**C interview Q/A**

**1.What is C language?**

C is a general-purpose, procedural programming language developed by Dennis Ritchie in the early 1970s at Bell Labs. It is widely used for system programming, developing operating systems, and applications because of its efficiency, simplicity, and close-to-hardware nature.

**2. Why is C called a middle-level language?**

C is considered a middle-level language because:

* It bridges the gap between **low-level programming** (like assembly language) and **high-level programming** (like Python or Java).
* It provides features to directly access hardware resources (low-level) and high-level abstractions like functions and structures.

**3. What are the key features of C?**

* **Simple and efficient**: Easy to learn and provides fast execution.
* **Rich library**: Comes with a variety of built-in functions.
* **Portability**: Code can be executed on various platforms with minimal changes.
* **Memory management**: Supports manual allocation and deallocation.
* **Pointers**: Enables direct memory access.
* **Modularity**: Code can be divided into functions and modules.
* **Structured programming**: Supports control structures like loops, conditionals, etc.
* **Extensibility**: Can extend functionality by writing custom libraries.

**4. What is the difference between a compiler and an interpreter?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Compiler** | **Interpreter** |
| **Definition** | Converts the entire source code into machine code before execution. | Converts and executes code line by line. |
| **Execution** | Faster, as the code is precompiled. | Slower, as it interprets code in runtime. |
| **Output** | Generates an executable file. | Does not generate an executable file. |
| **Example** | C, C++ | Python, JavaScript |

**5. Explain the structure of a C program.**

A typical C program structure:

1. **Preprocessor directives**: #include <stdio.h>
2. **Global declarations**: Variables, constants, or functions outside main().
3. **Main function**: Entry point of execution, int main().
4. **Statements and logic**: Code inside the main or other functions.
5. **Return statement**: Ends the program and optionally returns a value.

Example:

#include <stdio.h> // Preprocessor directive

int globalVar = 10; // Global declaration

int main() { // Main function

printf("Hello, World!\n"); // Statement

return 0; // Return statement

}

**6. What are tokens in C?**

Tokens are the smallest elements of a C program that have meaning. Examples include:

* **Keywords** (int, if, return)
* **Identifiers** (variable/function names)
* **Constants** (5, 'a')
* **Operators** (+, -, \*)
* **Special symbols** ({, }, ;)

**7. What are keywords in C? Name a few.**

Keywords are reserved words in C with predefined meanings, and they cannot be used as identifiers.  
**Examples**:  
int, float, return, if, else, void, while, break, continue.

**8. What are variables in C?**

Variables are named memory locations used to store data. Their values can be changed during program execution.  
**Example**:

int age = 25; // 'age' is a variable storing 25.

**9. What are the different data types in C?**

C provides the following data types:

* **Basic types**: int, char, float, double
* **Derived types**: Arrays, pointers, functions
* **User-defined types**: struct, union, enum
* **Void type**: void

**10. What is the difference between signed and unsigned variables?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Signed** | **Unsigned** |
| **Range** | Can hold positive and negative values. | Only positive values. |
| **Example (int)** | -32,768 to 32,767 | 0 to 65,535 |
| **Keyword** | Default (e.g., int x;) | Prefix unsigned (e.g., unsigned int x;) |

**11. What is the difference between float and double in C?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **float** | **double** |
| **Size** | 4 bytes | 8 bytes |
| **Precision** | Single precision (6-7 digits) | Double precision (15-16 digits) |
| **Use case** | When memory is limited. | For high-precision calculations. |

**12. What is a constant in C? How is it declared?**

A constant is a variable whose value cannot be changed after initialization.  
**Declaration**:

1. Using const:

const int PI = 3.14;

1. Using #define:

#define PI 3.14

**13. What is type casting in C?**

Type casting is converting a variable from one data type to another.  
**Example**:

float f = 3.14;

int x = (int)f; // Type casting float to int

**14. What is the difference between implicit and explicit type casting?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Implicit Type Casting** | **Explicit Type Casting** |
| **Definition** | Done automatically by the compiler. | Done manually by the programmer. |
| **Example** | int x = 5; float y = x; | float x = 5.5; int y = (int)x; |

**15. What is a storage class?**

A storage class defines the scope, lifetime, and visibility of a variable. It determines how memory is allocated and where the variable can be accessed.

