IAE ACADEMY INSTITUTE FOR ENGINEERS

CORE JOBS · INTERVIEW QUESTIONS

IAE Academy free study material · Updated 30 October 2026

Programming in C Interview Questions

INSIDE

- 24 questions with short answers
- 6 numericals with the working
- Grouped by topic

Companies do not publish their interview questions; these are ours.

Online, with links to the notes: **iaeacademy.in/study/c-programming/interview-questions**

FREE COUNSELLING

Preparing for GATE or a core job? Talk to IAE Academy.

Or call **+91 84204 36524**

Every day, 10:30 am to 7:30 pm

Chat on WhatsApp

Free. Not for sale. Pass it on.

C fundamentals

1. What are integer promotions in C?

Before arithmetic, operands of type char or short are converted to int, or to unsigned int if int cannot hold all their values. So adding two chars is done in int. This is why the sum of two signed chars can exceed 127 inside the expression; it is converted, and may change, only when stored back in a char.

2. What is the difference between signed and unsigned integer overflow in C?

Signed overflow is undefined behaviour, so the compiler may assume it never happens and the result is not guaranteed. Arithmetic in unsigned int or a wider unsigned type is defined to wrap modulo 2^N for an N-bit type. So UINT_MAX + 1 gives 0, while INT_MAX + 1 has no defined result.

3. Why does the test `p != NULL && p->x > 0` not crash when p is NULL?

The && operator evaluates its left operand first and skips the right operand if the left is 0. There is a sequence point between them, so the dereference never runs for a null pointer. The operator || works the same way, stopping when its left operand is nonzero.

4. Why is an expression such as `i = i++ + ++i` undefined in C?

The object i is modified more than once with no sequence point between the changes, and the + operator does not fix the order in which its operands are evaluated. The standard gives no meaning to such code, so different compilers may print different results, and all are correct.

5. On a machine with an 8-bit unsigned char, `unsigned char c = 250;` is followed by `c = c + 10;`. What value does c hold afterwards?

The variable c holds 4, because the sum 260 is computed in int and then wraps modulo 256 when stored back into the 8-bit unsigned char.

> **WORKING**
>
> 1. c is promoted to int, so c + 10 is computed as 260.
> 2. Storing 260 in an unsigned char reduces it modulo $2^8 = 256$.
> 3. $260 \bmod 256 = 4$.

6. What happens in a switch statement when a case ends without a break?

Execution falls through into the statements of the next case label, whatever its value, and continues until a break or the end of the switch. Case labels are only jump targets, not separate blocks. A default label placed in the middle falls through in the same way.

Pointers, arrays and strings in C

7. What is array-to-pointer decay?

In most expressions an array name is converted to a pointer to its first element. The exceptions are the operand of sizeof, the operand of unary &, and a string literal that initialises an array. That is why passing an array to a function passes a pointer. A parameter declared as an array is adjusted to a pointer, so sizeof on it gives the pointer size.

8. What is the difference between `char *s = "hi";` and `char s[] = "hi";`?

The first makes a pointer to a string literal, which usually sits in read-only storage, so modifying it is undefined behaviour. The second makes an array holding its own modifiable copy of the 3 bytes. Also, sizeof gives the pointer size for the first and 3 for the second.

9. A two-dimensional array `int a[10][20]` starts at address 2000. An int takes 4 bytes and the array is stored row by row. What is the address of `a[3][5]`?

The address of a[3][5] is 2260. In row-major order 3 full rows of 20 elements come before it, plus 5 elements in its own row, so 65 elements of 4 bytes each are added to the base address 2000.

> **WORKING**
>
> 1. Element offset = $3 \times 20 + 5 = 65$ elements.
> 2. Byte offset = $65 \times 4 = 260$.
> 3. Address = $2000 + 260 = 2260$.

10. On a machine where an int is 4 bytes and must sit at an address divisible by 4, what is the sizeof of `struct { char a; int b; char c; }`?

The structure occupies 12 bytes: 1 for a, 3 of padding, 4 for b, 1 for c and 3 of tail padding so that arrays of it stay aligned.

> **WORKING**
>
> 1. a sits at offset 0, then 3 padding bytes so that b starts at offset 4.
> 2. b occupies offsets 4 to 7 and c sits at offset 8, so the members end at 9 bytes.
> 3. The size is rounded up to a multiple of the largest alignment, 4, giving 12.

11. How does pointer arithmetic work in C?

Adding n to a pointer of type T* moves it by n times sizeof(T) bytes, so p + 1 points to the next element. Subtracting two pointers into the same array gives the number of elements between them. Arithmetic is valid only inside the array or one position past its end.

12. What is the difference between malloc and calloc?

The call malloc(n) takes the total number of bytes and leaves the memory uninitialised. The call calloc(count, size) takes an element count and an element size, and sets every byte to zero. Both return NULL on failure, and both blocks must be released with free.

13. Why is it good practice to set a pointer to NULL after calling free on it?

After free the pointer still holds the old address, which is no longer valid, so using it is undefined behaviour. Setting it to NULL lets a later null check catch the misuse, and free(NULL) is defined to do nothing, so a second free through that pointer is harmless. Other copies of the old address still dangle.

