# Amar Computer Point

## BCA Batch (2023-2026)

# Introduction to Computer Programming in C

## (Unit – 01)

# Introduction to Programming Languages:

A programming language is a set of instructions and syntax used to create software programs. A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**. It is a set of instructions written in any specific language (C, C++, Java, and Python) to perform a specific task.

## Types of Programming Language:

There are basically 5 major types of language.

1) Procedural programming languages.

2) Functional programming languages.

3) Object oriented programming languages.

4) Scripting languages.

5) Logic programming languages.

## Basic Terminologies in Programming Languages:

1) Algorithm

2) Variable

3) Data Type

4) Function

5) Control Flow

6) Syntax

7) Comment

8) Debugging

9) IDE

10) Operators

11) Statement

```
// C program for sum of 2 numbers
#include <stdio.h>

int main()
{
    int a, b, sum;
    a = 10;
    b = 15;
    sum = a + b;
    printf("Sum of %d and %d is: %d", a, b,
           sum); // perform addition operation
    return 0;
}
```

# The Basic Model of Computation:

A computational model is a computer implementation of a solution to a problem for which a mathematical representation is designed.

To solve a computer problem, computer tries to find a solution to a real- world problem and a computer program is the implementation of the solution to the problem.

## The Key feature of the basic computational models are –

### 1) Problem definition:

A problem definition involves the clear identification of the problem in terms of available input parameters and desired solution.

### 2) Approach towards solving the problem:

After a problem is identified, the user needs to implement a step-by-step solution in terms of algorithms.

### 3) Graphical representation of problem solving sequence:

This step involves representing the steps of algorithm pictorially by using a flowchart. Each component of the flowchart presents a definite process to solve the problem.

### 4) Converting the sequence in a programming language:

Converting the graphical sequence of processes into a language that the user and the computer can understand and use for problem solving is called programming.

After the program is compiled the user can obtain the desired solution for the problem by executing the machine language version of the program.

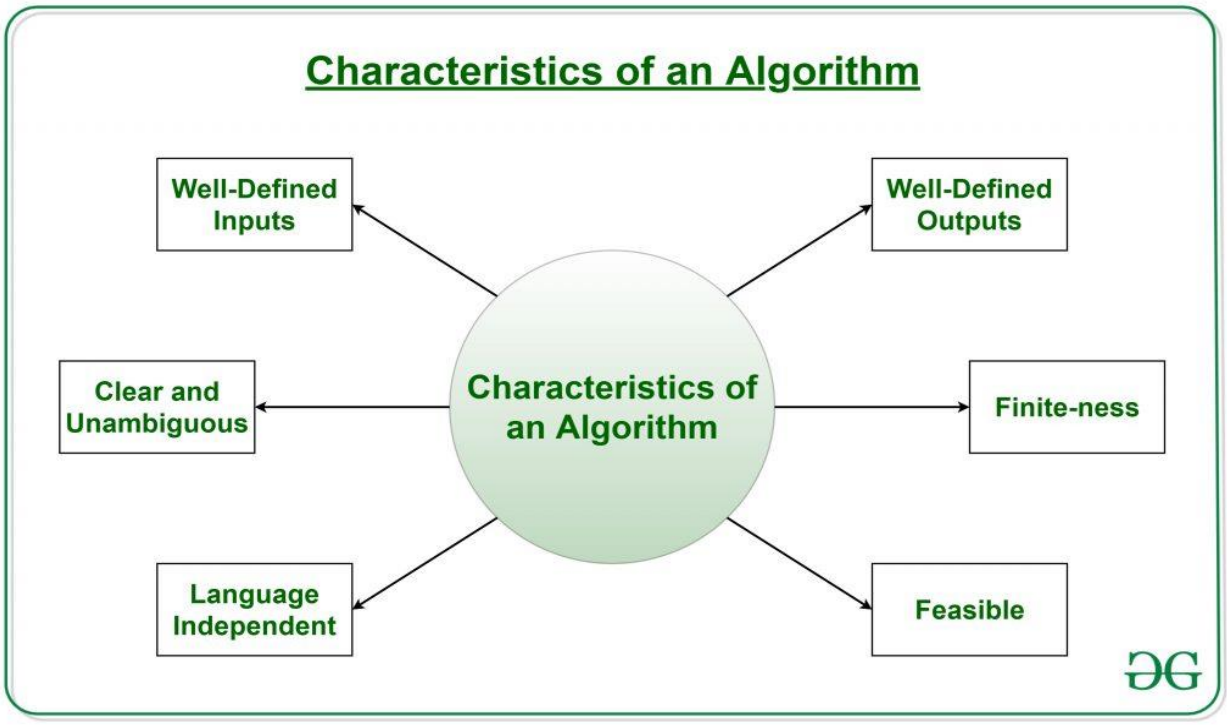
# Algorithm:

An algorithm is a step-by-step procedure to solve a problem. A good algorithm should be optimized in terms of time and space. Different types of problems require different types of algorithmic techniques to be solved in the most optimized manner.

A procedure for solving a mathematical problem in a finite number of steps that frequently involves recursive operations.

## What is the need for algorithms?

1. Algorithms are necessary for solving complex problems efficiently and effectively.
2. They help to automate processes and make them more reliable, faster, and easier to perform.
3. Algorithms also enable computers to perform tasks that would be difficult or impossible for humans to do manually.
4. They are used in various fields such as mathematics, computer science, engineering, finance, and many others to optimize processes, analyze data, make predictions, and provide solutions to problems.




## Properties of Algorithm:

- It should terminate after a finite time.
- It should produce at least one output.
- It should take zero or more input.
- It should be deterministic means giving the same output for the same input case.
- Every step in the algorithm must be effective i.e. every step should do some work.

## Types of Algorithms:

There are several types of algorithms available. Some important algorithms are:

1. Brute Force Algorithm
2. Recursive Algorithm
3. Backtracking Algorithm
4. Searching Algorithm
5. Sorting Algorithm
6. Hashing Algorithm
7. Divide and conquer Algorithm
8. Greedy Algorithm
9. Dynamic Programming Algorithm
10. Randomized Algorithm

## Advantages of Algorithms:

- It is easy to understand.
- An algorithm is a step-wise representation of a solution to a given problem.
- In an Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

## Disadvantages of Algorithms:

- Writing an algorithm takes a long time so it is time-consuming.
- Understanding complex logic through algorithms can be very difficult.
- Branching and looping statements are difficult to show in Algorithms **(imp)**.

## Example: Algorithm to add 3 numbers and print their sum:

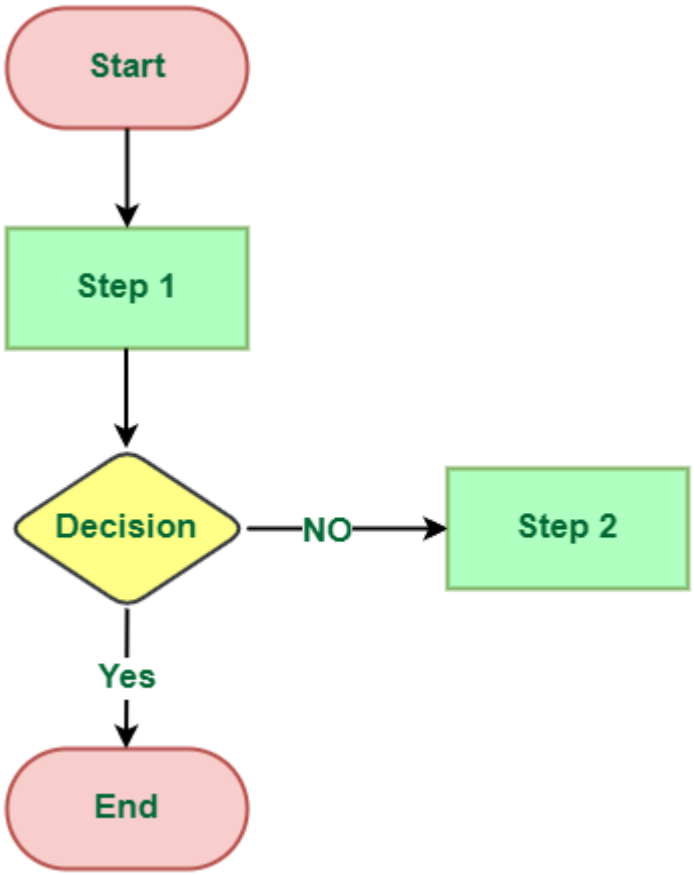
1. START
2. Declare 3 integer variables num1, num2, and num3.
3. Take the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
4. Declare an integer variable sum to store the resultant sum of the 3 numbers.
5. Add the 3 numbers and store the result in the variable sum.
6. Print the value of the variable sum