**16. What are the types of storage classes in C?**

1. **Automatic (auto)**: Default for local variables.
2. **Static (static)**: Retains value between function calls.
3. **External (extern)**: Declares a global variable accessible across files.
4. **Register (register)**: Suggests storing in CPU registers for fast access.

**17. What is the scope of a variable in C?**

The scope of a variable defines the region in the program where it can be accessed.

* **Local scope**: Inside a block or function.
* **Global scope**: Accessible throughout the program.
* **Block scope**: Within {} brackets.

**18. What are header files in C?**

Header files contain definitions of functions and macros. Common examples:

* <stdio.h>: Standard input/output functions.
* <stdlib.h>: General utility functions like memory allocation.
* <math.h>: Mathematical functions.

**19. What is the purpose of the main() function?**

The main() function is the entry point of a C program where execution begins. It usually returns an integer value to indicate the program's termination status.  
Example:

int main() {

return 0; // Indicates successful execution

}

**20. Explain the use of the return statement in C.**

The return statement:

* Ends the execution of a function.
* Returns a value to the calling function.
* In main(), it signifies the program's exit status (0 for success, non-zero for errors).  
  **Example**:

int sum(int a, int b) {

return a + b; // Returns the sum

}

**21. What are control statements in C?**

Control statements are used to control the flow of execution in a program. They enable decision-making, looping, and jumping between different parts of the program.  
**Types of control statements**:

1. **Decision-making statements**: if, if-else, switch.
2. **Looping statements**: for, while, do-while.
3. **Jump statements**: break, continue, goto, return.

**22. What is the difference between if and switch?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **if Statement** | **switch Statement** |
| **Condition Type** | Tests logical conditions using relational or logical operators. | Tests values against constants (equality). |
| **Flexibility** | Can evaluate a range of conditions. | Works only with discrete values. |
| **Syntax** | Supports complex conditions (if, else if). | Works with case labels for each condition. |
| **Performance** | Slower for multiple conditions. | Faster for large decision trees due to jump tables. |

**23. Explain nested if-else statements.**

Nested if-else means placing one if or if-else inside another. It allows testing multiple conditions.  
**Example**:

int num = 15;

if (num > 0) {

if (num % 2 == 0) {

printf("Positive even number\n");

} else {

printf("Positive odd number\n");

}

} else {

printf("Number is non-positive\n");

}

**24. What are loops in C?**

Loops execute a block of code repeatedly as long as a condition is true.  
**Types of loops in C**:

1. **for loop**: Executes a block a fixed number of times.
2. **while loop**: Executes as long as a condition is true.
3. **do-while loop**: Executes at least once before checking the condition.

**25. Explain the for loop with an example.**

A for loop is used for definite iteration.  
**Syntax**:

for (initialization; condition; increment/decrement) {

// Code block

}

**Example**:

for (int i = 1; i <= 5; i++) {

printf("%d\n", i);

}

**26. How does the while loop work?**

A while loop repeatedly executes a block of code as long as the condition evaluates to true.  
**Syntax**:

while (condition) {

// Code block

}

**Example**:

int i = 1;

while (i <= 5) {

printf("%d\n", i);

i++;

}

**27. What is the difference between while and do-while?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **while Loop** | **do-while Loop** |
| **Condition Check** | Condition is checked before execution. | Condition is checked after execution. |
| **Execution** | May not execute if the condition is false initially. | Executes at least once. |
| **Syntax** | while (condition) {} | do { } while (condition); |

**28. What is a break statement?**

The break statement is used to terminate a loop or a switch statement prematurely.  
**Example**:

for (int i = 1; i <= 10; i++) {

if (i == 5) break;

printf("%d\n", i);

