# Programming Terminology

## Introduction to Program Design

This chapter describes the various activities carried in a Program development to provide quality software. In case of software development the programmer have to decide

- what to do?
- how to do? and
- when to do.

Before writing the programs he must decide that what steps the computer should follow to solve the identified problem. This should be framed in a step-by-step procedure. This systematic procedure used to solve a particular problem is called an algorithm. When this procedure is presented in the form of a chart, is called a flowchart.

Traditionally program design has been linked with flowcharting. However, flowcharts are probable more useful in documentation rather than design. A complex structure of the problem creates the complexity in designing of flowchart. Flowcharting techniques are useful in describing program structure and in explaining programs. Some other techniques used to design programs are as follows :

## Modular Approach (Modular Design)

In the early days of computer era, there were no well defined procedures for managing the software development process. The use of computers was limited to scientific and engineering applications. The programs were written for well-defined problems. If the program did not work, the programmer will correct it. People started calling this approach of program development as **code-and-fix.**

When the use of` computers for solving complex problems started, the problems begin to appear with this approach. In many cases, it was found that the programmer could not really understand what the user wanted the program to do? And correcting problems often turned into an expensive and time consuming job. The programmers did not document their programs and even some programmers developed their own cryptic style so that they can secure their continued employment as no one else can figure out what the program is doing. Because of these practices, these early programs were almost impossible to debug and maintain, especially if the original programmer had left the company.

To handle these problems, the Program Development Life Cycle (PDLC) was introduced in the 1970s, and is still in widespread use. The PDLC provides an organized plan for breaking down the task of program development into manageable chunks, each of which must be successfully completed before moving on to the next phase.

The main aim of modular approach is to divide a project into segments and smaller units in order to simplify the analysis, design and programming efforts.

"A method in which long programs are divided into smaller programs or modules that can be designed, coded and debugged separately with a minimum amount of interaction is known as modular design, modular approach or modular programming."

## Program Design Tools

The various program design tools are described below :

**(a) Structure chart :** A structure chart is also known as an organogram hierarchy chart that shows the top-down design of a program. Each box in the chart indicates a task that the program must accomplish. The top module is also known as the main module or control module. It transfers the control to other modules.

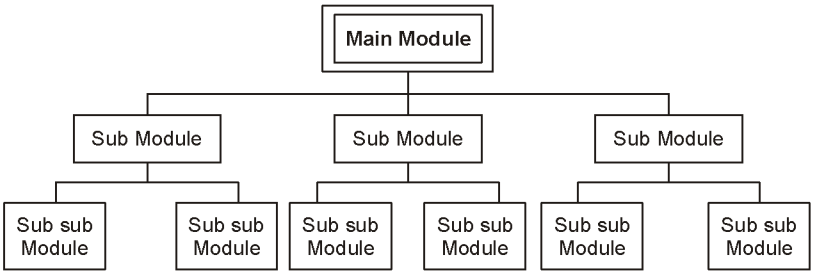



**Fig. 1.1 Structure Chart**

**(b) Algorithm :** An algorithm is a step-by-step procedure used to solve a particular problem. In other words, an algorithm is a precise specification of a sequence of instructions which is carried out in order to solve a particular problem.

**Complete**$(q, L(KAB), n)$
*input:*
$q$ - the user's query
$L(KAB) = \{(t, w(t), LA(t))\}$ where:
$t$ is a term
$w(t)$ its normalized frequency in the KAB
$LA(t) = \{(t_i, w(t, t_i))\}$
$(t, t_i)$ is a lexical affinity
$w(t, t_i)$ its normalized frequency in the KAB
$n$ - the maximum number of terms for completion

for each $t \in q$, retrieve $(t, w(t), LA(t))$ from $L(KAB)$
create a graph $G = (V, E)$ where:
$V = \{t \in q \text{ associated with } w(t)\}$
$E = \{(t_i, t_j) \text{ associated with } w(t_i, t_j) | t_i, t_j \in V\}$
$(t_j, w(t_i, t_j)) \in LA(t_i)$
compute the connected components of $G$, $C_1, \ldots, C_n$
eList $\leftarrow \emptyset$
$W \leftarrow 0$
for each component $C_i = \{t_1, \ldots, t_k\}$
$w(C_i) \leftarrow 1/k \sum_{j=1}^{k} w(t_j)$
$W \leftarrow W + w(C_i)$
Compute $LA(C_i) = \{(t, w'(t))\}$ where:
$t \in \bigcap_{i=1}^{k} LA(t_i)$
$w'(t) = 1/k \sum_{j=1}^{k} w(t, t_j)$
sort $LA(C_i)$ in decreasing order according to $w'(t)$
expand eList with the first $\frac{w(C_i)}{W} * n$ terms from $LA(C_i)$
return eList