7. END

## FlowChart :

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

It makes use of symbols that are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flowcharting”. Example: Draw a flowchart to input two numbers from the user and display the largest of two numbers




### Uses of Flowcharts in Computer Programming/Algorithms

The following are the uses of a flowchart:

- It is a pictorial representation of an algorithm that increases the readability of the program.
- Complex programs can be drawn in a simple way using a flowchart.
- It helps team members get an insight into the process and use this knowledge to collect data, detect problems, develop software, etc.
- A flowchart is a basic step for designing a new process or adding extra features.
- Communication with other people becomes easy by drawing flowcharts and sharing them.

## Types of Flowcharts

Three types of flowcharts are listed below:

1. **Process flowchart:** This is the most common type of flowchart, used to represent a sequence of steps in a process. It shows the flow of tasks, decisions, and actions from start to finish.
2. **Data flowchart:** DFDs focus on the flow of data through a system, illustrating how data is input, processed, and output. They are useful for understanding data interactions and dependencies.
3. **Business Process Modeling Diagram:** Using this flowchart or diagram, one can analytically represent the business process and help simplify the concepts needed to understand business activities and the flow of information

## Advantages of Flowchart

- It is the most efficient way of communicating the logic of the system.
- It acts as a guide for a blueprint during the program design.
- It also helps in the debugging process.
- Using flowcharts we can easily analyze the programs.
- Flowcharts are good for documentation.

## Disadvantages of Flowchart

- Flowcharts are challenging to draw for large and complex programs.
- It does not contain the proper amount of details.
- Flowcharts are very difficult to reproduce.
- Flowcharts are very difficult to modify.

**Question 1. Draw a flowchart to find the greatest number among the 2 numbers.**
**Solution:**

***Algorithm:***

*1. Start*

*2. Input 2 variables from user*

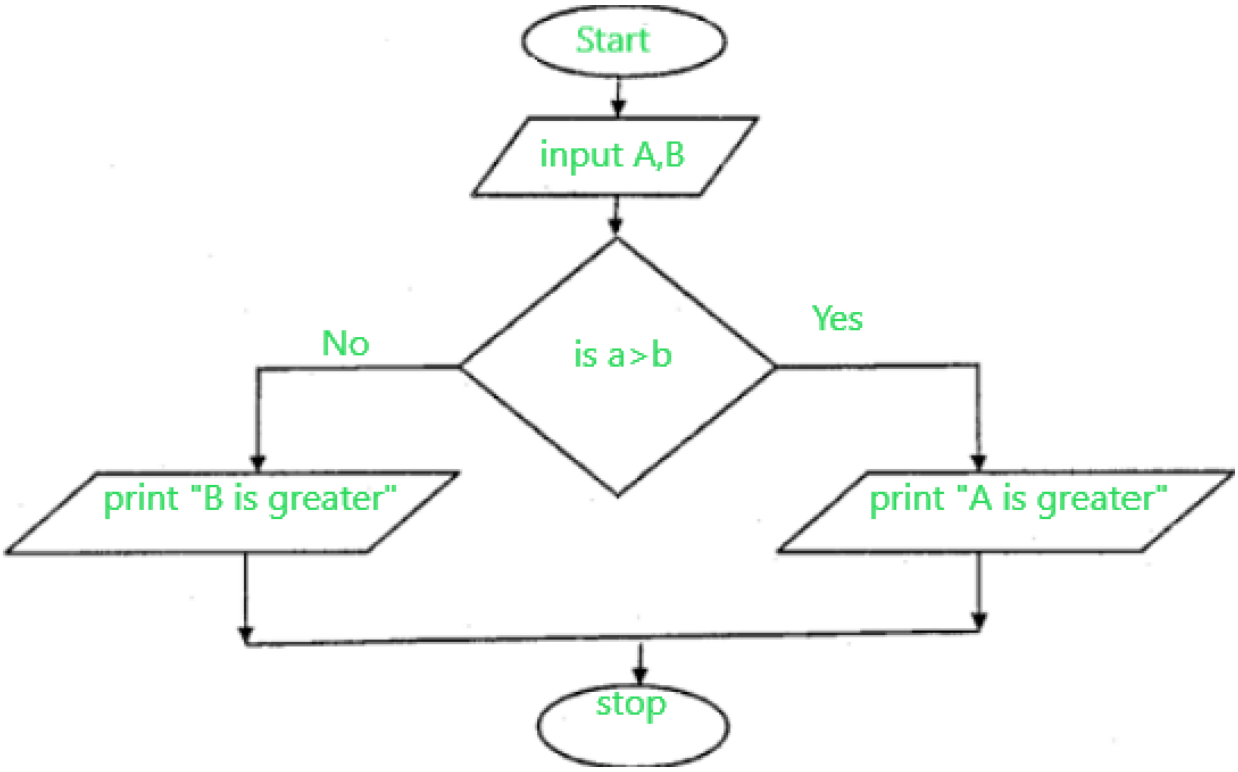
*3. Now check the condition If a > b, goto step 4, else goto step 5.*