}

**29. What is a continue statement?**

The continue statement skips the current iteration of a loop and proceeds to the next iteration.  
**Example**:

for (int i = 1; i <= 5; i++) {

if (i == 3) continue;

printf("%d\n", i);

}

**30. What is the goto statement?**

The goto statement allows jumping to a labeled part of the program. It is rarely used due to readability concerns.  
**Example**:

int x = 5;

if (x == 5) goto label;

printf("This won't print.\n");

label:

printf("Jumped to label!\n");

**31. What are labels in C?**

Labels are identifiers followed by a colon (:). They are used with goto statements to mark a location in the program.  
**Example**:

label:

printf("This is a labeled statement.\n");

goto label; // Infinite loop

**32. What is a conditional operator in C?**

The conditional operator (?:) is a shorthand for if-else.  
**Syntax**:

condition ? expression1 : expression2;

**Example**:

int x = 10, y = 20;

int max = (x > y) ? x : y;

printf("Maximum: %d\n", max);

**33. Explain the sizeof() operator with an example.**

The sizeof() operator returns the size (in bytes) of a data type or variable.  
**Example**:

int x = 5;

printf("Size of int: %zu bytes\n", sizeof(x)); // Outputs: 4 bytes (typically)

**34. What is the difference between =, ==, and ===?**

* =: Assignment operator (e.g., x = 5; assigns 5 to x).
* ==: Equality operator (e.g., x == 5 checks if x equals 5).
* ===: Not used in C (specific to JavaScript).

**35. What is the difference between && and &?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **&& (Logical AND)** | **& (Bitwise AND)** |
| **Operation** | Logical operation on conditions. | Bitwise operation on bits. |
| **Usage** | if (a && b) | c = a & b |
| **Evaluation** | Short-circuits if the first condition is false. | Evaluates all bits. |

**36. What is the difference between || and |?**

| **Aspect** | **|| (Logical OR)** | **| (Bitwise OR)** | |--------------------|--------------------------------------|-------------------------------------| | **Operation** | Logical operation on conditions. | Bitwise operation on bits. | | **Usage** | if (a || b) | c = a | b | | **Evaluation** | Short-circuits if the first condition is true. | Evaluates all bits. |

**37. How do you implement a nested loop in C?**

A nested loop is a loop inside another loop.  
**Example**:

for (int i = 1; i <= 3; i++) {

for (int j = 1; j <= 2; j++) {

printf("i=%d, j=%d\n", i, j);

}

}

**38. What is an infinite loop? Give an example.**

An infinite loop is a loop that never terminates because its condition is always true.  
**Example**:

while (1) {

printf("This loop runs forever.\n");

}

**39. Can switch statements be nested?**

Yes, switch statements can be nested, though it is rarely done due to complexity.  
**Example**:

switch (a) {

case 1:

switch (b) {

case 2: printf("a=1, b=2\n"); break;

}

break;

}

**40. Explain the concept of jump statements in C.**

Jump statements transfer control to other parts of the program. Types:

1. **break**: Exits a loop or switch.
2. **continue**: Skips the current loop iteration.
3. **goto**: Jumps to a labeled statement.
4. **return**: Exits from a function, optionally returning a value.

**41.What is a function in C?**

A function in C is a block of code that performs a specific task. It helps in breaking down a program into smaller, manageable parts, improving reusability and readability.  
**Example**:

int add(int a, int b) {

return a + b;

}

**42. What is the difference between function declaration and function definition?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Function Declaration** | **Function Definition** |
| **Purpose** | Declares the function's name, return type, and parameters to the compiler. | Provides the actual implementation of the function. |
| **Syntax** | Ends with a semicolon. | Includes the function body with logic. |
| **Example** | int add(int, int); | int add(int a, int b) { return a + b; } |