**(c) Flowchart :** A flowchart is a pictorial/graphical representation of an algorithm. Programmers create flowcharts either by hand using a flowcharts template or on the computer. Each flowcharting symbol has a meaning. For example, a parallelogram indicates an input or output, a rectangle indicates a process, and diamond indicates a condition etc.

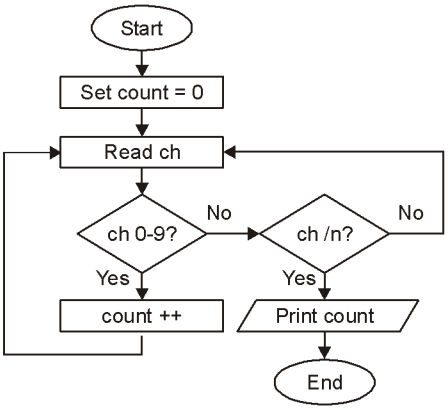



**Fig. 1.2**

## Structured Programming

In the 1960s and early l970s, theorists interested in algorithms developed techniques called structured programming. Because structured programs are easier to understand and test, they are usually more reliable than unstructured programs.

Structured programming is not a new programming language; it is a way of standardizing computer programming using existing languages. One of the objectives of structured programming using existing languages; One of the objectives of structured programming is to uncomplicated and untangle computer programs. For example, too much use of the GOTO statement is one of the biggest problems. Articles in computer related journals, with the titles such as "GOTO Statements Considered harmful" and "Spaghetti Code," have discussed and documented the problems of using GOTO statements. One way to see the effect of GOTO statements is to draw lines in the program with a pencil or pen that show where the GOTO statements send the computer. A program with a large number of GOTO statements will look like a bowl of spaghetti ro a tangled clump of fishing line. Because of the problems of GOTO statements, some programmers do not use any in their programs.

The basic idea behind structured programming is that a program can be broken down into groups of statements. There is no size restriction on how many statements can be in a group, but I usually these groups are formed according to their function. One group may read data, while another group does a certain processing task.

When using the structured programming approach, statements in a group must conform to a standardized pattern or structure. To begin with, there can only be one entering point into the block of statements and one exit point from it. Therefore, you cannot branch to or from the middle to the structured group of statements. This restriction eliminates many programming errors and makes debugging much easier. Further more, there should be no groups of statements that cannot be reached or executed. Using the structured programming approach, each group can be tested separately. What structures or patterns are allowed when using structured programming? There are only three types : (l) sequence (2) decision, and (3) loop.

In the sequence structure, there must be a definite starting and ending point. After starting the sequence, programming statements are executed one after another until all the statements in the sequence have been executed. Then the program continues to another sequence.

The decision structure allows the computer to branch, depending upon certain conditions.

Normally, there are only two possible branches. As with all structure, there is a starting point and an ending point for the decision structure. It is important to note that regardless of which branch is taken the ending point is the same?

The final structure is the loop. Actually, there are two commonly used structures for loops. One is the do until structure, and the other one is the do while structure. Both accomplish the same thing. In the do until structure, the loop is done until a certain condition is met. For the do while structure, the loop is done while a certain condition exists.

The C language is designed to support structured programming techniques.

Thus Structured programming is an orderly arrangement of what a program tells a computer to do. Older computer languages used an arrangement of instructions like this :

- Do the first thing
- If a condition is satisfied, go to instruction number 5.
- Do a second thing.
- Do a third thing.
- Do the last thing.

This program will do only the first and last things, in that order, if the condition is satisfied.

Otherwise, it will do all the four things.

An equivalent structured program will use an arrangement like the following :

Do the first thing.
If a condition is not satisfied, then.
Do a second thing.
Do a third thing.
Do the last thing.

