Frequently Asked Pointer Interview Questions and Answers

(Part -1)

Prepared for Embedded Systems Professionals

Linkedin

Owner	UttamBasu
Author	Uttam Basu
Linkedin	www.linkedin.com/in/uttam-basu/

1. **What is a pointer in C?**
   A pointer is a variable that stores the memory address of another variable.

2. **How do you declare a pointer?**

```
int *ptr;
```
declares a pointer to an integer.

3. **How do you assign an address to a pointer?**

```
int a = 10;
int *ptr = &a;
```

4. **What does the * operator do in pointers?**
   It dereferences a pointer to access the value at the memory address it points to.

5. **What does the & operator do?**
   It gives the address of a variable.

6. **What is the output of the following code?**

```
int a = 5; int *p = &a;
printf("%d", *p);
```

7. **Can you have a pointer to a pointer?**
   Yes, using int **pp; allows a pointer to another pointer.

8. **What is a NULL pointer?**
   A pointer that points to nothing (int *p = NULL;).

9. **What happens if you dereference a NULL pointer?**
   It causes a segmentation fault or crash.

10. **How do you check if a pointer is NULL?**

```
if (ptr == NULL) { ... }
```

Pointer Arithmetic

11. **What is pointer arithmetic?**
    Performing arithmetic operations like ptr + 1 advances to the next element based on data type size.

12. **What is the size of a pointer in C?**
    It depends on the system (typically 4 bytes on 32-bit, 8 bytes on 64-bit).

13. **What does ptr++ do?**
    Moves the pointer to the next memory location of its type.

14. **Can you subtract one pointer from another?**
    Yes, if both point to elements in the same array.

15. **What is the result of &a + 1 where a is an int?**
    The address of the next int (4 bytes ahead typically).

Pointers and Arrays

16. **How are arrays and pointers related?**
    The name of the array is a pointer to its first element.

17. **How do you access array elements using pointers?**
    *(arr + i) is the same as arr[i].

18. **Can you change the base address of an array?**
    No, array names are constant pointers.

19. **What is a pointer to an array?**
    A pointer that holds the address of an array: int (*ptr)[5];

20. **How do you pass an array to a function using a pointer?**
    void func(int *arr); or void func(int arr[]);

Function Pointers

21. What is a function pointer?
A pointer that points to a function's address.

22. How do you declare a function pointer?
```
int (*fp)(int, int);
```

23. How do you assign a function to a function pointer?
```
fp = &add; or fp = add;
```

24. How do you call a function using a function pointer?
```
(*fp)(a, b); or fp(a, b);
```

25. What are function pointers used for?
Callbacks, dynamic function dispatching, plugin systems.

Pointer and Structures

26. How do you declare a pointer to a struct?
```
struct MyStruct *ptr;
```

27. How do you access struct members using a pointer?
```
Use the -> operator: ptr->member.
```

28. What is the difference between . and -> in structs?
. is for objects; -> is for pointers to objects.

29. Can you pass a struct pointer to a function?
Yes, using func(&myStruct);

30. How do you dynamically allocate memory for a struct?

```
struct MyStruct *ptr = malloc(sizeof(struct MyStruct));
```

Pointer Pitfalls & Best Practices

31. What is a dangling pointer?
A pointer pointing to freed or invalid memory.

32. How do you avoid dangling pointers?
Set pointer to `NULL` after freeing memory.

33. What is a wild pointer?
An uninitialized pointer that points to random memory.

34. How do you prevent wild pointers?
Always initialize pointers.

35. Can a pointer be used before initialization?
No, using uninitialized pointers is undefined behavior.

Advanced Pointers

36. What is pointer to a constant?
```
const int *ptr;
```
value pointed to cannot be changed.

37. What is constant pointer?
```
int *const ptr;
```
address in pointer cannot be changed.

38. What is constant pointer to constant?
```
const int *const ptr;
```
neither value nor address can change.

39. What is void pointer?
A generic pointer:
```
void *ptr;
```
can point to any data type.

40. How do you use a void pointer?
You must cast it before dereferencing.

Pointer Applications

41. Why use pointers in functions?
To modify actual arguments (pass-by-reference simulation).

42. How are pointers used in dynamic memory allocation?
Functions like `malloc()` return pointers.

43. Can you return a pointer from a function?
Yes, but not to a local variable (causes dangling pointer).

44. What is pointer decay?
When an array is passed to a function, it decays into a pointer.

45. How to swap two values using pointers?

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

46. What does *(*ptr) mean in double pointer?
It dereferences the pointer twice to get the actual value.

47. What is the use of const in pointers?
To protect data or pointer from modification.

48. Can a pointer point to a function returning a pointer?
Yes, with nested pointer declarations.

49. What is memory leak in context of pointers?
Allocated memory not freed, leading to wasted memory.

50. What is calloc() vs malloc()?
calloc() initializes memory to 0, malloc() does not.

Use Cases

51. How can you pass a pointer to a function?

```
void func(int *p);
func(&x);
```

52. Can pointers be used to implement polymorphism in C?
Yes, via function pointers in structures (used in driver code or object-like patterns).