**43. How are arguments passed in C (call by value vs. call by reference)?**

1. **Call by value**: A copy of the actual parameter is passed to the function. Changes made do not affect the original variable.  
   **Example**:

void modify(int x) { x = 10; }

1. **Call by reference**: The address of the parameter is passed. Changes made affect the original variable.  
   **Example**:

void modify(int \*x) { \*x = 10; }

**44. What is recursion? Provide an example.**

Recursion is a process where a function calls itself directly or indirectly.  
**Example**: Factorial of a number:

int factorial(int n) {

if (n == 0) return 1; // Base case

return n \* factorial(n - 1); // Recursive case

}

**45. What is the difference between recursion and iteration?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Recursion** | **Iteration** |
| **Definition** | A function calls itself. | A block of code repeats using loops. |
| **Termination** | Requires a base case. | Terminates when the loop condition fails. |
| **Memory Use** | Consumes more memory due to function calls. | More efficient in memory usage. |

**46. Can a function return multiple values in C?**

No, a function cannot directly return multiple values. However, you can use:

* Pointers to modify multiple variables.
* Structures to encapsulate multiple values.

**Example using pointers**:

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

**47. What is the purpose of the void keyword?**

* Specifies that a function does not return any value.  
  **Example**: void display() { printf("Hello!"); }
* Declares a pointer to an unknown type (void \*). **Example**: void \*ptr;

**48. What are arrays in C?**

An array is a collection of elements of the same data type, stored in contiguous memory locations.  
**Example**:

int arr[5] = {1, 2, 3, 4, 5};

**49. How do you declare and initialize an array in C?**

**Declaration**:

int arr[5];

**Initialization**:

int arr[5] = {1, 2, 3, 4, 5};

**50. What is the difference between one-dimensional and two-dimensional arrays?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **One-Dimensional** | **Two-Dimensional** |
| **Structure** | A single row or column of elements. | Rows and columns of elements (matrix). |
| **Syntax** | int arr[5]; | int arr[3][4]; |

**51. How do you calculate the address of an element in a two-dimensional array?**

The address of an element at [i][j] in a 2D array can be calculated as:

Address = Base\_Address + ((i \* Number\_of\_Columns) + j) \* Element\_Size

**52. What is the difference between arrays and pointers?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Arrays** | **Pointers** |
| **Definition** | Fixed-size collection of elements. | Variable that stores an address. |
| **Size** | Fixed at compile-time. | Can dynamically point to any size. |
| **Access** | Access via indices. | Access via dereferencing (\*). |

**53. How can you pass arrays to functions in C?**

Arrays are passed by reference (using their base address).  
**Example**:

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

}

**54. Can you return an array from a function?**

C does not allow returning arrays directly, but you can return pointers or use static arrays.  
**Example**:

int\* createArray() {

static int arr[5] = {1, 2, 3, 4, 5};

return arr;

}

**55. How do you dynamically allocate memory for arrays?**

Use malloc or calloc to allocate memory dynamically.  
**Example**:

int \*arr = (int \*)malloc(5 \* sizeof(int));

**56. What is a character array in C?**

A character array stores a sequence of characters (a string).  
**Example**:

char str[6] = "Hello"; // Includes null character

**57. How do you reverse an array?**

**Example**:

void reverse(int arr[], int n) {

for (int i = 0; i < n / 2; i++) {

int temp = arr[i];

arr[i] = arr[n - i - 1];

arr[n - i - 1] = temp;

}

}

**58. How do you find the largest and smallest elements in an array?**

**Example**:

void findMinMax(int arr[], int n, int \*min, int \*max) {

\*min = \*max = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > \*max) \*max = arr[i];

if (arr[i] < \*min) \*min = arr[i];

}

}

**59. How do you sort an array?**

Use algorithms like **Bubble Sort**, **Selection Sort**, or library functions like qsort.  
**Example using Bubble Sort**:

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

**60. What is the difference between strcpy() and strncpy()?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **strcpy()** | **strncpy()** |
| **Purpose** | Copies a string to another. | Copies a string up to a specified length. |
| **Null Termination** | Ensures null-termination. | May not null-terminate if length is exceeded. |
| **Example** | strcpy(dest, src); | strncpy(dest, src, n); |