In a structured program, the instructions are not numbered or labeled there is no provision for "going to" a particular instruction.

In a structured program, instructions are executed in the order written, but they may be grouped into blocks, controlled by an instruction just before or after a block. As you see from the example above, a structured program tends to resemble an outline. Structured languages have different ways of denoting blocks of instructions; the C language puts braces around them.

## Program Development Life Cycle

Program development life cycle is a systematic approach of developing a quality software. It provides an organized plan for breaking down the task of program development into manageable chunks, each of which must be successfully completed before moving on to the next phase. The program development process is divided into following phases :

(a) Defining the problem.
(b) Designing the program.
(c) Coding the program.
(d) Testing and debugging the program.
(e) Documenting the program.
(f) Implementing and maintaining the program

**(a) Defining the Problem :** The first step in developing a program is to define the problem. In major software projects, this is a job of system analyst, who assigns to programmers in the form of a program specification. The program specification precisely defines the input data, the processing that should take place, the format of the output reports and the user interface. Depending on the size of the job, the program development might be handled by an individual or a team.

**(b) Designing the Program :** An architect doesn't design a home just by going to the site with cement and bricks. After determining the needs of the house's owner, the next step in designing a house is an architectural drawing. In this map is drawn on a plan paper. Like an architect's drawing, the program design specifies the components that make the program work.

Program design begins with focusing on the main goal that the program is trying to achieve and then breaking the program into manageable components, each of which contributes to this goal. This approach of program design is called top-down program design or modular programming. The first step involves identifying the main routine, which is the program's major activity. From there, programmers try to break down the various components of the main routine into smaller units called modules, until each module is highly focused. Experience shows that this is the best way to ensure program quality. If an error appears in a program designed this way, it is relatively easy to identify the module that must be causing the error.

For each module, the programmer draws a conceptual plan using an appropriate program design tool to visualize how the module will do its assigned job.

**(c) Coding the Program :** Coding the program means translating the algorithm into specific programming language instructions. While writing the code, prefer to use only well defined control structures. This technique of programming using only well defined control structures is known as structured programming.

The programmer must choose an appropriate programming language and then create the program by typing the code. The programmer must carefully follow the language's rules, which precisely specify how to express certain operations. Violation of language rules results in grammatical errors, more precisely known as syntax errors. Program development tools; such as a compiler, can check these syntax errors. These syntax errors must be eliminated before moving on to the next phase.

**(d) Testing and Debugging the Program :** After the removal of syntax errors, the program will be executed. However, the output of the program may not be correct. This is because of logical errors in the program. A logical error is a mistake that the programmer made while designing the solution to the problem. For example, a programmer tells the computer to calculate the net pay by adding deductions to the gross salary instead of subtracting. A program development tool, such as compiler, can not detect these errors.

Therefore, the programmer must find the correct logical errors by carefully examining the program output for a set of data for which results are already known. Such type of data is known as test data.

If the software developed is a complex one and consists of a large number of modules, then testing must be performed on these modules separately. This is known as unit testing. Once each module is thoroughly tested, they are integrated together to form a software package. Finally, testing is performed on the software package to uncover any sort of errors that may creep in as a result of integration of different modules. This is known as system testing.

Remember that it is not always possible to examine every output for each program condition i.e. exhaustive testing is not always possible. Inevitably, some errors will surface only when the system is implemented fully for a longtime.

Syntax errors and logical errors are collectively known as bugs. The process of identifying and eliminating these errors is known as debugging.

**(e) Documenting the Program :** After testing the software, project is almost complete. The structure charts, pseudocodes, flowcharts and decision tables developed during the design phase become documentation for others who are associated with this software project. In addition, more documentation needs to be done as the programs are being coded such as list of variable names and definitions, description of files that the program need to work with, and format of output that the program produces. All of this documentation need to be saved and placed together for future reference.

This phase ends by writing a manual that provides an overview of the program's functionality, tutorials for the beginner, in-depth explanations of major program features, reference

documentation of all program commands and a thorough description of the error messages generated by the program.

This manual is given to the user when the program is installed.

**(f) Implementation and Maintaining the Program :** In the final phase, the program is installed at the user's site. Here also, the program is kept under watch till the user goes green signal to it. Users may discover errors that were not caught in the testing phase, no matter how exhaustively the program was tested.

