

### **Introduction to Algorithms and Programming Languages:**

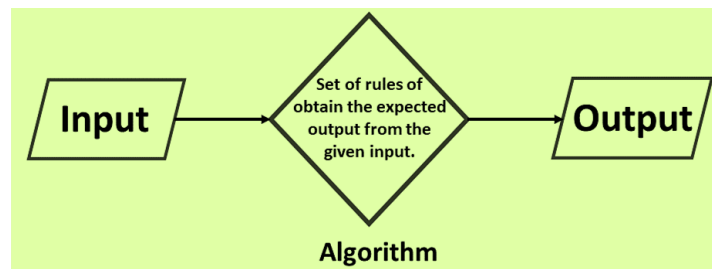
An algorithm is a popular term that come across in a variety of contexts, including computer programming, mathematics, and even our everyday life. In normal words, an algorithm describes a mathematical procedure or way to resolve an issue using a finite or definite number of steps.

According to the world of coding and computers, Algorithms are sets of instructions that assist us to resolve a problem within a definite or finite number of steps.

To better understand this definition by assuming it is like a recipe. A baker knows that there are many ways to make a cookie, but by following a recipe he knows to first preheat the oven. Next, he measures out the flour, and then adds chips, butter, etc. until the cookies are ready.

An algorithm is a step-by-step procedure that defines a set of instructions that must be carried out in a specific order to produce the desired result. It is not the entire program or code; it is simple logic to a problem represented as an informal description in the form of a flowchart or pseudocode.

Algorithm is a set of unambiguous instructions that must be followed for a computer to perform calculations or other problem-solving operations.



**Problem:** A problem can be defined as a real-world problem for which we need to develop a program or set of instructions.

**Algorithm:** An algorithm is defined as a step-by-step process that will be designed for a problem.

**Input:** After designing an algorithm, the algorithm is given the necessary and desired inputs.

**Processing unit:** The input will be passed to the processing unit, producing the desired output.

**Output:** The outcome or result of the program is referred to as the output.

***An algorithm must possess following characteristics:***

- ***Finiteness***: An algorithm should have finite number of steps and it should end after a finite time.
- ***Input***: An algorithm may have many inputs or no inputs at all.
- ***Output***: It should result at least one output.
- ***Definiteness***: Each step must be clear, well-defined and precise. There should be no any ambiguity.
- ***Effectiveness***: Each step must be simple and should take a finite amount of time.

***Guidelines for Developing an Algorithm:***

Following guidelines must be followed while developing an algorithm:

1. An algorithm will be enclosed by START (or BEGIN) and STOP (or END).
2. To accept data from user, generally used statements are INPUT, READ, GET or OBTAIN.
3. To display result or any message, generally used statements are PRINT, DISPLAY, or WRITE.
4. Generally, COMPUTE or CALCULATE is used while describing mathematical expressions and based on situation relevant operators can be used.

**Example of an Algorithm**

***Algorithm : Calculation of Simple Interest***

Step 1: Start

Step 2: Read principle (P), time (T) and rate (R)

Step 3: Calculate  $I = P \cdot T \cdot R / 100$

Step 4: Print I as Interest

Step 5: Stop

### **Advantages of an Algorithm**

*Designing an algorithm has following advantages:*

**Effective Communication:** Since algorithm is written in English like language, it is simple to understand step-by-step solution of the problems.

**Easy Debugging:** Well-designed algorithm makes debugging easy so that we can identify logical error in the program.

**Easy and Efficient Coding:** An algorithm acts as a blueprint of a program and helps during program development.

**Independent of Programming Language:** An algorithm is independent of programming languages and can be easily coded using any high level language.

### **Disadvantages of an Algorithm**

*An algorithm has following disadvantages:*

1. Developing algorithm for complex problems would be time consuming and difficult to understand.
2. Understanding complex logic through algorithms can be very difficult.

### **Flowcharts:**

Flowcharts are generally used in process industries to indicate the flow of the product during various stages of the process. This is a combination of an outline process chart and the flow diagram, where each operation is represented by the appropriate shape of the equipment. This gives a visual picture of the equipment as well as the operation sequence. Sometimes the equipment is represented by functional blocks, as illustrated therein. Even in computer programming, flowcharts are used to represent the series of decisions and the corresponding action taken.

