an integer with the value 10.

(C) Creates a raw pointer to an integer with the value 10.

(D) None of these.

▼ Solution

Answer: (A) Creates a shared pointer to an integer with the value 10.
Explanation: Shows shared_ptr initialization.

Question 14:

What will be the output of this code?

```
#include <memory>
int main() {
        shared_ptr<int> ptr1(new int(10));
        shared_ptr<int> ptr2 = ptr1;
        cout << *ptr1 << " " << *ptr2;
    return 0;
}
```

(A) 10 10

(B) 10 Garbage value

(C) Compilation error.

(D) Runtime error.

▼ Solution

Answer: (A) 10 10
Explanation: Demonstrates shared_ptr reference counting.

Pointer Arithmetic and Array Modification

Question 15:

What is the output of this code?

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

int main() {
    int arr[] = {10, 20, 30,40};
    int *ptr = arr;
    cout << *ptr;
    ptr += 2;
    cout << *ptr;
    arr++;
    cout << *arr;
    cout << "bye";
    return 0;
}
```

(A) Compilation Error

(B) Runtime Error

(C) 10, 30, 20

(D) None of these

▼ Solution

Answer: (A) Compilation Error
Explanation: Array names are constant pointers (can't modify).

Question 16:

What is the output of this code?

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

int main() {
    int i;
    double d;
    char c;
    int *ip = &i;
    double *dp = &d;
    char *cp = &c;

    cout << sizeof(i) << " " << sizeof(d) << " " << sizeof(c) << endl;
    cout << sizeof(ip) << " " << sizeof(dp) << " " << sizeof(cp) << endl;
}
```

(A) 4 8 1 , 4 8 1

(B) 8 8 8 , 4 8 1

(C) 4 8 1 , 8 8 8

(D) None of the above

▼ Solution

> Answer: (C) 4 8 1 , 8 8 8
> Explanation: Shows type sizes vs pointer sizes (64-bit system).

Question 17:

What is the output of this pointer arithmetic code?

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

int main() {
    int *ip;   // Assume 1500 as base address
    double *dp; // Assume 2500 as base address

    cout << ip << " " << dp;
    cout << ip + 1 << " " << dp + 2;
    cout << ip - 1 << " " << dp - 3;
    return 0;
}
```

(A) 1500 2500 1508 2508 1492 2588

(B) 1500 2500 1508 2516 1492 2576

(C) 1500 2500 1504 2516 1496 2476

(D) None of these

▼ Solution

> Answer: (C) 1500 2500 1504 2516 1496 2576
> Explanation: Demonstrates type-aware pointer arithmetic.

Question 18:

What is the output?

```
int arr[] = {1, 2, 3};
int *p = arr;
cout << *(p + 5);
```

(A) 1

(B) 3

(C) Undefined Behavior

(D) Compilation Error

▼ Solution

> Answer: (C) Undefined Behavior
> Explanation: Accessing beyond array bounds is UB.

Question 19:

What happens after this code?

```
vector<int> vec = {1, 2, 3};
int *p = vec.data();
vec.push_back(4);
cout << *p;
```

(A) Prints 1

(B) Prints 4

(C) Undefined Behavior

(D) Compilation Error

▼ Solution

> Answer: (A) Prints 1

Question 20:

What does this code do?

```
int arr[5] = {1, 2, 3};
int *p = arr + 3;
*p = 10;
```

(A) Assigns 10 to arr[3]

(B) Assigns 10 to arr[0]

(C) Undefined Behavior

(D) Compilation Error

▼ Solution

> Answer: (A) Assigns 10 to arr[3]
> Explanation: Valid access to initialized array element.

Question:

What is the output?

```
int arr[] = {1, 2, 3};
int *p = &arr[0];
p++;
std::cout << p[-1];
```

(A) 1

(B) 2

(C) 3

(D) Undefined Behavior

▼ Solution

> Answer: (A) 1
> Explanation: p[-1] is equivalent to *(p - 1).

Question:

What does this code do?

```
vector<int> vec(5, 10);
int *p = &vec[2];
vec.insert(vec.begin(), 3);
cout << *p;
```

(A) Prints 10

(B) Prints 3

(C) Undefined Behavior

(D) Compilation Error

- Solution

> Answer: (C) Undefined Behavior
> Explanation: insert invalidates iterators/pointers.

Question:

What is the output?

```
int arr[] = {1, 2, 3};
int *p = arr;
std::cout << (*(p + 1) == arr[1]);
```

(A) 0

(B) 1

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (B) 1
  Explanation: Both expressions access the same element.
  ```

Question:

What is the output?

```
vector<int> vec = {1, 2, 3};
int *p = vec.data();
vec.reserve(100);
cout << (p == vec.data());
```

(A) 0

(B) 1

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (A) 0
  Explanation: reserve may reallocate, changing data() address.
  ```

Question:

What does this code do?