*4. Print a is greater, goto step 6*

*5. Print b is greater*

*6. Stop*

**FlowChart:**



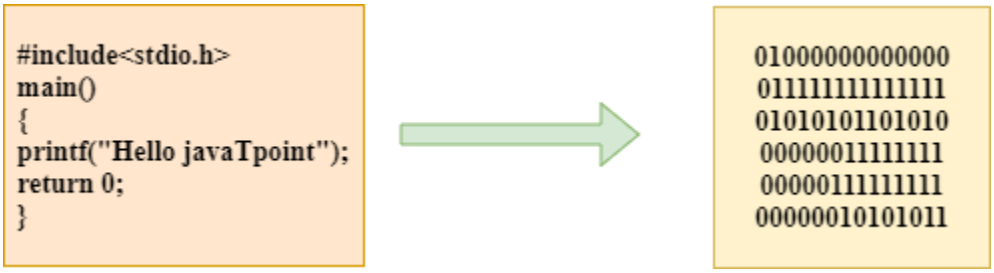

## Difference between algorithm and flow chart:

| S. No | Algorithm | Flowchart |
|---|---|---|

| | | |
|---|---|---|
| 1. | An algorithm is a step-by-step procedure to solve a problem. | A flowchart is a diagram created with different shapes to show the flow of data. |
| 2. | The algorithm is complex to understand. | A flowchart is easy to understand. |
| 3. | In the algorithm, plain text is used. | In the flowchart, symbols/shapes are used. |
| 4. | The algorithm is easy to debug. | A flowchart is hard to debug. |
| 5. | The algorithm is difficult to construct. | A flowchart is simple to construct. |
| 6. | The algorithm does not follow any rules. | The flowchart follows rules to be constructed. |
| 7. | The algorithm is the pseudo-code for the program. | A flowchart is just a graphical representation of that logic. |