A flowchart can be useful for illustrating the overall algorithm (process) graphically, particularly when learning programming.

A flowchart is a graphical representation of steps. It is a tool for representing algorithms and programming logic but had extended to use in all other kinds of processes. Nowadays, flowcharts play an extremely important role in displaying information and assisting reasoning. They help us

# RENAISSANCE UNIVERSITY, INDORE









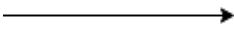
BCA I SEM

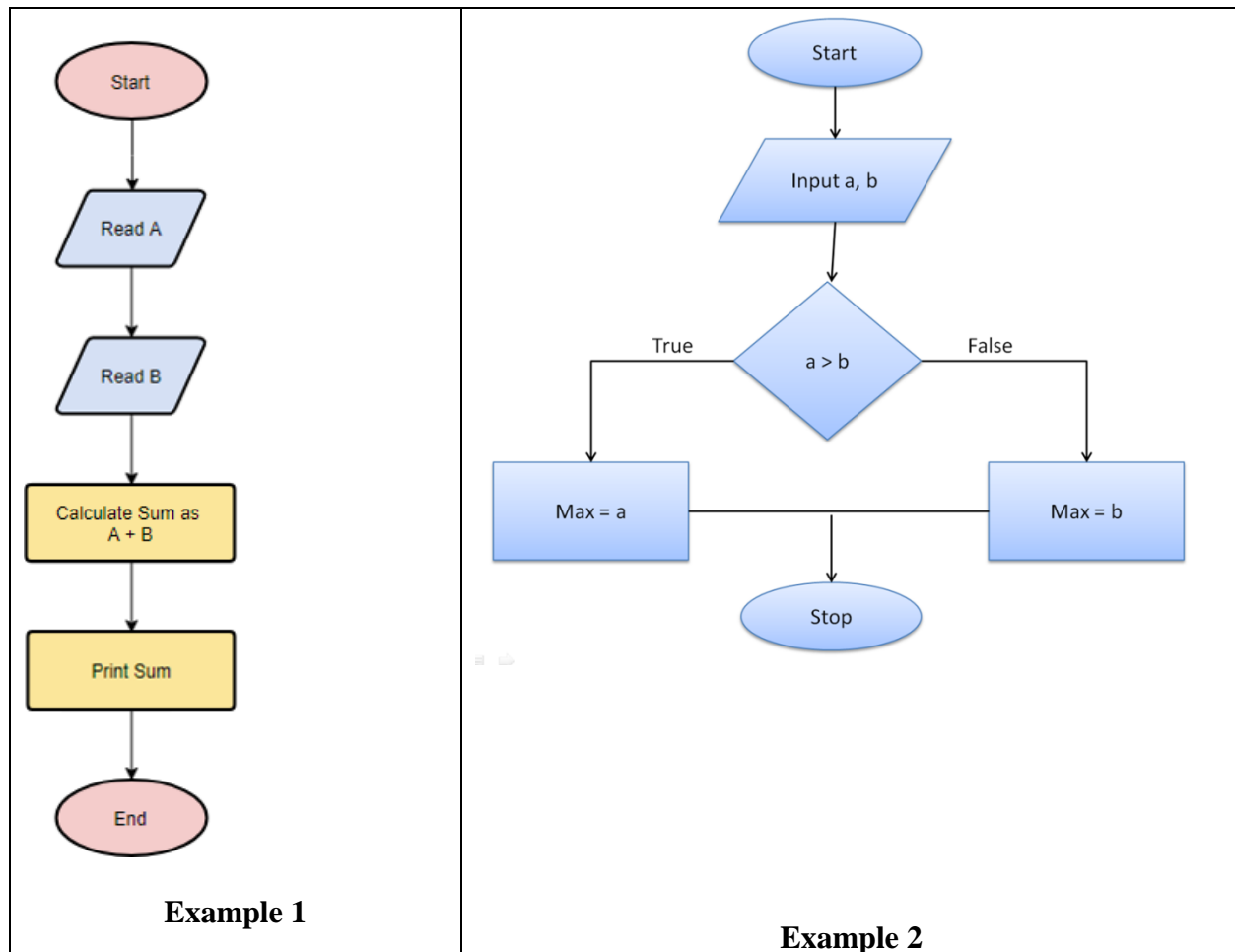
Subject: Programming and Problem Solving Using C

visualize complex processes, or make explicit the structure of problems and tasks. A flowchart can also be used to define a process or project to be implemented.

