What is C language and its feature

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is the most widely used computer language.

Key Features of the C Language:

- **Low-Level Memory Access:**

C allows for direct access to memory, which makes it ideal for system-level programming, like developing operating systems.
It supports the manipulation of memory addresses through pointers, providing fine control over hardware resources.

- **High Performance:**

C programs are known for their speed and efficiency because they can directly interact with hardware and memory.
This makes C a preferred choice for performance-critical applications such as operating systems, embedded systems, and other high-performance computing tasks.

- **Structured Programming:**

C promotes structured programming, enabling the division of complex tasks into smaller, manageable functions. This improves code clarity, organization, and reusability.

- **Portability:**

C code can be compiled and executed on different platforms with minimal or no changes, making it highly portable across various systems.

- **Comprehensive Standard Library:**

C includes a rich set of built-in functions that support common tasks like input/output operations, string manipulation, and memory management, simplifying many aspects of programming.

- **Versatility:**

Despite its low-level capabilities, C is also flexible enough for higher-level programming, enabling the development of complex applications such as compilers, databases, text editors, and network software.

Core Elements of C:

- **Data Types:** C offers a variety of data types, including int, float, char, and double, along with custom types like structures defined by the user.

- **Control Flow:** C includes essential control mechanisms such as loops (for, while, do-while) and conditional statements (if, else, switch) for decision-making and iteration.

- **Functions:** Functions in C help to organize code into reusable, modular segments, making complex programs easier to manage.

- **Pointers:** C allows the use of pointers, enabling direct memory manipulation and efficient management of arrays.

- **Memory Management:** C provides control over memory allocation through functions like malloc(), calloc(), and free(), allowing dynamic memory management.

Structure of C Programming

The fundamental structure of a C program is organized into 6 key sections, which help make the program easier to read, modify, document, and understand in a consistent format. For successful compilation and execution, a C program needs to adhere to this outlined framework. A well-structured C program also simplifies the debugging process.

Sections of the C Program

There are 6 basic sections responsible for the proper execution of a program. Sections are mentioned below:

- Documentation
- Preprocessor Section
- Definition
- Global Declaration
- Main() Function
- Sub Programs

- **Documentation:**

This section includes details about the program, such as its description, name, and the creation date and time. This information is provided at the beginning of the program as comments. Documentation is presented in this way:

Syntax: // description, name of the program, programmer name, date, time etc.
Anything written in comments serves as documentation for the program and does not affect the execution of the code. Essentially, it provides a summary for the reader about the program.

- **Preprocessor Section:**

In the preprocessor section, all the header files required by the program are declared. Header files allow us to include and use code from other sources in our own code. These files are copied into the program before the compilation process begins.

Example: #include<stdio.h>,#include<math.h>

- **Definition**

Preprocessors are tools that handle the source code before it is compiled. Several steps are involved in writing and running a program. Preprocessor directives begin with the # symbol. For example, the #define directive is used to create constants that are used throughout the program. When the compiler encounters this name, it is substituted with the defined value or code. Example: #define long long ll

- **Global Declaration**

The global declaration section contains global variables, function declaration, and static variables. Variables and functions which are declared in this scope can be used anywhere in the program.
Example: int num = 18;

- **Main() Function**

Every C program must have a main function. The main() function of the program is written in this section. Operations like declaration and execution are performed inside the curly braces of the main program. The return type of the main() function can be int as well as void too. void() main tells the compiler that the program will not return any value. The int main() tells the compiler that the program will return an integer value.
Example: void main() or int main()

- **Sub Programs**

User-defined functions are called in this section of the program. The control of the program is shifted to the called function whenever they are called from the main or outside the main() function. These are specified as per the requirements of the programmer.
Example:

```
int sum(int x, int y)
{
   return x+y;
}
```

Elements of C Programming

The key elements of C programming include the following components:

1. Keywords

Reserved words that have special meanings in C, such as int, return, if, while, etc.

2. Identifiers

Names given to various elements in the program like variables, functions, and arrays. Identifiers must begin with a letter or underscore and can be followed by letters, digits, or underscores.

3. Data Types

Defines the type of data that a variable can hold. Common data types include int (integers), float (floating-point numbers), char (characters), and double (double-precision floating-point numbers).

4. Variables

Storage locations identified by names used to hold data. Variables must be declared with a specific data type before they can be used.

5. Operators

Symbols used to perform operations on variables and values. Common operators include arithmetic operators (+, -, *, /), relational operators (<, >, ==), and logical operators (&&, ||, !).