```
int arr[] = {1, 2, 3};
int *p = &arr[2];
int *q = &arr[0];
cout << p - q;
```

(A) 2

(B) 3

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (A) 2
  Explanation: Pointer subtraction gives element count difference.
  ```

Question:

What is the output?

```
vector<int> vec = {1, 2, 3};
int *p = &vec[0];
vec.erase(vec.begin());
std::cout << *p;
```

(A) 1

(B) 2

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (C) 2
  ```

Question:

What does this code do?

```
int arr[] = {1, 2, 3};
const int *p = arr;
p++;
cout << *p;
```

(A) Prints 1

(B) Prints 2

(C) Undefined Behavior

(D) Compilation Error

- Solution

  Answer: (B) Prints 2
  Explanation: const prevents modification but allows traversal.

Question:

What is the output?

```
std::vector<int> vec = {1, 2, 3};
int *p = vec.data() + 1;
vec.resize(10);
std::cout << *p;
```

(A) 2

(B) 0

(C) Undefined Behavior

(D) Compilation Error

- Solution

  Answer: (C) Undefined Behavior
  Explanation: resize may reallocate, invalidating pointers.

Question:

Which statement is true?

```
int arr[] = {1, 2, 3};
int *p = arr;
```

(A) sizeof(arr) == sizeof(p)

(B) sizeof(arr) > sizeof(p)

(C) sizeof(arr) < sizeof(p)

(D) Compilation Error

- Solution

  ```
  Answer: (B) sizeof(arr) > sizeof(p)
  Explanation: Array size vs pointer size comparison.
  ```

Question:

What is the output?

```
int arr[] = {1, 2, 3};
int *p = arr;
int *q = arr + 3;
std::cout << (q - p);
```

(A) 0

(B) 3

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (B) 3
  Explanation: Pointer subtraction gives element count.
  ```

Question:

What happens here?

```
std::vector<int> vec;
vec.reserve(3);
int *p = vec.data();
vec.push_back(1);
std::cout << *p;
```

(A) Prints 1

(B) Undefined Behavior

(C) Compilation Error

(D) Prints 0

- Solution

    Answer: (A) Prints 1
    Explanation: reserve() pre-allocates stable memory.

Question:

What does this code do?

```
int arr[] = {1, 2, 3};
int *p = arr;
std::cout << *(p + (-1));
```

(A) Prints 1

(B) Prints garbage

(C) Undefined Behavior

(D) Compilation Error

- Solution

    Answer: (C) Undefined Behavior
    Explanation: Negative pointer arithmetic is UB.

Question:

What is the output?

```
std::vector<int> vec = {1, 2, 3};
int *p = vec.data();
vec.shrink_to_fit();
std::cout << (p == vec.data());
```

(A) 0

(B) 1

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (A) 0
  Explanation: shrink_to_fit may reallocate memory.
  ```

Question:

What is the output?

```
int arr[] = {1, 2, 3};
auto *p = &arr;
std::cout << (*p)[2];
```

(A) 1

(B) 2

(C) 3

(D) Compilation Error

- Solution

  ```
  Answer: (C) 3
  Explanation: p is pointer-to-array, (*p)[2] accesses third element.
  ```

Question:

What does this code do?

```
std::vector<int> vec = {1, 2, 3};
const int *p = vec.data();
vec[1] = 10;
std::cout << p[1];
```

(A) Prints 10

(B) Prints 2

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (A) Prints 10
  Explanation: const pointer doesn't prevent source modification.
  ```

Question:

What is the output?

```
int arr[] = {1, 2, 3};
int *p = arr;
std::cout << (*(&p) == p);
```

(A) 1

(B) 0

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (A) 1
  Explanation: *(&p) dereferences pointer-to-pointer, yielding p.
  ```

Question:

What is the output?

```
std::vector<int> vec = {1, 2, 3};
int *p = vec.data() + 1;
vec.emplace(vec.begin(), 0);
std::cout << *p;
```

(A) 0

(B) 1

(C) Undefined Behavior

(D) Compilation Error

- Solution

  ```
  Answer: (C) Undefined Behavior
  Explanation: emplace at begin invalidates pointers.
  ```

Question:

What happens if you attempt to modify the base address of an array in C++?

(A) The base address can be changed using pointer arithmetic.

(B) The base address cannot be changed; it is constant.

(C) The array will be reallocated to a new memory location.

(D) Compilation error occurs.

- Solution

  ```
  Answer: (B) The base address cannot be changed; it is constant.
  Explanation: Array names are constant pointers.
  ```

Question:

Which statement is true about assigning values to pointers in C++?

(A) A pointer can store any type of value directly.

(B) A pointer must be compatible with the type it points to.

(C) A pointer can store values without type compatibility using void*.

(D) Pointers automatically convert incompatible types at runtime.

- Solution

  Answer: (B) A pointer must be compatible with the type it points to.
  Explanation: Strong typing requirement in C++.

Question:

What is the purpose of a void* pointer in C++?

(A) To store addresses without type information.

(B) To store only integer addresses.

(C) To store addresses with strict type checking.

(D) To store multiple values simultaneously.

- Solution

  Answer: (A) To store addresses without type information.
  Explanation: void* is a generic pointer type.

Question:

What happens if you access an array index out-of-bounds in C++?