53. Can you use pointer arithmetic on void * in C?
No, because the size of void is unknown — cast it first.

54. How do you return multiple values from a function using pointers?
By passing multiple pointers as arguments and modifying them.

55. What is the use of restrict keyword in pointers?
Tells the compiler that the pointer is the only reference to that memory, enabling optimizations.

56. Can a pointer point to a constant string?
Yes:

```
const char *str = "Hello";
```

57. How do you create a dynamic 1D array using pointers?

```
int *arr = malloc(n * sizeof(int));
```

58. How do you create a dynamic 2D array using pointers?

```
int **arr = malloc(rows * sizeof(int*));
for (int i = 0; i < rows; i++)
    arr[i] = malloc(cols * sizeof(int));
```

59. What is a generic pointer?
A pointer of type void * that can point to any data type.

60. What is sizeof(*ptr) vs sizeof(ptr)?
***ptr:** size of the object;
ptr: size of the pointer itself.

Tricky Pointer Concepts

61. What happens if you free a NULL pointer?
Nothing — it's safe and defined.

62. What is pointer aliasing?
Two or more pointers referencing the same memory location.

63. What is a flexible array member in structures?
Last member of a struct declared as `int arr[];` (no size specified).

64. Can you increment a function pointer?
No — undefined behavior.

65. What is segmentation fault in context of pointers?
Accessing invalid or restricted memory through a pointer.

66. Is char *str = "abc"; modifiable?
No — string literals are read-only.

67. Can pointers be used for buffer overflow attacks?
Yes — improper bounds checking can lead to such vulnerabilities.

68. How do you compare two pointers?
Use relational operators (==, !=, <, >), but only valid if both point to the same array.

69. Can you assign one pointer to another?
Yes: p2 = p1; — both point to the same address.

70. What happens if you free memory twice (double free)?
It leads to undefined behavior — can crash or corrupt memory.

71. How do you avoid memory leaks?
Free all dynamically allocated memory and set pointers to NULL.

72. What tools help detect pointer bugs in C?
Valgrind, AddressSanitizer, and gdb debugger.

73. Is this valid? int *p = NULL; *p = 10;
No — dereferencing NULL causes a crash.

74. Why do people cast `malloc()` return value in C?
It's unnecessary in C, but required in C++ or mixed C/C++ code.

75. What is a pointer map or lookup table?
An array of pointers used to store function addresses or data references dynamically.

76. Can we assign void * to int * directly?
In C, yes

```
int *p = (int *)vptr;
```

77. What's the meaning of char **argv in main()?
Array of string pointers passed as command-line arguments.

78. How do you implement a stack using pointers?
Use linked lists with dynamic memory allocation.

79. What is the base address of a pointer?
The actual address stored in the pointer variable.

80. How do you find if two pointers point to the same memory?
Compare them using ==.

81. How to convert array indexing to pointer form?
arr[i] ↔ *(arr + i)

82. Is int *a[10]; an array or a pointer?
It's an array of 10 pointers to int.

83. What is pointer decay in functions?
Array names decay into pointers when passed to functions.

84. Why is char *argv[] preferred over char **argv?
Both are same, but argv[] emphasizes array-like use.

85. What happens if we use an uninitialized pointer?
Results in undefined behavior — could crash or cause silent bugs.

Pointers in Function Arguments

86. Why are arrays passed as pointers to functions in C?
Because C passes by value, and arrays decay to pointers when passed.

87. How do you pass a 2D array to a function using pointers?
Specify column size:

```
void func(int (*arr)[COLS]);
```

88. Can you change the address a pointer parameter points to inside a function?
Only if you pass a pointer to the pointer:

```
void func(int **p);
```

89. What is the difference between char *p and char p[] in function parameters?
They are equivalent due to array-to-pointer decay.