## Compilation:

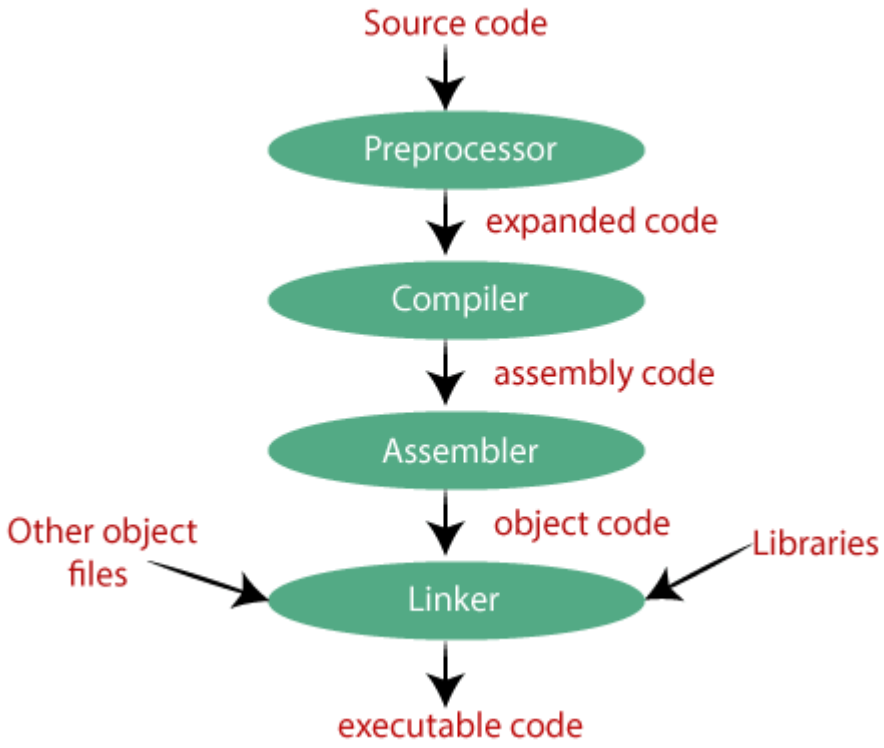
The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

The following are the phases through which our program passes before being transformed into an executable form:

- **Preprocessor**
- **Compiler**
- **Assembler**
- **Linker**




## Preprocessor

The source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

## Compiler

The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

## Assembler

The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. If the name of the source file is **'hello.c',** then the name of the object file would be 'hello.obj'.

## Linker

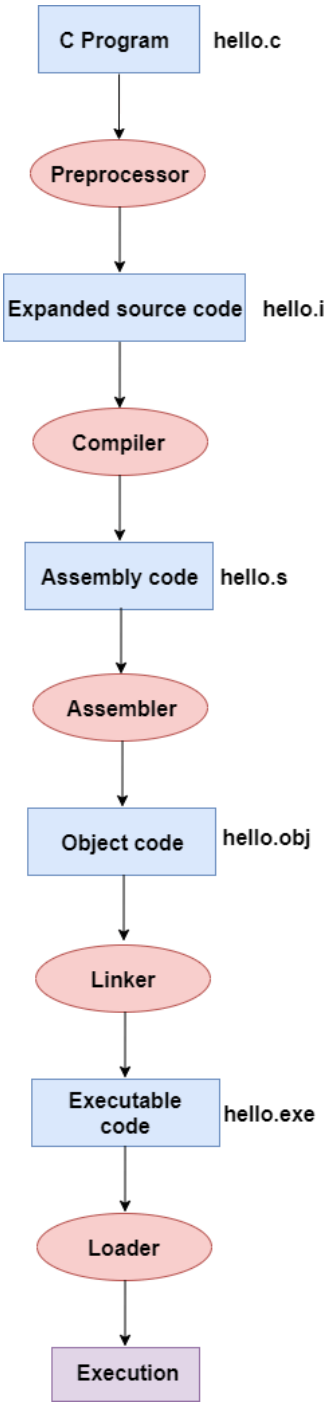
Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension. The main working of the linker is to combine the object code of library files with the object code of our program. Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this. It links the object code of these files to our program. if we are using printf() function in a program, then the linker adds its associated code in an output file.

**Let's understand through an example.**

**hello.c**

```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("Hello AmarComputerPoint");
5.     return 0;
6. }
```

**Now, we will create a flow diagram of the above program:**




# Linking and Loading

## Loading:

To bring the program from secondary memory to main memory is called Loading. It is performed by a loader. It is a special program that takes the input of

executable files from the linker, loads it to the main memory, and prepares this code for execution by a computer. There are two types of loading in the operating system:

- **Static Loading:**
    - Loading the entire program into the main memory before the start of the program execution is called static loading.
    - If static loading is used then accordingly static linking is applied.
- **Dynamic Loading:**
    - Loading the program into the main memory on demand is called dynamic loading.
    - If dynamic loading is used then accordingly dynamic linking is applied.