```
int arr[3] = {1, 2, 3};
cout << arr[5];
```

(A) Compiler error occurs due to bound checking.

(B) Runtime error occurs due to bound checking failure.

(C) Undefined behavior occurs as C++ does not perform bound checking for arrays.

(D) Garbage value is printed safely due to automatic bound checking.

- Solution

  ```
  Answer: (C) Undefined behavior occurs as C++ does not perform bound checking for arrays.
  Explanation: C++ trusts programmers with memory access.
  ```

Question:

What will be the output of the following code?

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

void increment(int* num) {
    (*num)++;
}

int main() {
    int value = 5;
    increment(&value);
    cout << value;
    return 0;
}
```

(A) 4

(B) 5

(C) 6

(D) Compilation error

- Solution

  ```
  Answer: (C) 6
  Explanation: The function increments the value by dereferencing the pointer.
  ```

Question:

What will happen if you run this code?

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

int main() {
    int* ptr;
    cout << *ptr; // Dereferencing uninitialized pointer
    return 0;
}
```

(A) Prints garbage value

(B) Compilation error

(C) Runtime error

(D) Prints 0

- Solution

    ```
    Answer: (C) Runtime error
    Explanation: Dereferencing an uninitialized pointer leads to undefined behavior (often segmentation fault).
    ```

Question:

What is the output of this code?

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

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 10, y = 20;
```

```
    swap(&x, &y);
    cout << x << " " << y;
    return 0;
}
```

(A) 10 20

(B) 20 10

(C) Compilation error

(D) Runtime error

- Solution

  ```
  Answer: (B) 20 10
  Explanation: The swap function correctly swaps the values using pointers.
  ```

Question:

What will this code print?

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

void allocateMemory(int** ptr) {
    *ptr = new int(42);
}

int main() {
    int* p = nullptr;
    allocateMemory(&p);
    cout << *p;
    delete p;
    return 0;
}
```

(A) 0

(B) 42

(C) Compilation error

(D) Memory leak

- Solution

  Answer: (B) 42
  Explanation: The function allocates memory and assigns 42 to the dereferenced pointer.

Question:

What will be the output of this code?

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

int main() {
    int arr[] = {1, 2, 3};
    int* ptr = arr;
    cout << *(ptr + 2);
    return 0;
}
```

(A) 1

(B) 2

(C) 3

(D) Compilation error

- Solution

  Answer: (C) 3
  Explanation: Pointer arithmetic accesses the third element (index 2).

Question:

What will happen if you execute this code?

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

void changeValue(int* ptr) {
    ptr = new int(100);
}

int main() {
    int* p = new int(50);
    changeValue(p);
    cout << *p;
    delete p;
    return 0;
}
```

(A) 50

(B) 100

(C) Compilation error

(D) Memory leak

- Solution

  > Answer: (A) 50
  > Explanation: The pointer `p` is passed by value, so the original remains unchanged.

Question:

What does this code output?

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

int main() {
    int x = 5, y = 10;
    int* ptr1 = &x;
    int* ptr2 = &y;
    cout << (*ptr1 + *ptr2);
    return 0;
}
```

(A) 15

(B) 5

(C) Compilation error

(D) Runtime error

- Solution

    ```
    Answer: (A) 15
    Explanation: Adds the values pointed to by `ptr1` (5) and `ptr2` (10).
    ```

Question:

What will be printed by this code?

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

void printPointer(int* ptr) {
    cout << "Pointer address: " << ptr;
}

int main() {
    int var = 20;
    printPointer(&var);
}
```

(A) Address of `var` in hexadecimal

(B) Value of `var` (20)

(C) Compilation error

(D) Runtime error

- Solution

```
Answer: (A) Address of `var` in hexadecimal
Explanation: The function prints the memory address of `var`.
```

Question:

What is wrong with this code?

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

int main() {
    int* p1, p2; // p2 is not a pointer
    p1 = new int(10);
    cout << *p1 << " " << p2;
}
```

(A) Prints garbage for `p2`

(B) Compilation error (invalid dereference of `p2`)

(C) Memory leak

(D) Runtime error

- Solution

  ```
  Answer: (B) Compilation error
  Explanation: `p2` is an integer, not a pointer, so dereferencing it is invalid.
  ```

Question:

What does this code do?

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

void modifyArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
```

```
        arr[i] *= 2;
    }
}

int main() {
    int myArray[] = {1, 2, 3};
    modifyArray(myArray, 3);
    for (int i : myArray) cout << i << " ";
}
```

(A) Prints original array

(B) Prints doubled values (2, 4, 6)

(C) Compilation error

(D) Runtime error

- Solution

  > Answer: (B) Prints doubled values (2, 4, 6)
  > Explanation: The function modifies the array in-place by doubling each element.

Happy Coding!