6. Control Structures

Constructs that control the flow of execution in a program. These include conditionals (if, else, switch) and loops (for, while, do-while).

7. Functions

Blocks of code designed to perform a specific task. Functions can be called from other parts of the program and can return values. They help in modularizing the code and improving reusability.

8. Arrays

Collections of variables of the same type that are accessed using an index. Arrays can be one-dimensional or multidimensional.

9. Pointers

Variables that store memory addresses. Pointers are used for dynamic memory allocation and efficient array manipulation.

10. Structures

User-defined data types that group together different types of variables under a single name. Structures help in organizing complex data.

11. Unions

Similar to structures, but they allow storing different data types in the same memory location, though only one member can be used at a time.

12. Enumerations

Define a set of named integer constants. Enumerations improve code readability and maintainability.

13. Preprocessor Directives

Instructions for the preprocessor, which are executed before the actual compilation. Examples include #include, #define, and #ifdef.

14. Comments

Non-executable parts of the code used to annotate and explain the code. Comments are denoted by /* ... */ for multi-line comments and // for single-line comments.

15. Input/Output Functions

Functions used for handling input and output operations, such as printf() for output and scanf() for input.

Algorithm in C

An algorithm is a series of sequential steps or instructions created to accomplish a specific task or resolve a problem. In computer science, algorithms act as a structured plan for tackling problems in an organized and efficient manner.

Properties of an Algorithm

An algorithm must possess the following five properties −

- Input
- Output
- Finiteness
- Definiteness
- Effectiveness

Example:

Algorithm for finding the average of three numbers is as follows −

- Step 1- Start
- Step 2- Declare 3 numbers a,b,c
- Step 3- Compute sum = a+b+c
- Step 4- Compute average = sum/3
- Step 5- Print average value
- Step 6- Stop

Algorithm for sum two numbers is as follows -

- Step 1 - Get started
- Step 2 - Declare 3 integers a, b, c
- Step 3 - Define the values of a and b
- Step 4 - Add the values of a and b
- Step 5 - Save the output of step 4 in c
- Step 6 - Print c
- Step 7 - Stop

Flowchart in c Program

A flowchart is a commonly used graphical tool for representing algorithms and procedural workflows. It utilizes different symbols to illustrate the operations and decisions that occur within a program, progressing in a step-by-step sequence.

Rules for Drawing Flowchart

The various Rules or Guidelines for drawing the flowchart are given below.

- Only standard flowchart symbols should be utilized.

- Ensure that names and variables are used appropriately within the flowchart.
- For large or complex flowcharts, apply connector symbols to maintain clarity.
- Every flowchart should include clear start and stop points.

Flowchart symbols:

Different flowchart symbols carry distinct conventional meanings.

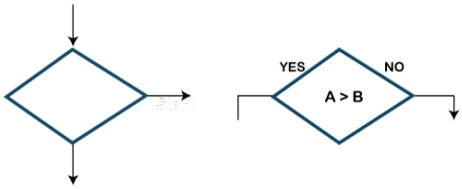
Here are the common symbols used in flowchart design:

- **Terminal Symbol:** Represented by an oval shape, it is used to indicate the start and stop points in a flowchart. The symbol shown below represents the terminal symbol.

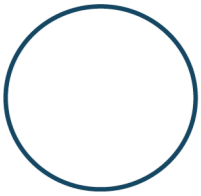
- **Input/Output Symbol:** This symbol is used to indicate data inputs and outputs in a flowchart. It represents actions like reading input or displaying output. The parallelogram shape shown below is used to depict the input/output symbol.

- **Processing Symbol:** In a flowchart, this symbol is represented by a rectangle. It is used to indicate arithmetic operations or data movement instructions. The rectangle symbol below represents the processing symbol.

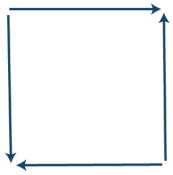
Decision Symbol: Represented by a diamond shape, this symbol is used for decision-making statements within a flowchart. The diamond symbol shown below indicates where decisions are made.



- **Connector Symbol:** This symbol is used to indicate where the flow of a process is interrupted and continues elsewhere. It helps maintain clarity in flowcharts by linking different parts of the flow. The symbol shown below represents the connector symbol.



- **Flow Lines:** These represent the specific sequence in which instructions are executed in a flowchart. Arrows are used to indicate the direction of flow between symbols. The arrow symbol shown below represents the flow lines.



- **Hexagon Symbol (Flat):** This symbol is used to create a preparation box that contains loop setting statements. It is represented by a flat hexagon shape. The symbol shown below represents the hexagon symbol used for this purpose.

