# CHAPTER ONE

# Introduction to Programming

## **The evolution of programming languages**

**What is programming language?**

Programming language is a systematic notation by which we describe computational processes to others. Computational process is a set of steps, which a machine can perform for solving a problem.

**There are three types of programming languages.**

* 1. Machine language
  2. Assembly language
  3. High level language

1. **Machine language**

Machine language is the only language the computers can understand directly. It is machine dependent, i.e. means a particular machine language can be used on only one type of computer. It uses binary numbers/digits i.e. a combination of 0’s and 1’s.

*Example*, Machine language code might look like this:

000000101111001010

000000101111110010

000000110011101010

1. **Assembly language**

Assembly language is a programming language, which uses mnemonics/symbols to write a program. It is machine dependent-refer the meaning from the sentence under machine language. **Example**, Assembly language program that adds overtime pay to base pay and stores the result in gross pay is:

**LOAD** BASEPAY

**ADD** OVERPAY

**STORE** GROSSPAY

1. **High level language**

High-level language is not machining dependent – this means it can be used on different machine with little modifications. It easier to learn and understand because it resembles/look like English. Require less time to write and easier to debug errors. Some examples of high level programming languages are C, C++, FORTRAN, COBOL, and Algol etc. **Example**, A payroll program written in high-level language might contain a statement such as **grossPay** = **basePay** + **overTimePay**

## **Language Translators**

Programs that translate a program written in high-level language or assembly language to machine code program are called **translators**. A program written in high level language or assembly language is called **source code** or **source program** and the translated machine code is called **object code** or **object program**.

**There are three types of language translators. These** **are**:

* 1. Assembler
  2. Interpreter, and
  3. Compiler

1. **Assembler**

It is a software tool for translating low-level language (assembly language) into machine code.

1. **Interpreter**

An interpreter translates a program as it reads it, turning the program instructions or code directly into actions. Because interpreters read, the code as it is written and executes the code on the spot and interpreters are easy for the programmer to work with.

1. **Compiler**

A compiler translates the code into an intermediary form. This step is called compiling and produces an object file. The compiler then invokes a linker, which turns the object file into an executable program. Compilers produce a program that is very fast each time it is run. However, the time-consuming task of translating the source code into machine language has already been accomplished.

## **Algorithm, Pseudo code and flow chart**

1. **Algorithm**

The sequence of steps to be performed in order to solve a problem by the computer is known as an **algorithm**. In mathematics, computer science, and related subjects, an algorithm is a finite sequence of steps expressed for solving a problem. An algorithm can be defined as a process that performs some sequence of operations in order to solve a given problem.

**Hence, in order to qualify as an algorithm, a sequence of instructions must possess the following characteristics:**

* 1. Every instruction should be precise and unambiguous
  2. Each instruction should be such that it can be performed in a finite time
  3. One or more instructions should not be repeated infinitely.
  4. After the algorithm terminates, the desired results must be obtained.

**Quality of Algorithm**

The primary factors that are often used to judge the quality of an algorithm are:

1. **Time requirement**:
   * This is the time required to execute the corresponding program on a given computer system.
2. **Memory requirement**:
   * This is the memory space required to execute the corresponding program on a given computer system.
3. **Accuracy of solution**:
   * Although multiple algorithms may provide correct solutions to a given problem, some of these may provide more accurate results than others. Therefore, we have to choose the better algorithm from the point of view of accuracy of solution.
4. **Generality**:
   * A generalized algorithm which can handle a range of input data is better than one which has been designed to solve a problem for a single input data.
5. **Pseudo code**

Pseudo code is one of the tools that can be used to write a preliminary plan that can be developed into a computer program. Pseudo code is a generic way of describing an algorithm without use of any specific programming language syntax. It is, as the name suggests, pseudo code - it cannot be executed on a real computer, but it models and resembles real programming code, and is written at roughly the same level of detail. Pseudo code, by nature, exists in various forms although most borrow syntax from popular programming languages (like C, Lisp, or FORTRAN). Natural language is used whenever details are unimportant or distracting.

1. **Flowcharts**

A Flowchart is a type of diagram (graphical or symbolic) that represents an algorithm or process. Each step in the process is represented by a different symbol and contains a short description of the process step. A flowchart describes what operations (and in what sequence) are required to solve a given problem. A flowchart can be likened to the blueprint of a building. **Generally**, Flowcharts are a pictorial or graphical representation of a process.

**Flowchart Symbols & Guidelines:**

Some standard symbols, which are frequently required for flowcharting many computer programs are shown.

1. **Terminator**: An oval flow chart shape indicates the start or end of the process, usually containing the word “Start” or “End”.

**Terminator**

1. **Process:** A rectangular flow chart shape indicates a normal/generic process flow step. For example, “Add 1 to X”, “M = M\*F” or similar.

Connector

**Process**

1. **Decision**: This symbol is used when a decision needs to be made, commonly a Yes/No question or True/False test.

**Decision**

1. **Connector**: A small, labeled, circular flow chart shape used to indicate a jump in the process flow. Connectors are generally used in complex or multi-sheet diagram.

**Connector**

1. **Input/output:** A parallelogram that indicates data input or output (I/O) for a process. Examples: Get X from the user, Display X.

**Input/Out put**

1. **Arrow**: used to show the flow of control in a process. An arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to.

**Advantages of Using Flowcharts**

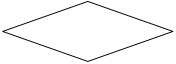
* **Communication:** flowcharts are better way of communicating the logic of a system to all concerned.
* **Effective analysis:** with the help of flowchart, problem can be analyzed in way that is more effective.
* **Proper documentation:** program flowcharts serve as a good program documentation, which is needed for various purposes.
* **Efficient coding:** the flowcharts act as a guide or blueprint during the systems analysis and program development phase.
* **Proper Debugging:** The flowchart helps in debugging process.
* **Efficient Program Maintenance:** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

**Limitations of Using Flowcharts**

Although a flowchart is a very useful tool, there are a few limitations in using flowcharts which are listed below:

* **Complex logic:** Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
* **Alterations and Modifications:** If alterations are required the flowchart may require re-drawing completely
* **Reproduction:** As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.