## Flowchart Symbols

Different flowchart shapes have different conventional meanings. The meanings of some of the more common shapes are as follows:

<b>Terminator</b>		The terminator symbol represents the starting or ending point of the system.
<b>Process</b>		A box indicates some particular operation.
<b>Document</b>		This represents a printout, such as a document or a report.
<b>Decision</b>		A diamond represents a decision or branching point. Lines coming out from the diamond indicates different possible situations, leading to different sub-processes.
<b>Data</b>		It represents information entering or leaving the system. An input might be an order from a customer. Output can be a product to be delivered.
<b>On-Page Reference</b>		This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on the same page.
<b>Off-Page Reference</b>		This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on a different page.
<b>Delay or Bottleneck</b>		Identifies a delay or a bottleneck.
<b>Flow</b>		Lines represent the flow of the sequence and direction of a process.



## Pseudocode:

Pseudocode literally means '*fake code*'. It is an informal and contrived way of writing programs in which you represent the sequence of actions and instructions (aka algorithms) in a form that humans can easily understand.

You see, computers and human beings are quite different, and therein lay the problem.

The language of a computer is very rigid: you are not allowed to make any mistakes or deviate from the rules. Even with the invention of high-level, human-readable languages like Java and Python, it's still pretty hard for an average human developer to program in those coding languages.

***Example 1: Calculate the Average of Three Numbers***

START

INPUT five numbers and store them in variables num1, num2 and num3

CALCULATE the sum of the three inputted numbers and store them in variable sum

CALCULATE the average of the three inputted numbers and store them in variable avg

PRINT the value of the variable avg

END

**Example 2 Compute Area of a Triangle**

START

Enter the base , B

Enter height, H

Calculate the area =  $1/2 * B * H$

Display area

END

**Advantages of Pseudocode**

There are several advantages inherent to pseudocode and its use. Some of these include:

- Pseudocode is a simple way to represent an algorithm or program.
- It is written easily in a word processing application and easily modified.
- Pseudocode is easy to understand and can be written by anyone.
- Pseudocode can be used with various structured programming languages.

**Disadvantages of Pseudocode**

The pseudocode also has some disadvantages that should be considered. Some of these disadvantages include:

- Pseudocode is not a programming language, so it cannot be executed by a computer.
- Pseudocode can be ambiguous and sometimes open to interpretation.
- Pseudocode can be difficult to read if not written in a clear and consistent manner.

### **Programming Language:**

As we know, to communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language is called Programming language.

Language is a mode of communication that is used to share ideas, opinions with each other. For example, if we want to teach someone, we need a language that is understandable by both communicators.

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, Python) to perform a specific task.

### **Types of programming language-**

#### **1. Low-level programming language**

Low-level language is machine-dependent (0s and 1s) programming language. The processor runs low-level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast.

Low-level language is further divided into two parts -

##### ***i. Machine Language***

Machine language is a type of low-level programming language. It is also called as machine code or object code. Machine language is easier to read because it is normally displayed in binary or hexadecimal form (base 16) form. It does not require a translator to convert the programs because computers directly understand the machine language programs.

The advantage of machine language is that it helps the programmer to execute the programs faster than the high-level programming language.

##### ***ii. Assembly Language***

Assembly language (ASM) is also a type of low-level programming language that is designed for specific processors. It represents the set of instructions in a symbolic and human-understandable form. It uses an assembler to convert the assembly language to machine language.

The advantage of assembly language is that it requires less memory and less execution time to execute a program.

## **2. High-level programming language**

High-level programming language (HLL) is designed for developing user-friendly software programs and websites. This programming language requires a compiler or interpreter to translate the program into machine language (execute the program).