## Linking:

Linking is a process of collecting and maintaining pieces of code and data into a single file. Linker also links a particular module into the system library. It takes object modules from the assembler as input and forms an executable file as output for the loader. Linking is performed at the last step in compiling a program. There are two types of loading in the operating system:

- **Static Linking:**
    - A statically linked program takes constant load time every time it is loaded into the memory for execution.
    - Static linking is performed by programs called linkers as the last step in compiling a program. Linkers are also called link editors.
    - In static linking, if any of the external programs has changed then they have to be recompiled and re-linked again else the changes won’t reflect in the existing executable file.
- **Dynamic Linking:**
    - Dynamic linking is performed at run time by the operating system.
    - In dynamic linking, this is not the case and individual shared modules can be updated and recompiled. This is one of the greatest advantages dynamic linking offers.
    - In dynamic linking load time might be reduced if the shared library code is already present in memory.

# Testing and Debugging

## Testing:

Testing is the process of verifying and validating that a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

## Debugging:

Debugging is the process of fixing a bug in the software. It can be defined as identifying, analyzing, and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

## The main differences between testing and debugging are:

| Testing | Debugging |
|---|---|
| Testing is the process to find bugs and errors. | Debugging is the process of correcting the bugs found during testing. |
| It is the process to identify the failure of implemented code. | It is the process to give absolution to code failure. |
| Testing is the display of errors. | Debugging is a deductive process. |
| Testing is done by the tester. | Debugging is done by either programmer or the developer. |
| There is no need of design knowledge in the testing process. | Debugging can’t be done without proper design knowledge. |

| Testing | Debugging |
| --- | --- |
| Testing can be done by insiders as well as outsiders. | Debugging is done only by insiders. An outsider can’t do debugging. |
| Testing can be manual or automated. | Debugging is always manual. Debugging can’t be automated. |
| It is based on different testing levels i.e. unit testing, integration testing, system testing, etc. | Debugging is based on different types of bugs. |
| Testing is a stage of the software development life cycle (SDLC). | Debugging is not an aspect of the software development life cycle, it occurs as a consequence of testing. |
| Testing is composed of the validation and verification of software. | While debugging process seeks to match symptoms with cause, by that it leads to error correction. |
| Testing is initiated after the code is written. | Debugging commences with the execution of a test case. |
| Testing process based on various levels of testing-system testing, integration testing, unit testing, etc. | Debugging process based on various types of bugs is present in a system. |

## Character set:

In C programming language, a character set is defined as a set of characters that can be used to write the source code.

The character set refers to a set of all the valid characters that we can use in the source program for forming words, expressions, and numbers.

## Types of Characters in C

The C programming language provides support for the following types of characters. In other words, these are the *valid* characters that we can use in the C language:

- Digits
- Alphabets
- Special Characters

**Digits:**
The C programming language also supports digits for building numeric expressions.
Digits: 0-9 (10 characters).

**Alphabets:**
The C programming language allows for all upper- and lower-case alphabets in its programs.
Lower Case alphabets: a-z (26 characters).
Upper case alphabets: A-Z (26 characters).

**Special Characters:**
We use some special characters in the C language for some special purposes, such as logical operations, mathematical operations, checking of conditions, backspaces, white spaces, etc.
Special Characters: ` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ – | / \ ; : ‘ “ , . ?

# Variables

Variable is basically nothing but the name of a memory location that we use for storing data. We can change the value of a variable in C or any other language, and we can also reuse it multiple times. We use symbols in variables for

representing the memory location- so that it becomes easily identifiable by any user.

The name of a variable can be a composition of digits, letters, and also underscore characters. The name of the character must begin with either an underscore or a letter. In the case of C, the lowercase and uppercase letters are distinct. It is because C is case-sensitive in nature. Let us look at some more ways in which we name a variable.