**Multiple Choice Questions and Answer on Chapter One**

1. \_\_\_\_\_\_\_ is a procedure or step by step process for solving a problem.
2. **Algorithm** B. Flowchart C. Pseudocode D. All
3. The \_\_\_\_\_\_\_ provides pictorial representation of given problem.
4. Algorithm B. **Flowchart** C. Pseudocode D. All
5. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ symbol is used at the beginning of a flow chart.
6. **Circle** B. Rectangle C. C. Diamond D. None
7. The \_\_\_\_\_\_\_ symbol is used to represent decision in flowchart.
8. Circle B. Rectangle C. **Diamond** D. None
9. The \_\_\_\_\_\_\_ symbol is used to represent process in flowchart.
10. Circle B. **Rectangle** C. Diamond D. None
11. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Symbol is used to represent input and output operation in flowchart.
12. Circle B. Rectangle C. Diamond D. **Parallelogram**
13. \_\_\_\_\_\_ is a symbol used connects two symbols of flowchart.
14. Circle B. Rectangle C. Diamond D. **Arrow**
15. In computer science, algorithm refers to a pictorial representation of a flowchart.
16. True B. **False**
17. The following box denotes?  
    [](https://www.sanfoundry.com/wp-content/uploads/2018/07/computer-fundamentals-questions-answers-flowcharts-q5.png)
18. **Decision** B. Initiation C. Initialization D. I/O
19. A box that can represent two different conditions.
20. Rectangle B. **Diamond** C. Circle D. Parallelogram
21. Which of the following is not an advantage of a flowchart?
22. Better communication C. Systematic testing
23. Efficient coding D. **Improper documentation**
24. What is an Algorithm?
25. A flowchart C. A decision
26. **Step by step instructions used to solve a problem** D. A Pseudocode
27. What is the difference between a flowchart and pseudocode?
28. A flowchart is diagrammatic whilst pseudocode is written in a programming language
29. A flowchart is textual but pseudocode is diagrammatic
30. **A flowchart is a diagrammatic description of an algorithm whilst pseudocode is a textual description of an algorithm**
31. A flowchart and pseudocode are the same thing
32. In a flowchart a calculation (process) is represented by
33. **A rectangle**  B. A rhombus  C. A parallelogram  D. A circle
34. When you write an algorithm the order of the instructions is very important.
35. **True**
36. False
37. What should be considered when designing an algorithm?
38. If the correct hardware is being used C. If the correct software is being used
39. **If there is more than one way of solving the problem** D. None
40. In a flowchart how are symbols connected?
41. Symbols do not get connected together in a flowchart
42. **With lines and an arrow to show the direction of flow**
43. With dashed lines and numbers
44. With solid lines to link events
45. When can algorithms be used?
46. Only with computers C. Only when programming
47. Only with flowcharts D. **Any time to design solutions to problems**
48. A flowchart
49. Helps you plan out  computer code C. Uses shapes to help organize a process
50. is a type of graphic diagram that represents an algorithm,  **D. All**
51. What does this shape represent?
    * 1. **Input/output** B. Decision C. Process D. Start/Stop
52. What is this symbol?
    * 1. Decision B. Input/output **C. Start/End**  D. Process
53. What is pseudocode?
    * 1. **Simplified programming language, that is not a specific language**
      2. Complicated programming language
      3. Simple programming language, which is linked to a specific language
      4. A type of cheese
54. The operation represented by parallelograms.
55. **Input/output** B. Assignment C. Comparison D. Conditions
56. Which of the following is not a flowchart structure?
57. **Process** B. Sequence C. Repetition D. Case
58. A \_\_\_\_\_\_\_\_\_\_\_ is a connector showing the relationship between the representative shapes.
59. Line b. arrow  
    c) Process  
    d) box

# CHAPTER TWO

# Fundamentals of C++ programming Languages

## **An overview of C++ program**

**What is C++?**

C++ is general purpose programming language, and it supports various programming styles such as object-oriented, procedural, functional, and so on. This makes C++ both powerful and flexible. It is used to program computers to perform specific tasks. Specialized software is used to manage the task of developing programs; in particular converting the program written in its programming language to binary form needed by the computer.

## **Basic structure of C++ program**

**A program in C++ is divided into three sections**

* + - 1. Standard Libraries section
      2. Main Function section
      3. Function Body section

**Example, look the following simple "Hello World" program:**

// my first program in C++ **output**

#include<iostream> hello world!

using namespace std;

int main() {

cout<<"hello world!";

return 0;

}

* C++ program is a collection of functions. The previous example contains only one function, main().
* As usual, execution begins at main().
* Every C++ program must have a main().
* In C++ the separation between instructions is specified with an ending semicolon (;) after each one.
* The C++ statements terminate with semicolon.

## **Basic Data Types in C++**

* **Primitive data types** are the built-in data types that C++ language provides. Therefore, we can use them directly to declare entities like variables, constants, etc.
* We can also call the Primitive data types as pre-defined data types or standard data types.

**In C++, there are different primitive (Built-in) data types**

* 1. **Integer (int)**
* **Integer** is a data type that is used in C++ programming to store only integer values.
* Example: 69, -30, 3230 etc.
* In C++, the **int** keyword is used to define integer.
* **Syntax**: int sd; where sd is variable.
  1. **Float (float), and Double (double)**
* **float** and **double** are used to store **floating**-**point** **numbers** (numbers with decimal point). Eg: 69.65, 3.1415
* Floats are 4 bytes in size, while doubles are 8 bytes. Hence, double has two times the precision of float.
* **Syntax**: float sd; or double sd; where **sd** is variable.
  1. **Character (char)**
* **Character** is a data type that is used in C++ programming to store characters like letters and punctuation marks.
* In C++, the **char** keyword is used to define a character.
* Syntax: char sd; where **sd** is variable.
* The char datatype has a 1-byte (8-bit) size.
* Characters in C++ are enclosed inside single quotes ' '.
  1. **Boolean (bool)**
* **Boolean** is a data type, used in C++ programming to store Boolean or logical values.
* Boolean variable has two possible values: **0** or **1**. where 0 represents boolean value **false** and 1 represents boolean value **true**.
* In C++, the **bool** keyword is used to define Boolean.
* Syntax: bool sd=false; where **sd** is variable.
* Booleans are used in conditional statements and loops.