The main advantage of a high-level language is that it is easy to read, write, and maintain.

High-level programming language includes Python, Java, JavaScript, PHP, C#, C++, C etc.

### **Generations of programming language:**

Programming languages have been developed over the year in a phased manner. Each phase of developed has made the programming language more user-friendly, easier to use and more powerful. Each phase of improved made in the development of the programming languages can be referred to as a generation. The programming language in terms of their performance reliability and robustness can be grouped into five **different generations**,

1. First generation languages (1GL)
2. Second generation languages (2GL)
3. Third generation languages (3GL)
4. Fourth generation languages (4GL)
5. Fifth generation languages (5GL)

#### ***1. First Generation Language (Machine language)***

The first generation programming language is also called low-level programming language because they were used to program the computer system at a very low level of abstraction. i.e. at the machine level. The machine language also referred to as the native language of the computer system is the first generation programming language. In the machine language, a programmer only deals with a binary number.

#### **Advantages of first generation language**

- They are translation free and can be directly executed by the computers.
- The programs written in these languages are executed very speedily and efficiently by the CPU of the computer system.
- The programs written in these languages utilize the memory in an efficient manner because it is possible to keep track of each bit of data.



## ***2. Second Generation language (Assembly Language)***

The second generation programming language also belongs to the category of low-level programming language. The second generation language comprises assembly languages that use the concept of mnemonics for the writing program. In the assembly language, symbolic names are used to represent the opcode and the operand part of the instruction.

### **Advantages of second generation language**

- It is easy to develop understand and modify the program developed in these languages are compared to those developed in the first generation programming language.
- The programs written in these languages are less prone to errors and therefore can be maintained with a great ease.

## ***3. Third Generation languages (High-Level Languages)***

The third generation programming languages were designed to overcome the various limitations of the first and second generation programming languages. The languages of the third and later generation are considered as a high-level language because they enable the programmer to concentrate only on the logic of the programs without considering the internal architecture of the computer system.

### **Advantages of third generation programming language**

- It is easy to develop, learn and understand the program.
- As the program written in these languages are less prone to errors they are easy to maintain.
- The program written in these languages can be developed in very less time as compared to the first and second generation language.

**Examples:** FORTRAN, ALGOL, COBOL, C++, C

## ***4. Fourth generation language (Very High-level Languages)***

The languages of this generation were considered as very high-level programming languages required a lot of time and effort that affected the productivity of a programmer. The fourth generation programming languages were designed and developed to reduce the time, cost and effort needed to develop different types of software applications.

### **Advantages of fourth generation languages**

- These programming languages allow the efficient use of data by implementing the various database.
- They require less time, cost and effort to develop different types of software applications.

- The programs developed in these languages are highly portable as compared to the programs developed in the languages of other generation.

**Examples:** SOL, CSS, coldfusion

### ***5. Fifth generation language (Artificial Intelligence Language)***

The programming languages of this generation mainly focus on constraint programming. The major fields in which the fifth generation programming language are employed are Artificial Intelligence and Artificial Neural Networks

#### **Advantages of fifth generation languages**

- These languages can be used to query the database in a fast and efficient manner.
- In this generation of language, the user can communicate with the computer system in a simple and an easy manner.

**Examples:** mercury, prolog, OPS5

### **Structured Programming Language:**

In structured programming, we sub-divide the whole program into small modules so that the program becomes easy to understand. The purpose of structured programming is to linearize control flow through a computer program so that the execution sequence follows the sequence in which the code is written. The dynamic structure of the program than resemble the static structure of the program. This enhances the readability, testability, and modifiability of the program. This linear flow of control can be managed by restricting the set of allowed applications construct to a single entry, single exit formats.

We use structured programming because it allows the programmer to understand the program easily. If a program consists of thousands of instructions and an error occurs then it is complicated to find that error in the whole program, but in structured programming, we can easily detect the error and then go to that location and correct it. This saves a lot of time.