### ****61.What is a function in C?****

A function in C is a block of code that performs a specific task. It helps in breaking down a program into smaller, manageable parts, improving reusability and readability.  
**Example**:

int add(int a, int b) {

return a + b;

}

### ****62. What is the difference between function declaration and function definition?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Function Declaration** | **Function Definition** |
| **Purpose** | Declares the function's name, return type, and parameters to the compiler. | Provides the actual implementation of the function. |
| **Syntax** | Ends with a semicolon. | Includes the function body with logic. |
| **Example** | int add(int, int); | int add(int a, int b) { return a + b; } |

### ****63. How are arguments passed in C (call by value vs. call by reference)?****

1. **Call by value**: A copy of the actual parameter is passed to the function. Changes made do not affect the original variable.  
   **Example**:

void modify(int x) { x = 10; }

1. **Call by reference**: The address of the parameter is passed. Changes made affect the original variable.  
   **Example**:

void modify(int \*x) { \*x = 10; }

### ****64. What is recursion? Provide an example.****

Recursion is a process where a function calls itself directly or indirectly.  
**Example**: Factorial of a number:

int factorial(int n) {

if (n == 0) return 1; // Base case

return n \* factorial(n - 1); // Recursive case

}

### ****65. What is the difference between recursion and iteration?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Recursion** | **Iteration** |
| **Definition** | A function calls itself. | A block of code repeats using loops. |
| **Termination** | Requires a base case. | Terminates when the loop condition fails. |
| **Memory Use** | Consumes more memory due to function calls. | More efficient in memory usage. |

### ****66. Can a function return multiple values in C?****

No, a function cannot directly return multiple values. However, you can use:

* Pointers to modify multiple variables.
* Structures to encapsulate multiple values.

**Example using pointers**:

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

### ****67. What is the purpose of the**** void ****keyword?****

* Specifies that a function does not return any value.  
  **Example**: void display() { printf("Hello!"); }
* Declares a pointer to an unknown type (void \*). **Example**: void \*ptr;

### ****68. What are arrays in C?****

An array is a collection of elements of the same data type, stored in contiguous memory locations.  
**Example**:

int arr[5] = {1, 2, 3, 4, 5};

### ****69. How do you declare and initialize an array in C?****

**Declaration**:

int arr[5];

**Initialization**:

int arr[5] = {1, 2, 3, 4, 5};

### ****70. What is the difference between one-dimensional and two-dimensional arrays?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **One-Dimensional** | **Two-Dimensional** |
| **Structure** | A single row or column of elements. | Rows and columns of elements (matrix). |
| **Syntax** | int arr[5]; | int arr[3][4]; |

### ****71. How do you calculate the address of an element in a two-dimensional array?****

The address of an element at [i][j] in a 2D array can be calculated as:

Address = Base\_Address + ((i \* Number\_of\_Columns) + j) \* Element\_Size

### ****72. What is the difference between arrays and pointers?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Arrays** | **Pointers** |
| **Definition** | Fixed-size collection of elements. | Variable that stores an address. |
| **Size** | Fixed at compile-time. | Can dynamically point to any size. |
| **Access** | Access via indices. | Access via dereferencing (\*). |

### ****73. How can you pass arrays to functions in C?****

Arrays are passed by reference (using their base address).  
**Example**:

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

}

### ****74. Can you return an array from a function?****

C does not allow returning arrays directly, but you can return pointers or use static arrays.  
**Example**:

int\* createArray() {

static int arr[5] = {1, 2, 3, 4, 5};

return arr;

}

### ****75. How do you dynamically allocate memory for arrays?****

Use malloc or calloc to allocate memory dynamically.  
**Example**:

int \*arr = (int \*)malloc(5 \* sizeof(int));

### ****76. What is a character array in C?****

A character array stores a sequence of characters (a string).  
**Example**:

char str[6] = "Hello"; // Includes null character

### ****77. How do you reverse an array?****

**Example**:

void reverse(int arr[], int n) {

for (int i = 0; i < n / 2; i++) {

int temp = arr[i];

arr[i] = arr[n - i - 1];

arr[n - i - 1] = temp;