## **Input/output in C++**

**C++ includes libraries that provide several ways for performing input and output.**

In C++, input and output are performed in the form of a sequence of bytes or more commonly known as streams (flow of data).

* **Input Stream:** When bytes flow from a device (for example, a keyboard) to main memory, this process is referred to as input operation.
* **Output Stream:** When bytes flow from main memory to a device (display screen), this process is referred to as output operation.

In C++, **cin** and **cout** objects are frequently used for taking **inputs** from user, and printing **outputs** to the screen respectively. To use **cin** and **cout** in C++ one must include the **iostream** **header** **file** (#include<iostream>) in the program.

## **Comments in C++**

In C++, **comments** are hints that a programmer can add to make their source code easier to read and understand. It helps a developer in explaining the logic of the code and improves program readability. At run-time, comments are completely ignored by C++ compilers.

**Two types of comments in C++**

* + 1. Single line comments, and
    2. Multi-line comments
    3. **Single line comments**

In C++, any line that start with double slash (//) is a single-line comment.

* **Single-line comments** starts with a double slash symbol and terminate at the end of line.
* **Example**: **c=5.0/9\*(f-32);**  //conversion formula
  + 1. **Multi-line comments**
* C++ introduces a multi-line comment that starts with a forward slash and asterisk /\* and finishes with an asterisk and forward slash \*/.
* The compiler will ignore any text written between **/\*** and **\*/**.
* **Example: c=5.0/9\*(f-32);**

/\* this is an example of C++ program to convert the given Fahrenheit to Celsius \*/

**Note:** Both **single-line** and **multi-line** comments in the program are ignored by the compiler.

## **C++ tokens**

Token is the smallest individual unit of a program written in any language.  It is the smallest element of a program that is meaningful to the compiler. It is source-program text that the compiler does not break down into component elements. The **keywords**, **identifiers**, **variables**, **constants**, **strings** and **operators** are examples of tokens. Punctuation characters such as **brackets** [ ], **braces** { }, **parentheses** ( ), and **commas** (,) are also tokens.

* 1. **Keywords**
* In a programming language, keywords are pre-defined or reserved words that have special meanings to the compiler.
* Each keyword in a program is intended to perform a specific function.
* Keywords cannot be used as variable names because by doing so, we are trying to assign a new meaning to the keyword, which is not allowed.
* **Example;** *int*, *do*, *while*, *asm* etc**.** 
  1. **Identifiers**
* Identifiers are names given to variables, arrays, functions, classes or other entities by the programmer.
* A valid identifier is a sequence of one or more letters, digits or underscores symbols. However, they must begin with either a letter or an underscore (\_) symbols.
* Identifiers can never begin with a digit.
* Neither spaces nor marked letters can be part of an identifier.
* Only letters, digits and underscore characters are valid.
* They cannot match any keyword of the C++ language.
* **Example**: int age=25; // age is a valid identifier.

**Examples** of Valid and Invalid identifiers

|  |  |
| --- | --- |
| **Valid Identifier** | **Invalid Identifier** |
| totalMark | 1age |
| age\_1 | int |

**Very important:** the C++ language is "**case sensitive**", that means an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. **For Example:** the variable **age** is not the same as the variable **AGE** or **Age**.

* 1. **Variables**
* In C++ programming, a variable is a location (storage area) in your computer's memory in which you can store a value (information) and from which you can later retrieve that value. Or,
* Variable is a name (identifier) given to a memory location. Each variable should have its own unique name (identifier) to indicate the storage area.

**Rules for naming a variable**

* A variable name can only have alphabets, numbers, and the underscore (\_) symbol.
* A variable name cannot begin with a number.
* It is a preferred to begin variable names with a lowercase character. Eg. age is preferable to Age.
* A variable name can start with an underscore. However, it is not considered a good practice.
* A variable name cannot be a keyword. **Example**, int is a keyword that is used to denote integers.

**Declaration of variables**

* In order to use a variable in C++, we must first declare it i.e. specifying the data types and data types name we want it to be.
* **Syntax** data\_type valid\_identifier;
* **For example**

int a;

float myNumber; All are valid variable declarations.

double b\_c;

char ch;

If you need to declare several variables of the same type and you want to save some writing work you can declare all of them in the same line separating the identifiers with commas.

**For Example:** int a, b, c;

It declares three variables (a, b and c) of type int and has exactly the same meaning as if we had written:

int a;

int b;

int c;

* 1. **Constants**
* A Constant is a memory location in which values can be stored. Unlike variables, its value can not be changed or modified after their definition.
* It means once we define a variable as the constant in a program, its value will be fixed and never be changed.

**Example:** const int a=50;

a=60; // Error

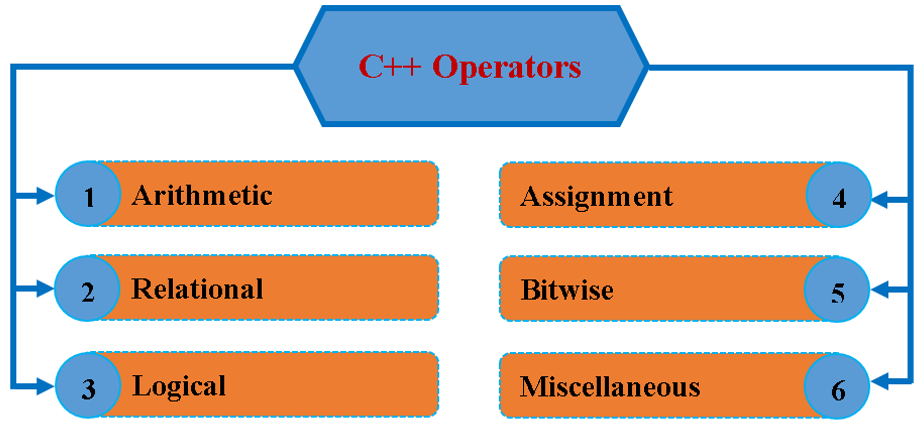
If we try to change the value of constant type variable, it shows an error message in the program.