## Introduction to C:

C is a procedural programming language. It was initially developed by *Dennis Ritchie* in the year 1972. It was mainly developed as a system programming language to write an operating system. The main features of the C language include low-level memory access, a simple set of keywords, and a clean style, these features make C language suitable for system programming like an operating system or compiler development.

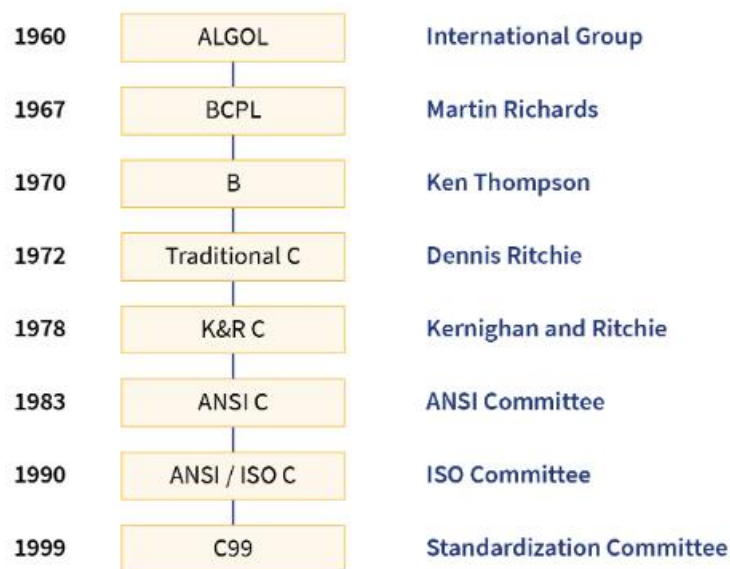
Many later languages have borrowed syntax/features directly or indirectly from the C language. Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on the C language. C++ is nearly a superset of C language.

C is a general-purpose programming language that is extremely popular, simple, and flexible to use. It is a structured programming language that is machine.

## History of C Language:

History of C language is interesting to know. Here we are going to discuss a brief history of the c language. C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A. Dennis Ritchie is known as the founder of the c language. It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in UNIX operating system. It inherits many features of previous languages such as B and BCPL.



***Features of C Language:***

C is the widely used language. It provides many features that are given below.

***1) Simple***

C is a simple language in the sense that it provides a structured approach (to break the problem into parts), the rich set of library functions, data types, etc.

***2) Machine Independent or Portable***

Unlike assembly language, c programs can be executed on different machines with some machine specific changes. Therefore, C is a machine independent language.

***3) Mid-level programming language***

Although, C is intended to do low-level programming. It is used to develop system applications such as kernel, driver, etc. It also supports the features of a high-level language. That is why it is known as mid-level language.

***4) Structured programming language***

C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify. Functions also provide code reusability.

***5) Rich Library***

C provides a lot of inbuilt functions that make the development fast.

***6) Dynamic Memory Management***

It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free() function.

***7) Speed***

The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

***8) Pointer***

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array, etc.

***9) Recursion***

In C, we can call the function within the function. It provides code reusability for every function. Recursion enables us to use the approach of backtracking.

***10) Syntax Based Language***

Like most High-level languages, for example, Java, C++, C#, the C language has a syntax, there are proper rules for writing the code, and the C language strictly follows it. If you write anything that is not allowed, you will get a compile-time error, which happens when the compiler is unable to compile your code because of some incorrect code syntax.

***11) Powerful***

C language is a very powerful programming language. It has a broad range of features like support for many data types, operators, keywords, etc., allows structuring of code using functions, loops, decision-making statements, then there are complex data-structures like structures, arrays, etc., and pointers, which makes C quite resourceful and powerful, etc.

***10) Extensible***

C language is extensible because it can easily adopt new features.