## Rules for Naming a Variable in C

We give a variable a meaningful name when we create it. Here are the rules that we must follow when naming it:

1. The name of the variable must not begin with a digit.

2. A variable name can consist of digits, alphabets, and even special symbols such as an underscore ( _ ).

3. A variable name must not have any keywords, for instance, float, int, etc.

4. There must be no spaces or blanks in the variable name.

5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

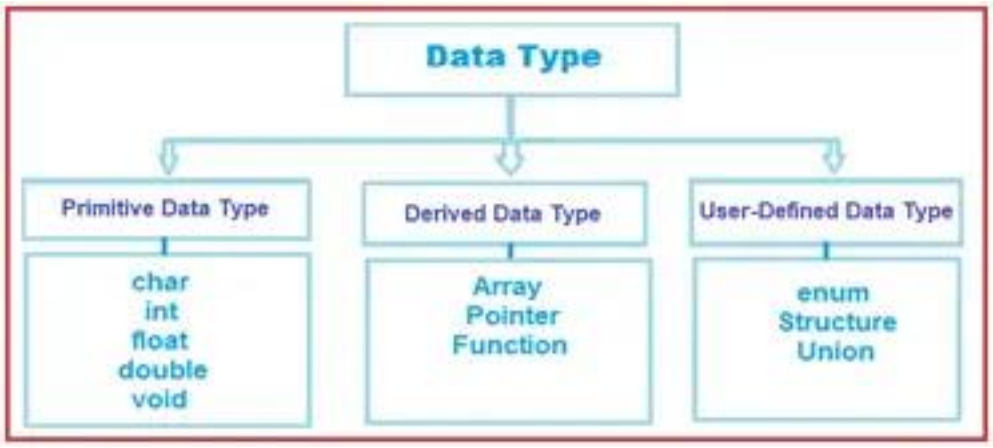
***Let us look at a few examples,***

```
// int type variable in C

int marks = 45;

// char type variable in C

char status = ‘G’;

// double type variable in C

double long = 28.338612;

// float type variable in C
```

float percentage = 82.5;

# DataTypes

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it. The data type is a collection of data with values having fixed values, meaning as well as its characteristics.




## 1) Primitive Data Type

Primitive data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.

## 2) User Defined Data Types

The user-defined data types are defined by the user himself.

## 3) Derived Types

The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.

**The following are some main primitive data types in C:**

**Integer Data Type**

The integer datatype in C is used to store the whole numbers without decimal values. Octal values, hexadecimal values, and decimal values can be stored in int data type in C.

- **Range:** -2,147,483,648 to 2,147,483,647
- **Size:** 4 bytes
- **Format Specifier:** %d

**Syntax –** int var_name

**Character Data Type**

Character data type allows its variable to store only a single character. The size of the character is 1 byte. It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

- **Range:** (-128 to 127) or (0 to 255)
- **Size:** 1 byte
- **Format Specifier:** %c

**Syntax** - char *var_name;*

## Float Data Type

In C programming float data type is used to store floating-point values. Float in C is used to store decimal and exponential values. It is used to store decimal numbers (numbers with floating point values) with single precision.

- **Range:** 1.2E-38 to 3.4E+38
- **Size:** 4 bytes
- **Format Specifier:** %f

**Syntax** - float *var_name;*

## Size of Data Types in C

The size of the data types in C is dependent on the size of the architecture, so we cannot define the universal size of the data types. For that, the C language provides the sizeof() operator to check the size of the data types.

Example

```
// C Program to print size of
// different data type in C
#include <stdio.h>

int main()
{
   int size_of_int = sizeof(int);
   int size_of_char = sizeof(char);
   int size_of_float = sizeof(float);
   int size_of_double = sizeof(double);

   printf("The size of int data type : %d\n", size_of_int);
   printf("The size of char data type : %d\n",
         size_of_char);
   printf("The size of float data type : %d\n",
         size_of_float);
   printf("The size of double data type : %d",
         size_of_double);

   return 0;
}
```

**Output**

The size of int data type : 4

The size of char data type : 1

The size of float data type : 4

The size of double data type : 8

# Operators:

C Operators are symbols that represent operations to be performed on one or more operands. C provides a wide range of operators, which can be classified into different categories based on their functionality. Operators are used for performing operations on variables and values.