**How to Define Constants in C++**

In C++, we can define constants in two ways

* + - 1. Using **#define** Preprocessor
  + It used to define constant
  + **Syntax**: #define variable\_name value
  + **Example**: #define PI 3.1415
    - 1. Using a **const** keyword
  + With the **const** prefix, you can declare constants with specific type in the same way as you would do with a variable.
  + **Syntax**: const data\_type variable\_name = intial\_value;
  + **Example**: const float PI = 3.1415;
  1. **Operators**

An **operator** is a symbol that tells the compiler to perform certain mathematical or logical manipulations. On the other hand, it is a symbol used for performing operations on operands.



* 1. **Arithmetic Operators**
* **Arithmetic Operators** are used to perform mathematical operations on the operands.
* C++ provides all the basic arithmetic operators like addition (+), subtract (-), Multiplication (\*), division (/), modulus (%), increment (++) and decrement (--).
* **Modulus** (**%**):- this operator returns the remainder after division.
* **Example**: int a=5, b=2;

int c=a%b; // c=1

* 1. **Relational operators**
* In C++, Relational Operators are used for comparison of two or more numerical values stored in an operand.
* After comparison, the returned result is a boolean data type value, either 0 (false) or 1 (true).
* C++ provides six relational operators like equal to (==), less than (<), less than or equal (<=), greater than (>), greater than or equal (>=) and not equal (!=). Relational Operators used for decision-making process.
  1. **Logical operators**
* To determine whether an expression is true or false, we use logical operators. If the expression is true, it returns 1, and if it is false, it returns 0.

**C++ has the following three logical operators.**

1. Logical and (**&&**)
2. Logical or (**||**)
3. Logical not (**!**)
   1. **Assignment operators**

* Used to assign the result of an expression to a variable and the symbol used is "=". it is of 3 types.

**C++ has the following three assignment operators.**

1. Simple assignment------------a = 9;
2. Multiple assignment----------a = b = c = 36;
3. Compound assignment-- a+=15; (equal to a=a+15)
   1. **Conditional operator (ternary operator)**

* A ternary operator (conditional operator) evaluates the test condition and then executes a block of code based on the condition's result.
* **Syntax**: test\_condition? expression1 : expression2;
* test\_condition is evaluated first, if the result is true then expression1 is evaluated else expression2 is evaluated and that value becomes the value of the expression.
  1. **Bitwise operators**
* Performs operation on the bit level. High-level language does not support the bitwise operators.

1. **Bitwise operators;** This adds corresponding bits in its operands, if both are 1 result is 1 otherwise 0.

**Example:** if a = 8 00001000

b = 3 00000011

**00000000** = **0 (zero) in decimal**

1. **Bitwise or (|):** if one of the bits is 1, then result is 1. If both are 0 result is 0.

**Example:** if a = 8 00001000

b = 3 00000011

* + 1. = **11 (eleven) in decimal**

1. **Bitwise exclusive or (^):** If one side bit is 0 and the other side is 1, then the result bit is 1, if both sides are the same then the result is 0.

**Example:**  if a = 8 00001000

b = 3 00000011

**0000**1**0**11 = **11 (eleven) in decimal**

1. **Bitwise complement (~):** This is a unary operator which invert the bits in its operand i.e., 1 becomes 0 and 0 becomes 1.

**Example**: if a=18 then 10010 1’s complement 01101 = -19

1. **Bitwise Shift operators:**
   * **(<<) bitwise left shift;** It shifts bits to the left. **Example:** if a=8, b=2 then c=a<<b i.e. a is to be shifted 2 places to the left. So, c becomes 32 in decimal. a = 00001000 after shift c= 00100000
   * **(>>) bitwise right Shift**; It shifts bits to the right. **Example**: if a=8, b=2 and c=a>>b i.e., a is to be shifted 2 places to the right. **Therefore**, c becomes 2 in decimal. a = 00001000 after shift c=00000010.

## **Library Functions**

C+ + consists of many library functions which contain the functions that are used in the program construction of the language. These are the header files that are to be included before main () & are sometimes termed as preprocessor statements.

**Here are some files given.**

* 1. **iostream** - Standard input/output streams like cin, cout etc.
  2. **math.h** - Mathematical functions like sin (), cos (), sqrt (), log () etc.
  3. **stdlib.h** - Standard library functions like conversion of one type to other etc.
  4. **string.h** - String manipulation functions like strcpy (), strcat (), strcmp () etc.
  5. **ctype.h** - Declares functions for testing characters. E.g. isalpha (), isnum (), islower (), toupper (), tolower () etc.
  6. **time.h** - Includes date & time functions.
     1. Who invented C++?
        + 1. Dennis Ritchie C. Ken Thompson
          2. Brian Kernighan **D. Bjarne Stroustrup**
     2. What is C++?
        + 1. C++ is an object oriented programming language
          2. C++ is a procedural programming language
          3. **C++ supports both procedural and object oriented programming language**
          4. C++ is a functional programming language
     3. Which of the following is used for comments in C++?

1. /\* comment \*/ C. // comment \*/
2. // comment **D. both // comment or /\* comment \*/**
   * 1. Which of the following is a correct identifier in C++?
        + 1. **VAR\_1234** C. $var\_name
          2. 7VARNAME D. 7var\_name
     2. Which of the following type is provided by C++ but not C?
        + 1. double B. float C. int **D. bool**
     3. By default, all the files in C++ are opened in \_\_\_\_\_\_\_\_\_ mode.
        + 1. Binary B. VTC **C. Text**  D. ISCII
     4. What will be the output of the following C++ code?

#include<iostream>

using namespace std;

int main () {

int cin;

cin >> cin;

cout << "cin: " << cin;

return 0;

}

1. Segmentation fault C. Nothing is printed
2. Error **D. cin: garbage value**
   * 1. What will be the output of the following C++ code?

#include <iostream>

using namespace std;

int main() {

char c = 74;

cout << c;

return 0;

}

1. I **B. J** C. A D. N
   * 1. Which keyword is used to define the macros in C++?
        + 1. #macro **B. #define** C. macro D. define
     2. Which of the following symbol is used to declare the pre-processor directives in C++?
        + 1. $ B. ^ **C. #** D. \*
     3. Which of the following is the correct syntax to print the message in C++ language?
        + 1. **cout <<"Hello world!";** C. cout << Hello world! ;
          2. out <<"Hello world!; D. None of the above
     4. Which of the following statements is correct about the formal parameters in C++?
        + 1. **Parameters with which functions are called**
          2. Parameters which are used in the definition of the function
          3. Variables other than passed parameters in a function
          4. Variables that are never used in the function
     5. For inserting a new line in C++ program, which statement can be used?
        + 1. **\n** B. \r C. \a D. None

# CHAPTER THREE

# Control statement

Control statements allow decision making within programs. Control statements come in several different forms to provide flexibility in the controlling of program execution.