90. Can you use a pointer to access global variables?
Yes, just take the address of the global variable.

91. How do you simulate output parameters using pointers?
By modifying the value at the passed address in a function.

92. What's the output of:

```
void update(int *a) {
    *a = 10;
}

int x = 5;
update(&x);
printf("%d", x);
```

Output: 10

93. Why do we use double pointers for dynamic 2D arrays?
Because we need an array of pointers, each pointing to an array.

94. Can function pointers be stored in arrays?
Yes: int (*ops[3])(int, int);

95. Why pass pointer to pointer to change the pointer's value itself?
Because passing a pointer by value copies it. To change the original, pass &ptr.

Code Understanding

96. What is the output?

```
int a = 10;
int *p = &a;
*p += 5;
printf("%d", a);
```

Output: 15

97. What is the output?

```
int a[] = {1, 2, 3};
int *p = a;
printf("%d", *(p + 2));
```

Output: 3

98. What is the output?

```
char *s = "ABC";
printf("%c", *(s + 1));
```

Output: B

99. Does this compile?

```
const int x = 5; int *p = &x;
```

No. A warning or error occurs due to discarding const qualifier.

100. **Can you assign NULL to an integer variable?**
No, NULL is for pointers only.

101. **Can we do int a = *(&a);?**
Yes. It gives the same value a.

102. **What is the address difference between two adjacent int in an array?**
Typically 4 bytes (depends on system architecture).

103. **What does *(arr + i) mean in pointer context?**
It accesses the i-th element of the array.

104. **What is printed?**

```
int a = 5, *p = &a;
printf("%p", p);
```

Output: Memory address of a.

105. **Does sizeof(*p) depend on pointer value or type?**
On **type**, not the value stored.

Memory & Allocation

106. **What are the common memory areas for pointers in C?**
Stack, heap, static/global, code/text segment.

107. **Where do function pointers reside in memory?**
Text/code segment.

108. **Where do dynamically allocated variables reside?**
Heap segment.

109. **How do you avoid memory leaks when using pointers?**
Free all dynamically allocated memory.

110. **What happens if you don't free allocated memory?**
Memory leak — memory remains reserved but unused.

111. **How do you reallocate an array to a larger size?**
Use realloc(ptr, new_size);

112. **Why should you always check if malloc() returned NULL?**
To ensure memory allocation was successful.

113. **What happens if you write beyond malloc'd memory?**
Undefined behavior — may crash or corrupt data.

114. **What's the use of calloc() over malloc()?**
It initializes allocated memory to zero.

115. **What is the output?**

```
int *p = malloc(sizeof(int));
*p = 20;
printf("%d", *p);
```

Output: 20

Pointer Operators & Const

1. **What does const int *p mean?**
Pointer to constant int — you can't change the value it points to.

2. **What does int *const p mean?**
Constant pointer to int — you can't change the pointer itself.

3. **What does const int *const p mean?**
Pointer and pointee both are constant.

4. **What does *p++ mean?**
Access value, then increment pointer.

5. **What does (*p)++ mean?**
Increment the value pointed to.

6. **What does ++*p mean?**
Same as (*p)++ — increment value.

7. **What does *&x mean?**
It resolves to x.

8. **What does &*p mean?**
It resolves to p.

9. **What does int *p = (int *)1000; do?**
Creates a pointer pointing to address 1000 — unsafe in real use.

10. **Can you cast between pointer types?**
    Yes, but use with caution. Unsafe casts can cause alignment issues.

Pointer Use in Data Structures & Complex Scenarios

1. **What is the role of pointers in implementing linked lists?**
   Each node uses a pointer to connect to the next node dynamically.

2. **How do you traverse a linked list using pointers?**
   Use a pointer that moves: ptr = ptr->next;

3. **How do you insert a node at the beginning of a linked list using pointers?**
   Create new node, set new->next = head, then head = new;

4. **How do you reverse a linked list using pointers?**
   Iteratively reassign .next using three pointers: prev, curr, next.

5. **How are pointers used in binary trees?**
   Each node has left and right child pointers.

6. **What is a pointer-based implementation of a stack?**
   Use linked list with push/pop using pointers.

7. **How do you implement a queue using pointers?**
   Use front and rear pointers for enqueue/dequeue.

8. **What are self-referential structures?**
   Structs that have pointers to the same type:

```
struct Node {
        int data;
        struct Node *next;
};
```

9. **What is a pointer to a structure?**
   A pointer variable that holds the address of a struct.

10. **Why are pointers crucial in implementing dynamic data structures?**
    They allow memory to be allocated at runtime and link elements together without fixed size.