- **On-Page Reference Symbol:** This symbol includes a letter inside it, indicating that the flow continues at another location on the same page where a matching symbol with the same letter is found. The symbol shown below represents the on-page reference symbol.

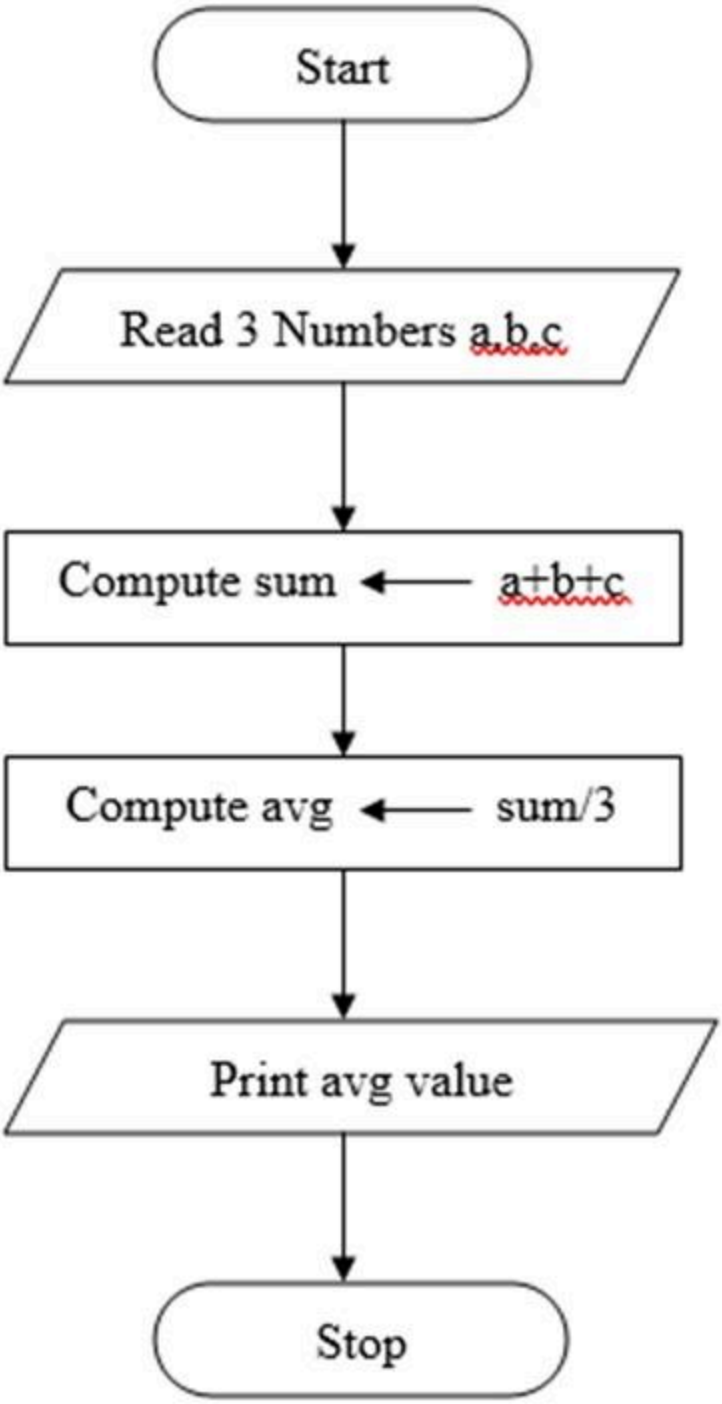
Off-Page Reference Symbol: This symbol includes a letter inside it to indicate that the flow continues on a different page where a matching symbol with the same letter is found. The symbol shown below represents the off-page reference symbol.

- **Delay or Bottleneck Symbol:** This symbol is used to indicate a delay or bottleneck in a flowchart. It shows where the process may be slowed down or interrupted. The symbol shown below represents the delay or bottleneck symbol.

- **Document Symbol:** This symbol is used in a flowchart to represent a document or report. The shape shown below denotes the document symbol.

- **Internal Storage Symbol:** This symbol is used to represent internal storage or memory within a flowchart. The shape shown below denotes the internal storage symbol.

Example:Given below is the flowchart for finding an average of three numbers −


Start
Read 3 Numbers a,b,c
Compute sum ← a+b+c
Compute avg ← sum/3
Print avg value
Stop

Tokens in C

Tokens in C are the fundamental elements necessary for writing a C program. A token can be defined as the smallest individual unit in a C program. Just as a sentence cannot be formed without words, a C program cannot be created without tokens. Thus, tokens in C serve as the basic building blocks or essential components for constructing any C program.