* One form is to optionally execute C ++ statements.
* Another form is to form a loop, where all the statements in a loop are executed repeatedly until the exit condition is satisfied.
* Another form is to branch to a certain statement and continue execution.

## **Conditional statement**

**Decision making with if statement**

* The **if** statement is a powerful decision making statement and is used to control the flow of execution of statements. It is a *two-way* decision statement and is used in conjunction with an expression.
* **If** can be used in different forms depending upon nature of conditional test.

1. if 2. if…else 3. nested–if.

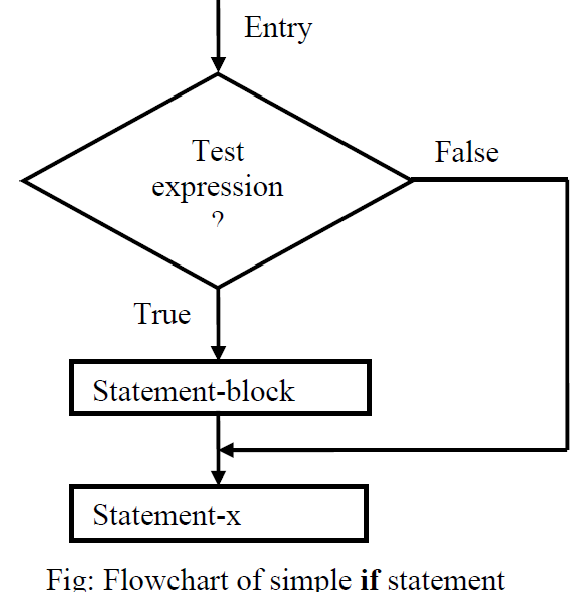
1. ***Simple* *if statement***

* For example, the following code fragment prints out x is 100 only if the value stored in variable x is indeed 100

**if (x = = 100)**

**cout << "x is 100";**

This is illustrated in the following figure.



1. ***if – else statement:***

* The **if…else** statement is an extension of the simple **if** statement. The general form/syntax is:

**if (test expression)**{

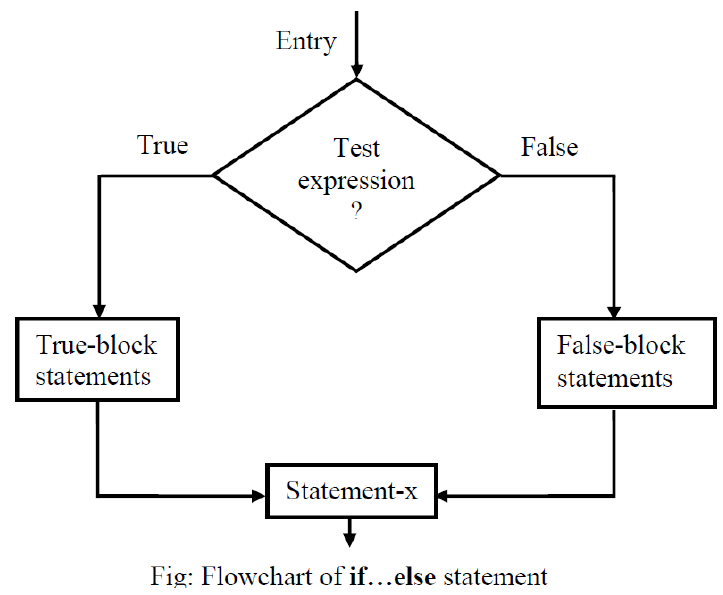
true block statement(s);

} **else** {

false block statement(s);

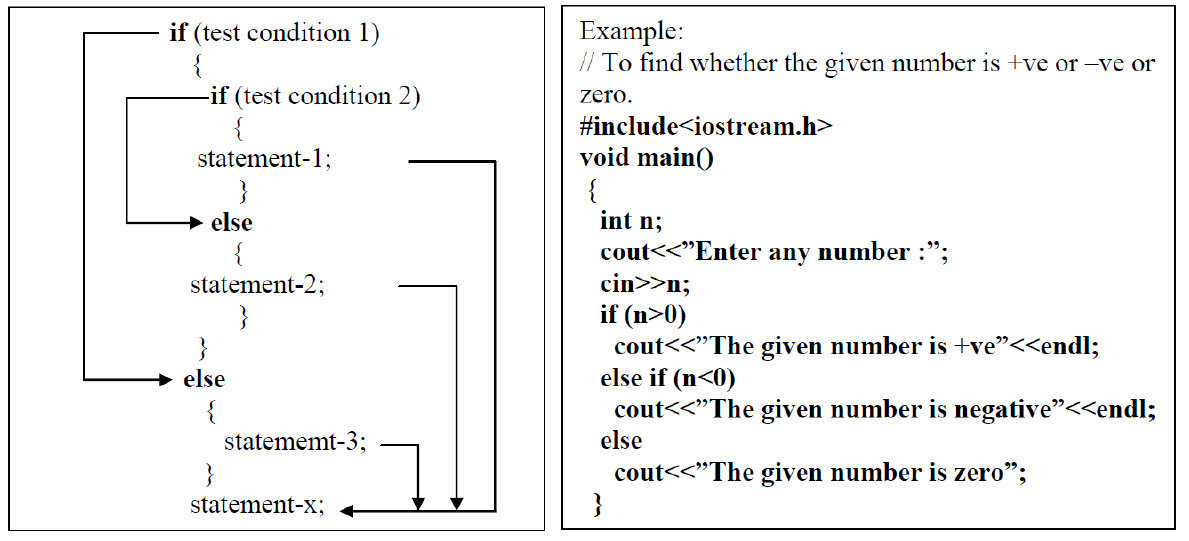
}

statement-x;