}

}

### ****78. How do you find the largest and smallest elements in an array?****

**Example**:

void findMinMax(int arr[], int n, int \*min, int \*max) {

\*min = \*max = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > \*max) \*max = arr[i];

if (arr[i] < \*min) \*min = arr[i];

}

}

### ****79. How do you sort an array?****

Use algorithms like **Bubble Sort**, **Selection Sort**, or library functions like qsort.  
**Example using Bubble Sort**:

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

### ****80. What is the difference between**** strcpy() ****and**** strncpy()****?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **strcpy()** | **strncpy()** |
| **Purpose** | Copies a string to another. | Copies a string up to a specified length. |
| **Null Termination** | Ensures null-termination. | May not null-terminate if length is exceeded. |
| **Example** | strcpy(dest, src); | strncpy(dest, src, n); |

### ****81. What is a pointer in C?****

A pointer is a variable that stores the address of another variable. It is used for dynamic memory allocation, arrays, functions, and more.

### ****82. What is the syntax for declaring a pointer?****

**Syntax**:

data\_type \*pointer\_name;

**Example**:

int \*ptr;

### ****83. How do you use pointers to access array elements?****

Pointers can be used to access array elements by incrementing the pointer.  
**Example**:

int arr[3] = {10, 20, 30};

int \*ptr = arr;

printf("%d", \*(ptr + 1)); // Accesses arr[1]

### ****84. What is a null pointer?****

A null pointer is a pointer that does not point to any memory location.  
**Example**:

int \*ptr = NULL;

### ****85. What is a void pointer?****

A void pointer (void \*) can hold the address of any data type but cannot be directly dereferenced.  
**Example**:

void \*ptr;

int x = 10;

ptr = &x;

### ****86. What are dangling pointers?****

A dangling pointer points to a memory location that has been deallocated or is out of scope.  
**Example**:

int \*ptr = (int \*)malloc(sizeof(int));

free(ptr); // Now ptr is dangling

### ****87. What are wild pointers?****

Wild pointers are uninitialized pointers that point to arbitrary memory locations, leading to undefined behavior.  
**Example**:

int \*ptr; // Wild pointer

### ****88. How do you pass pointers to functions?****

Pointers can be passed to functions to allow modification of the original data.  
**Example**:

void modify(int \*x) {

\*x = 10;

}

int main() {

int a = 5;

modify(&a);

}

### ****89. How can you use pointers to return multiple values from a function?****

By passing multiple pointers to a function, multiple values can be modified.  
**Example**:

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

### ****90. What is a function pointer?****

A function pointer stores the address of a function and can be used to call it.  
**Example**:

void display() { printf("Hello!"); }

void (\*funcPtr)() = display;

funcPtr(); // Calls display()

### ****91. What is the difference between**** \*ptr++ ****and**** ++\*ptr****?****

* \*ptr++: Dereferences the current pointer and then increments the pointer.
* ++\*ptr: Increments the value pointed to by the pointer.

### ****92. What is a structure in C?****

A structure is a user-defined data type that groups variables of different types under one name.  
**Example**:

struct Point {

int x;

int y;

};

### ****93. How do you declare and initialize a structure?****

**Declaration**:

struct Point p1;

**Initialization**:

struct Point p1 = {10, 20};

### ****94. What is the difference between**** struct ****and**** typedef struct****?****

* struct: Requires the struct keyword to declare variables.
* typedef struct: Defines a new type name to avoid using struct.  
  **Example**:

typedef struct { int x; int y; } Point;

Point p1;

### ****95. How do you access members of a structure using pointers?****

Use the arrow operator (->).  
**Example**:

struct Point { int x, y; };

struct Point p1 = {10, 20};

struct Point \*ptr = &p1;

printf("%d", ptr->x); // Accesses p1.x

### ****96. What is a union in C?****

A union is similar to a structure but shares memory among all its members.  
**Example**:

union Data { int i; float f; };

### ****97. How is a union different from a structure?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Structure** | **Union** |
| **Memory** | Allocates memory for all members. | Shares memory among members. |
| **Access** | All members can store values simultaneously. | Only one member can store a value at a time. |

### ****98. What is the size of a union?****

The size of a union is equal to the size of its largest member.