# RENAISSANCE UNIVERSITY, INDORE

BCA I SEM

Subject: Programming and Problem Solving Using C

## Structure of the C Program:

Section	Description
<b>Documentation</b>	Consists of the description of the program, programmer's name, and creation date. These are generally written in the form of comments.
<b>Link</b>	All header files are included in this section which contains different functions from the libraries. A copy of these header files is inserted into your code before compilation.
<b>Definition</b>	Includes preprocessor directive, which contains symbolic constants. E.g.: #define allows us to use constants in our code. It replaces all the constants with its value in the code.
<b>Global Declaration</b>	Includes declaration of global variables, function declarations, static global variables, and functions.
<b>Main() Function</b>	For every C program, the execution starts from the main() function. It is mandatory to include a main() function in every C program.
<b>Subprograms</b>	Includes all user-defined functions (functions the user provides). They can contain the inbuilt functions and the function definitions declared in the Global Declaration section. These are called in the main() function.

## Example:

```
1. /* Sum of two numbers */
2. #include<stdio.h>
3. int main()
4. {
5.     int a, b, sum;
6.     printf("Enter two numbers to be added ");
7.     scanf("%d %d", &a, &b);
8.     // calculating sum of two numbers
9.     sum = a + b;
10.    printf("%d + %d = %d", a, b, sum);
11.    return 0;
12. }
```

## **C Character Set:**

As every language contains a set of characters used to construct words, statements, etc., C language also has a set of characters which include **alphabets**, **digits**, and **special symbols**. C language supports a total of 256 characters.

Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C language character set contains the following set of characters...

1. Alphabets
2. Digits
3. Special Symbols

### ***Alphabets***

C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.

lower case letters - **a to z**

UPPER CASE LETTERS - **A to Z**

### ***Digits***

C language supports 10 digits which are used to construct numerical values in C language.

Digits - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

### ***Special Symbols***

C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.

Special Symbols - **~ @ # \$ % ^ & \* ( ) \_ - + = { } [ ] ; : ' " / ? . > , < etc.**

## **Comments in C:**

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments
2. Multi-Line Comments

### ***Single Line Comments-***

Single line comments are represented by double slash \\. Let's see an example of a single line comment in C.

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

### ***Multi Line Comments-***

Multi-Line comments are represented by slash asterisk \\* ... \*. It can occupy many lines of code, but it can't be nested. Syntax:

```
1. #include<stdio.h>
2. int main(){
3. /*printing some information
4. Multi-Line Comment*/
5. printf("Hello C");
6. return 0;
7. }
```

### **Tokens in C:**

Tokens in C are the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For example, you cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, tokens in C is the building block or the basic component for creating a program in C language.



### *Classification of tokens in C*

Tokens in C language can be divided into the following categories:

- Keywords
- Identifiers
- Strings
- Operators
- Constant
- Special Characters

**1. Keywords:** Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. You cannot redefine keywords. However, you can specify the text to be substituted for keywords before compilation by using C/C++ preprocessor directives. C language supports 32 keywords which are given 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

**2. Identifiers:** Identifiers are used as the general terminology for the naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(\_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value.

***There are certain rules that should be followed while naming c identifiers:***

- 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.

**3. Strings:** Strings are nothing but an array of characters ended with a null character ('\0'). This null character indicates the end of the string. Strings are always enclosed in double-quotes. Whereas, a character is enclosed in single quotes in C. Declarations for String:

```
char string[20] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', '\0'};
```

```
char string[20] = "programming";
```

```
char string [] = "programming";
```

when we declare char as "string[20]", 20 bytes of memory space is allocated for holding the string value.

When we declare char as "string[]", memory space will be allocated as per the requirement during the execution of the program.

**4. Operators:** Operators are symbols that trigger an action when applied to C variables and other objects. The data items on which operators act upon are called operands.

Depending on the number of operands that an operator can act upon, operators can be classified as follows:

- **Unary Operators:** Those operators that require only a single operand to act upon are known as unary operators. For Example increment and decrement operators
- **Binary Operators:** Those operators that require two operands to act upon are called binary operators. Binary operators are classified into :

Arithmetic operators

Relational Operators

Logical Operators

Assignment Operators

Bitwise Operator

- **Ternary Operator:** The operator that require three operands to act upon are called ternary operator. Conditional Operator(?) is also called ternary operator.

Syntax: (Expression1)? expression2: expression3;

**5. Constants:** A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

There are two ways of declaring constant:

Using const keyword

Using #define pre-processor

**6. Special characters:** Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (\*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.