1. ***Nested if statement***

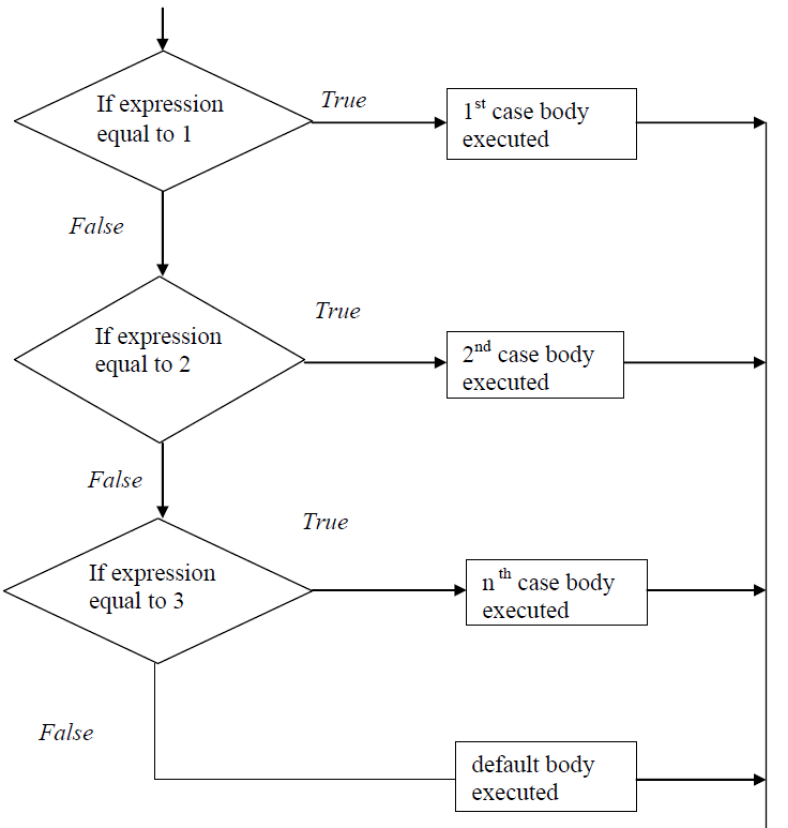
* When a series of decisions are involved, we may have to use more than one **if…else** statement in nested form as follows:

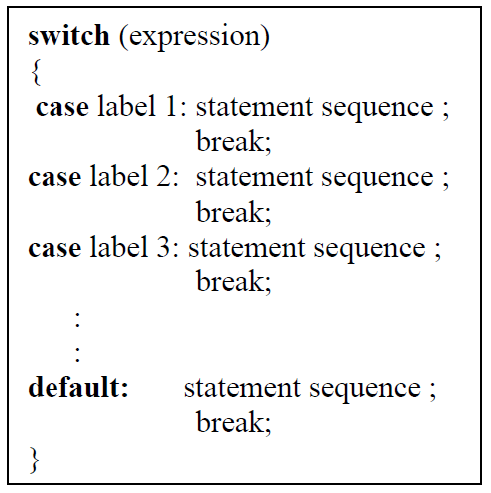


1. ***The switch-case statement***

* This is a multiple branching control statement. It is useful for decision making when more than one case is involved.

**The general form of switch statement is as follows**



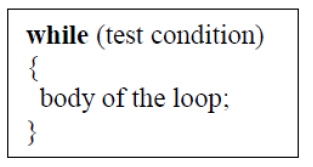


## **Loop statements**

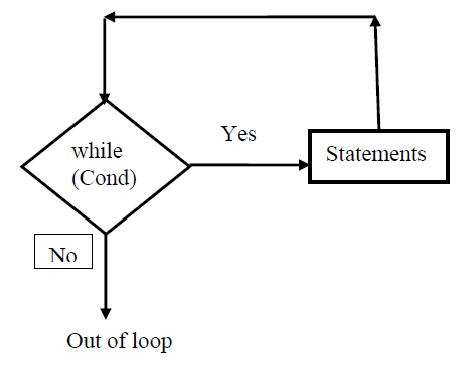
1. ***the while loop statement***

* The simplest of all the looping structures in C++ is the while statement.

**The basic format/syntax of the while statement is:**

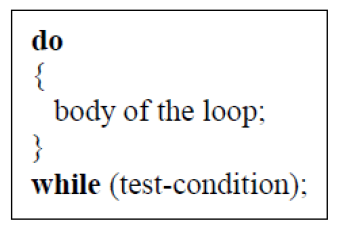


**The flowchart of while loop is as follows**

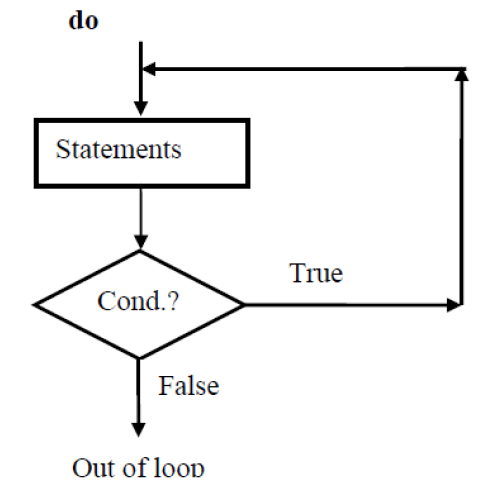
****

1. ***the do-while loop statement***

**The basic form of the do-while statement is as follows:**

****

**The flowchart of do-while loop is as follows:**

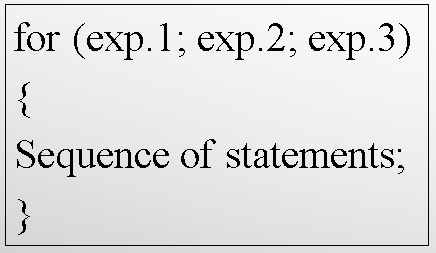
****

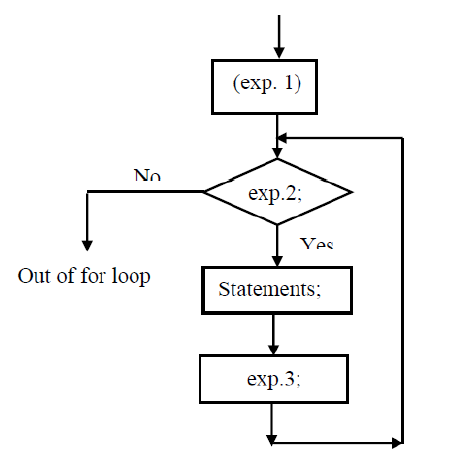
1. ***for loop statement***

* This is used when we know the number of times the instructions are to be executed.
* **Syntax:**

