Q. Why is C a High-Level Language?
Ans: High-level languages are programming languages that are used for writing programs or software that can be understood by humans and computers. High-level languages are easier to understand for humans because they use a lot of symbols and letter phrases to represent logic and instructions in a program. C is a programmer-friendly language, less memory efficient, easy to understand, simple to maintain, portable, and can run on any platform and needs a compiler for translation. So, C is a high-level programming language.

Q. What is a compiler and how does it work?
Ans: A compiler is a special program that typically takes a high-level language code as input and converts the input to a lower-level language at once. It lists all the errors if the input code does not follow the rules of its language. The process is much faster than interpreter but it becomes difficult to debug all the errors together in a program.
The process of compiling code involves several steps. Code passes through these steps sequentially and if there is any mistake in the code then it will be examined through these steps and thus compilation process stops in between and shows a compilation error. Otherwise, if everything is ok then the compiler does not show any error and compiles the source code.

Q. Features of C Programming Language:
Ans:

- **C is a structured programming language:** Structured programming is a programming paradigm that facilitates the creation of a program with readable code and reusable components. C is called structured programming language because a program in c language can be divided into small logical functional modules or structures with the help of function procedures.
- **C is a general-purpose language:** A general-purpose programming language is a programming language for building software in a wide variety of application domains. C is widely used for developing system software, application software, and embedded systems and provides a straightforward, consistent, and powerful

interface for programming systems. That's why, we can call that C is a general-purpose language.

- **C is case-sensitive language:** C language treats upper-case characters and lower-case characters as different. The reason why c language is case sensitive is that the keywords in C are case sensitive. Case-sensitive design is simpler for computers.

Q. What is a header file in C?
Ans: In C programming, a header file is a piece of code that contains declarations of functions, variables, constants, and other entities that are used in multiple source files. Header files have a '.h' extension and are included at the beginning of source code files using the '#include' preprocessor directive.

Q. What is main function in C?
Ans: Every C program has a primary function that must be named main. The main function serves as the starting point for program execution. It usually controls program execution by directing the calls to other functions in the program. Every program must have exactly one main function. If you use more than one main function the compiler can not understand which one marks the beginning of the program.

Q. What is an Identifier in C?
Ans: An identifier is a name given to various program elements such as variables, functions, arrays, constants, labels, and user-defined data types. User-defined name must start with a letter or '_', middle characters are letters, digits, or underscore('_'), must not match with keywords, maximum 31 characters. In an identifier, upper & lowercase are treated as distinct examples count, Count & COUNT are three separate identifiers.

Q. Keyword: In C programming, a keyword is a reserved word that has a predefined meaning and purpose within the language. Keywords are an integral part of the C language syntax and are used to define the structure, behavior, and flow of C programs. They cannot be used as identifiers (variable names, function names, etc.) because they are reserved for specific language features. C has 32 keywords such as auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for,

goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void.

(you don’t need to memorize all keywords. Just read these keyword's names in case, if you are asked)

Q. Declaration of Variables:
Ans:

- It tells the compiler what the variable name is.
- It specifies what type of data the variable will hold.
- Format:
    - data_type var_name;
    - data_type var_name1, var_name2, var_name3 etc

Q. How to add a comment in C:
Ans: There are two types of comments in C. The first single-line comment is //. Everything from the // to the end of the line is a comment. The second is a multi-line comment where use /* to start the comment and */ to end the comment.

Q. Operator Precedence in C:
ANS:
Here's a list of operator precedence in C, from highest to lowest:

1. Parentheses: ()
2. Array Subscript: []
3. Function Call: ()
4. Postfix Increment/Decrement: ++ --
5. Unary Increment/Decrement: ++ --
6. Unary Plus/Minus: + -
7. Logical NOT: !
8. Bitwise NOT: ~
9. Dereference (Pointer): *

10. Address-of (Pointer): &
11. Multiplication: *
12. Division: /
13. Modulus: %
14. Addition: +
15. Subtraction: -
16. Left Shift: <<
17. Right Shift: >>
18. Less Than: <
19. Less Than or Equal: <=
20. Greater Than: >
21. Greater Than or Equal: >=
22. Equality: ==
23. Inequality: !=
24. Bitwise AND: &
25. Bitwise XOR: ^
26. Bitwise OR: |
27. Logical AND: &&
28. Logical OR: ||
29. Conditional Operator (Ternary): ? :
30. Assignment: =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=
31. Comma: ,

It's important to note that operator precedence affects how expressions are evaluated. When in

doubt, use parentheses to make the evaluation order explicit and avoid any ambiguity.

Some Examples :

1. Arithmetic Operators:

```
int a = 10, b = 5, c = 3;
int result1 = a + b * c; // 10 + (5 * 3) = 25
int result2 = (a + b) * c; // (10 + 5) * 3 = 45
```

2. Relational and Logical Operators:

```
int x = 8, y = 5;
int result3 = x > y && y != 0; // 1 (True)
int result4 = x <= y || y == 0; // 0 (False)
```

3. Bitwise Operators:

```
unsigned int num = 7;
unsigned int result5 = num << 2; // 28 (Binary: 111 << 2 = 11100)
unsigned int result6 = num | 8; // 15 (Binary: 111 | 1000 = 1111)
```

4. Ternary (Conditional) Operator:

```
int m = 5, n = 10;
int result7 = (m > n) ? m : n; // 10
```

5. Assignment Operators:

```
int p = 20;
p += 5; // p = p + 5 => 25
p *= 2; // p = p * 2 => 50
```

6. Unary Operators:

int q = 8;

int result8 = ++q; // Increment q and assign to result8 => 9

int result9 = q--; // Assign q to result9 and then decrement q => 9 (q is now 8)