### ****99. Can structures and unions contain arrays as members?****

Yes, both can have arrays as members.  
**Example**:

struct Point { int arr[5]; };

### ****100. What is the use of bit fields in structures?****

Bit fields allow allocating a specific number of bits to a structure member, saving memory.  
**Example**:

struct Flags { unsigned int a:1; unsigned int b:2; };

### ****1. What is dynamic memory allocation in C?****

Dynamic memory allocation allows allocating memory at runtime using functions like malloc, calloc, and realloc.

### ****22. What is the difference between**** malloc() ****and**** calloc()****?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **malloc** | **calloc** |
| **Initialization** | Does not initialize memory. | Initializes memory to zero. |
| **Syntax** | malloc(size) | calloc(num, size) |

### ****23. What is**** realloc() ****used for?****

realloc() is used to resize previously allocated memory.  
**Example**:

int \*ptr = malloc(5 \* sizeof(int));

ptr = realloc(ptr, 10 \* sizeof(int));

### ****24. How is memory deallocated in C?****

Use free() to release dynamically allocated memory.  
**Example**:

free(ptr);

### ****25. What is a memory leak? How can it be prevented?****

A memory leak occurs when allocated memory is not deallocated.  
**Prevention**:

* Always use free() after malloc or calloc.
* Use tools like Valgrind to detect leaks.

### ****26. What is a segmentation fault?****

A segmentation fault occurs when a program accesses restricted or invalid memory locations.

### ****27. What are preprocessor directives?****

Preprocessor directives are commands processed before compilation, starting with #.  
**Example**:

#include <stdio.h>

#define PI 3.14

### ****28. What is the difference between**** #define ****and**** const****?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **#define** | **const** |
| **Compilation** | Handled by the preprocessor. | Handled by the compiler. |
| **Type Safety** | No type checking. | Enforces type checking. |

### ****29. What is the use of**** #include****?****

#include is used to include header files in a program.

### ****30. What is conditional compilation?****

Conditional compilation includes/excludes code based on conditions.  
**Example**:

#ifdef DEBUG

printf("Debugging mode");

#endif

### ****31. How do macros work in C?****

Macros are defined using #define and replaced during preprocessing.  
**Example**:

#define SQUARE(x) ((x) \* (x))

### ****32. What is the difference between macros and functions?****

* **Macros**: No type checking; expanded inline.
* **Functions**: Type-checked; compiled separately.

### ****33. What is the purpose of the**** #undef ****directive?****

#undef removes a macro definition.  
**Example**:

#define PI 3.14

#undef PI

### ****34. What are inline functions?****

Inline functions are expanded at the call site, reducing function call overhead.  
**Example**:

inline int square(int x) { return x \* x; }

### ****35. What is the purpose of the**** volatile ****keyword?****

volatile tells the compiler that a variable can change unexpectedly, preventing optimization.  
**Example**:

volatile int flag;

### ****36. What is the purpose of the**** extern ****keyword?****

extern declares a variable or function defined in another file.  
**Example**:

extern int x;

### ****37. What is the difference between**** static ****and**** extern****?****

|  |  |  |
| --- | --- | --- |
| **Aspect** | **static** | **extern** |
| **Scope** | Limited to the file or function. | Accessible across multiple files. |

### ****38. What is a file pointer?****

A file pointer (FILE \*) is used to access and manipulate files in C.  
**Example**:

FILE \*fp = fopen("file.txt", "r");

### ****39. How do you read and write to a file in C?****

**Reading**:

fscanf(fp, "%d", &x);

**Writing**:

fprintf(fp, "%d", x);

### ****40. How do you handle errors in file operations?****

Use ferror() or check if the file pointer is NULL.  
**Example**:

if (fp == NULL) {

perror("Error opening file");

}

Top of Form

Bottom of Form

Top of Form

Bottom of Form