**Here**

* **exp.1** – initial value
* **exp.2** – Testing condition,
* **exp.3** - step value (increment /decrement)





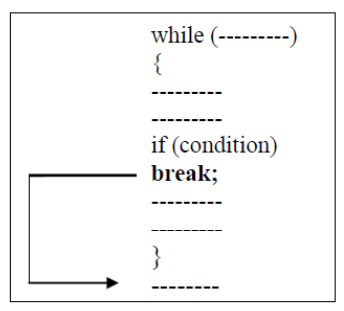
## **Breaking control statements**

1. **Break** statement:

* The **break** statement causes an immediate exit from the do, for, switch, and while statement in which it appears, the program continues executing with the next program statement following the do, for, switch, and while statement block.

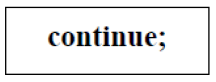
**The format of the break statement is simply**

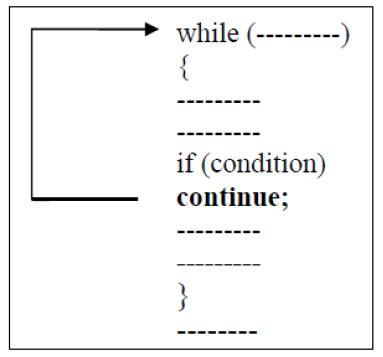
**break**;



1. **Continue** statement

* The **continue** statement tells the compiler, “**skip the following statements and continue with the next iteration.**”
* The format of the continue statement is simply





1. **The goto** instruction

* It allows making an absolute jump to another point in the program. The destination point is identified by a label, which is then used as an argument for the goto instruction.
* A label is made of a valid identifier followed by a colon (:).

**Multiple Choice Questions and Answer on Chapter Three**

1. A flowchart needs to represent the a situation where for each mark a student is award 'Pass' or 'Fail'...the system will consider the mark and if it's 50 or over award 'Pass', else it awards 'Fail'. This is an example of which of the algorithm constructs?
2. **Decision** B. Loop  C. Sequence  D. All
3. The action performed by a \_\_\_\_\_\_\_\_\_\_\_ structure must eventually cause the loop to terminate.
4. Sequence B. case **C. repetition**  D. process
5. How many loops are there in C++ 98?
   * + - 1. 1 B. 2 **C. 3** D. 4
6. Which of the following can replace a simple if-else construct?
   * + - 1. while loop D. do-while loop
         2. for loop **E. Ternary operator**
7. Each pass through a loop is called a/an
   * 1. Enumeration B. **iteration** C. culmination D. pass through
8. What is true about the loop structure…?
   * 1. Condition B. loop control variable C. body of the loop **D. all**
   1. What is the final value of x when the code int x; for(x=0; x<10; x++) {} is run?
      1. 0 B. 1 C. 9 **D. 10**
   2. Decision Control statements in C++ can be implemented using
      1. If B. if-else C. Conditional Operator **D. All**
   3. What is currect syntax of for loop?
      1. **for(initialization; condition; increment/decrement)**
      2. for(increment/decrement; initialization; condition)
      3. C. for(initialization, condition, increment/decrement
      4. D. None of these
   4. Which of the following is an entry-controlled loop?
      1. do-while loop B. while loop C. for loop **D. Both (B) and (C)**

# CHAPTER FOUR

# C++ FUNCTIONS

## **Introduction to functions**

A function is a block of code that performs a specific task. In order to avoid complexity of a program while coding, debugging & testing, the program is divided into functional part or subprograms. Dividing a complex problem into smaller chunks makes our program easy to understand and reusable.

***There are two types of function:***

* 1. **Standard library functions:** predefined in C++
* For example, function strcat () to concatenate two strings, function memcpy () to copy one memory location to another location and many more functions.
  1. **User-defined function:**created by users
* C++ allows the programmer to define his or her own function.
* A user-defined function group’s code to perform a specific task and that group of code is given a name (identifier).

***Function has three sections***

* + ***Function declaration -*** tells the compiler about a function's name, return type, and parameters.
  + ***Function definition*** - provides the actual body of the function.
  + ***Calling a Function*** - to use the function, we need to call it.

**General form:**

**type-specifier function-name (*argument declaration*) {**

*Body of the function;*

*return (expression);*

**}**

* + - ***Type-specifier*** — specifies type of value (data type) that the return statement of function returns. By default, function returns integer value if not specified.
    - ***Argument*** — is a comma separated list of variable names that receive the values, when function is called (parenthesis required even no arguments) Example: int dol () returns int value to main. float sum () returns float value to main.
    - ***Return*** —

1. Causes an immediate exit from the function.
2. It returns the value--when this is encountered, control is passed back to the calling function. If function is declared void then there is no need to include return statement.

## **Scope of variables**

1. ***Local variables***

* Variable declared within a function are called local variables.
* They are created when the function is called and destroyed automatically when the function is exited.
* Scope of the local variable is within the function only; it is not valid outside of the function.

1. ***Global variables***

* Global variables are declared separately, preferably outside the main function.
* They are accessible to all functions included in the program. Therefore, Scope of the global variable is throughout program.
* Once a variable has been declared as global, any function can use it and change its value.

## **Calling a Function**

Function can be called in two ways.

* + 1. Call by value
    2. Call by reference

1. ***Call by value***

This is one way of data transformation from calling portion to called portion. Means changes inside the function cannot affect the main program.

#include <iostream>

using namespace std;

void greet(int c) {

c=10;

}

int main() {

int c=5;

greet(c); // calling the function

cout << “The value of c is “<<c;

return 0;

}

1. ***Call by reference***

This is two way of data transformation from calling portion to called portion and called portion to calling portion. Means when function is called by reference, changes inside the function affect main program also. When function is called, argument corresponding to a reference parameter is not copied.