Operators can be defined as the symbols that help us to perform specific mathematical, relational, bitwise, conditional, or logical computations on operands. In other words, we can say that an operator operates the operands. For example, '+' is an operator used for addition, as shown below:

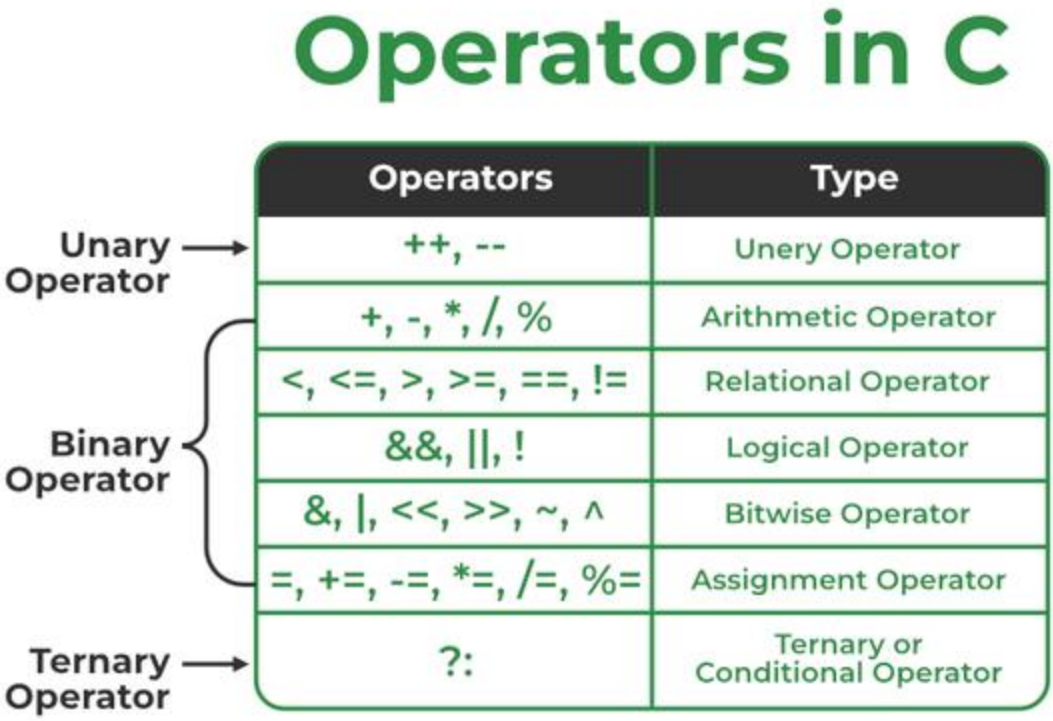
```
c = a + b;
```

Here, '+' is the operator known as the addition operator, and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'. The functionality of the C programming language is incomplete without the use of operators.

## Types of Operators in C

C has many built-in operators and can be classified into 6 types:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators




*Operators in C*

**The above operators have been discussed in detail**:

**1. Arithmetic Operations in C**

These operators are used to perform arithmetic/mathematical operations on operands. Examples: (+, -, *, /, %,++,–). Arithmetic operators are of two types:

**a) Unary Operators**:

Operators that operate or work with a single operand are unary operators. For example: Increment(++) and Decrement(–) Operators

```
int val = 5;

print("%d",++val);  // 6
```

**b) Binary Operators**:

Operators that operate or work with two operands are binary operators. For example: Addition(+), Subtraction(-), multiplication(*), Division(/) operators

```
int a = 7;
```

**int b = 2;**

**printf(“%d”,a+b); // 9**

## 2. Relational Operators in C

These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, whether an operand is greater than the other operand or not, etc. Some of the relational operators are (==, >= , <= ).

```
#include <stdio.h>

int main()

{

   printf("%d",(4 > 5));

   return 0;

}   //// 1
```

### 3. Logical Operator in C

Logical Operators are used to combining two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either **true** or **false**.

For example, the **logical AND** represented as the **‘&&’ operator in C** returns true when both the conditions under consideration are satisfied. Otherwise, it returns false. Therefore, a && b returns true when both a and b are true (i.e. non-zero)

```
printf("",(4 != 5) && (4 < 5)); // 1
```

### 4. Bitwise Operators in C

The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing. For example, the **bitwise AND** operator represented as **‘&’ in C** takes two numbers as operands and does

AND on every bit of two numbers. The result of AND is 1 only if both bits are 1(True).

**5. Assignment Operators in C**

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

```
#include <stdio.h>

int main()

{

   int a = 10;

   int b = 20;

   int sum = a+b;

   printf("The sum of %d and %d is %d",a,b,sum);

   return 0;

}
```