14. What is the difference between `const int *p` and `int *const p`?

In `const int *p` the pointer can be changed to point elsewhere, but the int it points to cannot be modified through p. In `int *const p` the pointer itself is fixed after initialisation, but the int it points to can be changed. Read the declaration from the name outwards.

Functions, parameter passing, scope and storage classes

15. What does call by value mean in C, and how is a swap function written?

C passes a copy of each argument, so changing a parameter inside the function does not change the caller's variable. To swap two ints, pass their addresses and exchange the values through the pointers, for example with a temporary: t = *a; *a = *b; *b = t. The pointers themselves are still passed by value.

16. What is the difference between an auto local variable and a static local variable?

An auto local is created on each entry to its block and destroyed on exit, and holds garbage unless initialised. A static local lives for the whole program, is initialised once (to zero by default) and keeps its value between calls. Its name is still visible only inside its block.

17. What is the difference between static scoping and dynamic scoping, and which does C use?

With static (lexical) scoping, a name refers to the declaration found in the enclosing text where the function is written. With dynamic scoping, it refers to the most recent declaration in the chain of callers at run time. C uses static scoping, so a function never sees its caller's locals.

18. With `#define SQ(x) x*x`, the statement `int y = SQ(a+1);` runs with a = 3. What value does y get?

The variable y becomes 7, not 16, because the macro pastes the text a+1*a+1 and multiplication has higher precedence than addition. Defining the macro as ((x)*(x)) makes it give 16.

> **WORKING**
>
> 1. The preprocessor substitutes text, so SQ(a+1) becomes `a+1*a+1`.
> 2. Multiplication binds tighter than addition, so this is $a + (1 \times a) + 1$.
> 3. With a = 3 this is $3 + 3 + 1 = 7$.

19. What does the extern keyword do, and how does it differ from static at file scope?

An extern declaration without an initialiser declares an object that is defined elsewhere, so it allocates no storage and refers to that same object. A file-scope variable has external linkage by default. Marking it static gives it internal linkage, so other files cannot refer to it by name.

Recursion

20. What happens on the call stack when a recursive function runs, and why does it need a base case?

Each call gets its own activation record holding its parameters, locals and return address, stacked on the previous one. A base case returns without calling again, so the stack stops growing and then unwinds. Without it the calls never end and the stack overflows.

21. Why does a printf placed before the recursive call print values in the opposite order to one placed after it?

A statement before the call runs on the way in, as each call starts, so values appear in call order. A statement after the call runs on the way out, as calls return, which happens in reverse order because the last call finishes first. This is the last-in, first-out behaviour of the stack.

22. What is the difference between tail recursion and non-tail recursion?

In tail recursion the recursive call is the last action, so nothing remains to do when it returns, and a compiler may reuse the frame, though C does not guarantee this. In non-tail recursion work is pending after the call, as in n * f(n-1), so the stack grows with the depth.

23. How many moves does the standard recursive Tower of Hanoi solution make for 5 disks?

The recursion makes 31 moves for 5 disks. Moving n disks takes two moves of n-1 disks plus one move of the largest, so $T(n) = 2T(n-1) + 1$, which solves to $2^n - 1$.

WORKING

1. Moving n disks needs two moves of n-1 disks and one move of the largest: $T(n) = 2T(n-1) + 1$.
2. With $T(0) = 0$ the solution is $T(n) = 2^n - 1$.
3. $T(5) = 2^5 - 1 = 31$.

24. A naive recursive function has fib(0) = 0 and fib(1) = 1, and otherwise returns fib(n-1) + fib(n-2). How many calls in total does fib(5) make, counting the first call?

The call fib(5) makes 15 calls in total. Each call counts itself plus the calls made by its two sub-calls, so $c(n) = 1 + c(n-1) + c(n-2)$ with $c(0) = c(1) = 1$, which gives 15.

WORKING

1. Let $c(n)$ be the number of calls: $c(n) = 1 + c(n-1) + c(n-2)$, with $c(0) = c(1) = 1$.
2. $c(2) = 3$, $c(3) = 5$, $c(4) = 9$.
3. $c(5) = 1 + 9 + 5 = 15$, matching $2F_6 - 1 = 2 \times 8 - 1$.

FROM IAE ACADEMY

Preparing for GATE or a core job?

IAE Academy teaches GATE in seven papers in Ballygunge, Kolkata, and online. Counselling is free: ask us about batches, fees or your preparation.

FREE COUNSELLING

Chat on WhatsApp

Call +91 84204 36524

Open every day, 10:30 am to 7:30 pm.

Scan to chat on WhatsApp

GATE COURSES

Classroom batches in Ballygunge and online batches. Fees and details are on the course pages.

Scan: courses

See the courses

FREE ON IAEACADEMY.IN

Notes and solved previous-year questions for Programming in C, topic by topic.

Scan: the notes

Open the notes

Found this useful? Send it to a friend preparing for core-job interviews.

Send to a friend on WhatsApp

IAE Academy free study material. Companies do not publish their interview questions; these are ours. Prepared and published by IAE Academy, Kolkata. Free to read and free to share, unaltered and with attribution.

IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE coaching in Kolkata since 2012 · EC, EE, IN, CS and IT, DA, BT and CE · in the classroom and online

227A Rash Behari Ave, Ballygunge, Kolkata 700019

+91 84204 36524 · iaeacademy.in · Updated 30 October 2026

Get directions