#include <iostream>

**Output**

**The value of c is 10**

using namespace std;

void greet(int \*c) {

\*c = 10;

}

int main() {

int c=5;

greet(&c); // calling the function

cout << "The value of c is "<<c;

return 0;

}

## **Argument passing**

***Functions depend on whether arguments are present or not, whether a value is returned or not, can be classified as:***

* + Functions with no arguments & no return values.
  + Functions with arguments & no return values.
  + Functions with arguments & return values.

## **C++ Function Overloading**

* In C++, two functions can have the same name if the number and/or type of arguments passed is different.
* These functions having the same name but different arguments are known as overloaded functions.

**For example:**

#include <iostream>

using namespace std;

int greet(int a, int b) {

int c = a+b;

cout <<"The sum is "<<c<<endl;

}

double greet(double a, double b) {

double c = a+b;

cout <<"The sum is "<<c;

}

int main() {

greet (10,20);

greet (10.5, 20.5);

return 0;

}

## **Inline functions**

* In C++, we can declare a function as inline. This copies the function to the location of the function call in compile time and may make the program execution faster.
* The inline specifier indicates the compiler that inline substitution is preferred to the usual function call mechanism of a specific function.

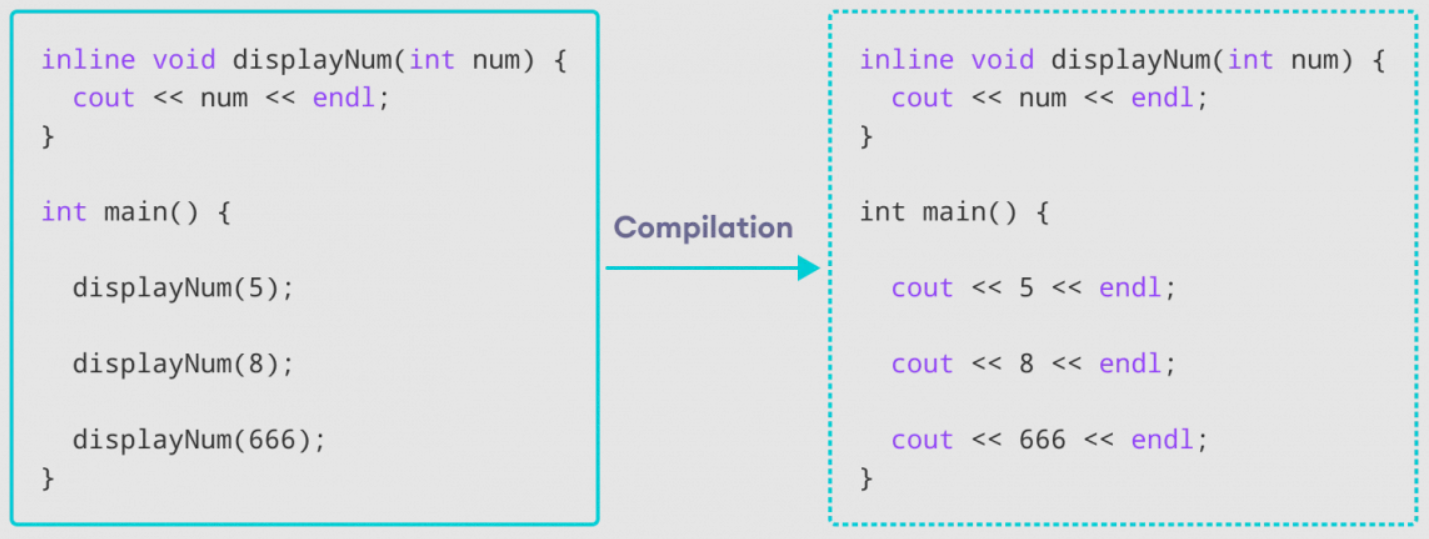
***The format for its declaration is:***

*inline type name (arguments ...)*

*{*

*instructions...*

*}*



## **Recursive Functions**

* A function is said to be recursive if a statement in the body of the function calls itself.
* It is useful for many tasks, like sorting or calculates the factorial of numbers.

**For example;**

#include <iostream>

using namespace std;

void greet(int a) {

if(a>0){

cout <<a<<" ";

a--;

greet(a);

}

}

int main() {

greet(10); // calling the function

return 0;

}

**Multiple Choice Questions and Answer on Chapter Four**

* + - 1. Which is more effective while calling the C++ functions?
         1. call by object C. call by pointer
         2. call by value D. **call by reference**
      2. Which of the following is used to terminate the function declaration in C++?  
         **A.;** B.] C. ) D. :
      3. Pick the incorrect statement about inline functions in C++?

1. Saves overhead of a return call from a function
2. **They are generally very large and complicated function**
3. These functions are inserted/substituted at the point of call
4. They reduce function call overheads
   * + 1. Which of the following is the default return value of functions in C++?
          1. **int** B. char C. float D. void
       2. What is an inline function?
          1. **A function that is expanded at each call during execution**
          2. A function that is called during compile time
          3. A function that is not checked for syntax errors
          4. A function that is not checked for semantic analysis
       3. An inline function is expanded during \_\_\_\_\_\_\_\_\_\_\_\_\_\_
          1. **compile-time**  C. run-time
          2. Never expanded D. end of the program
       4. When we define the default values for a function?
          1. When a function is defined **C. When a function is declared**
          2. When the scope of the function is over D. When a function is called
       5. What will be the output of the following C++ code?

#include <iostream>

using namespace std;

int fun(int=0, int = 0);

 int main() {

cout << fun(5);

return 0;

}

int fun(int x, int y) { return (x+y); }

* + 1. -5 B. 0 C. 10 **D. 5**
       1. Which of the following is important in a function?
          1. Return type
          2. Function name
          3. **Both return type and function name**
          4. The return type, function name and parameter list

**Links for More C++ question with Answer.**

1. <https://www.sanfoundry.com/cplusplus-interview-questions-answers/>
2. <https://www.javatpoint.com/cpp-mcq>
3. <https://engineeringinterviewquestions.com/c-programming-multiple-choice-questions-and-answers/>
4. <https://engineeringinterviewquestions.com/c-programming-multiple-choice-questions-and-ansswers/>
5. <https://www.sanfoundry.com/cplusplus-programming-questions-answers-functions/>
6. <https://technictiming.com/c/control-statements/>