Types of Tokens in C

A C source code consists of various types of tokens. These tokens in C are categorized into the following types:

- Keywords
- Identifiers
- Constants
- Strings
- Special Symbols
- Operators

Keywords:

Keywords in C are predefined or reserved words that carry specific significance, with each one serving a unique function. Since these keywords are recognized and utilized by the compiler, they cannot be used as variable names. Using a keyword as a variable name would assign it a new meaning, which is not permitted. The C language includes 32 keywords, as listed below:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers in C

Identifiers in C are used to name variables, functions, arrays, structures, and more. These are user-defined names that can consist of uppercase letters, lowercase letters, underscores, or digits, but must begin with either an alphabet or an underscore. Identifiers cannot be the same as keywords.

Rules of Identifiers

The rules for creating identifiers in C are outlined below:

- They must begin with a letter or underscore(_).
- They must consist of only letters, digits, or underscore. No other special character is allowed.
- It should not be a keyword.
- It must not contain white space.
- It should be up to 31 characters long as only the first 31 characters are significant.

Variable in C

A variable represents a memory location used to store information. Its value can be changed and reused multiple times. Variables act as symbolic representations of memory locations, making them easy to identify.

In C programming, variables are fundamental components used to store and manipulate data. Each variable occupies a specific region of memory and holds a value of a defined data type. Every variable has a unique name (identifier) and a data type that specifies the kind of data it can contain.

Syntax: data_type variable_name;

Constant in C

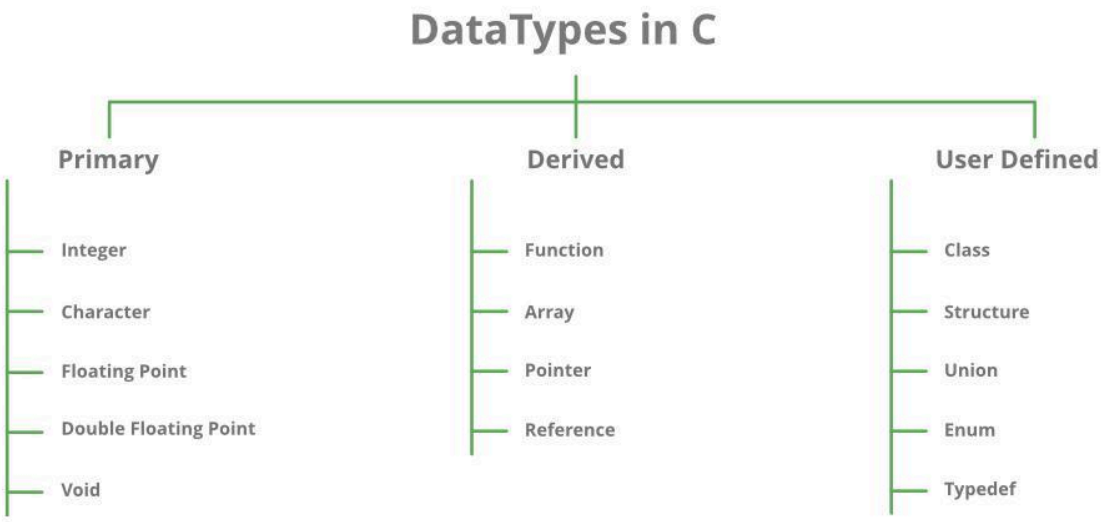
Constants in C are fixed values that cannot be altered during the execution of a program. They are used to represent values that should remain the same throughout the program, providing clarity and avoiding magic numbers in the code.

Syntax: const data_type var_name = value;

Data Types in C

In C, data types determine the kind of data that a variable can store. They define both the size and the nature of the data that a variable can hold. Knowing about data types is essential for efficient memory management and performing operations correctly in C programming.

Types of Data Types



Data Types	Memory Size	Format Specifier
char	1 byte	%c
signed Char	1 byte	%c
unsigned Char	1 byte	%c
short	2 byte	%hd
signed Short	2 byte	%hd
unsigned Short	2 byte	%hu
int	2 byte	%d
signed int	2 byte	%d
unsigned int	2 byte	%u
Short int	2 byte	%hd
signed short int	2 byte	%hd
unsigned short int	2 byte	%hu
long int	4 byte	%ld

signed long int	4 byte	%ld
unsigned long int	4 byte	%lu
float	4 byte	%f
double	8 byte	%lf
long double	10 byte	%Lf