Even after the software project is complete, it needs to be maintained and evaluated regularly. In program maintenance, the programming team fixes program errors that users discover during its day to day use. In periodic evaluations, the team asks whether the program is fulfilling its objectives.

This evaluation may lead to modifications to abandon the current system and develop a new one, and so the program development life cycle begins afresh.

It is important to note that the best written software may fail if either it does not meet the user's requirements or the user interface is poorly designed. Note that the user is not concerned with the technicalities of the program logic, his/her prime concern is that whether the software solves his/her problem and the ease with which he/she can use the software.

## Algorithm

An algorithm is a finite set of steps that describes the solution of a particular problem or "An algorithm is a precise specification of a sequence of instructions which is carried out in order to solve a particular problem." An algorithm may be expressed in English like language.

The algorithm must be designed in such a way that if it is properly converted into a program, the desired result should be obtained.

### Characteristics :

**(1) Input :** There are one or more values which are supplied from the external environment.

**(2) Output :** At least one value should be produced i.e. after termination of algorithm the results should be obtained.

**(3) Definiteness :** Each and every step must be concise, clear and unambiguous.

**(4) Finiteness :** In algorithm one or more steps should not be repeated infinitely. The algorithm must terminate after a finite number of steps.

**(5) Effectiveness :** Each step must be sufficiently basic, definite and feasible. In order to solve a particular problem, each step of an algorithm must be in a sequence. Otherwise the logical errors may create in the program. The algorithm may be expressed in a number of ways. Some algorithms are expressed in the same sequence as the program will be written in high level language. These are called language dependent algorithms. Some algorithms are written in the form of descriptive language. These are called language independent Algorithms.

**Example :** Write an algorithm to find the largest number among three numbers A, B and C.

**Algorithm :**

Step l. Read the number A, B and C.
Step 2. If A > B then goto step (3) else goto step (4)
Step 3. If A > C then goto step (5) else goto step (6)
Step 4. lf B > C then goto step (7) else goto step (6)
Step 5. Print A is the largest and goto step (8)
Step 6. Print C is the largest and goto step (8)
Step 7. Print B is the largest and goto step (8)
Step 8. Stop

## Flowchart

Flowchart is a next stage of algorithm used in program development life cycle. "A flowchart is a graphical (pictorial) representation of an algorithm." In the flowchart, different types of symbols are used to represent different instructions/operations. The actual instruction is written within the boxes using clear and concise statements. According to the flow of instructions the boxes are connected by rigid lines containing arrow signal. The arrow signal represents the direction of instructions.

Since the flowchart is a pictorial representation of program logic therefore it minimizes the errors possible in logic development. When the flowchart is completed and tested by executing it manually, it makes program coding, development more easy and convenient. In this way the

flowchart is basically a logically structural layout of a plan used to develop a program/software. It guides the programmer how to achieve the aims of the given project.

Sometimes the experienced system analyst/programmer develops the software/programs without drawing flowcharts but it is not a good practice. For beginners, the flowchart reduces the logical errors and omissions in the program.

When we plan to develop large software's. Instead of going into actual coding/direct coding, one should design the flowchart first. This practice reduces the risk factor. To understand this better, let's take an example of constructing a building. Suppose the builder directly starts/acts on constructing the building without first designing a map and in between the construction, if he realizes/thinks that the design in progress has some drawbacks in fulfilling his requirements or some modification is required, then at that stage it is not possible for him to alter the changes/modifications without demolishing the entire or some part of the construction. There fore, before designing anything, one should well analyses and plan what he really wants? so that there is less need of modification/changes. In this way possible errors can be omitted.

### Symbols used to Design Flowchart

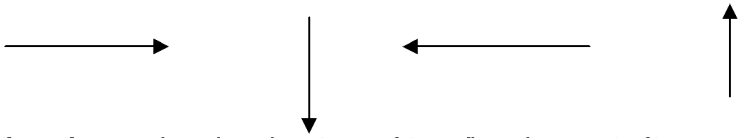
We have already discussed that a flowchart uses boxes of different shapes to denote different types of instructions. The symbols used in the flowchart have standardized by the American National Standards Institute (ANSI). Following are symbols used to draw the flowchart.

**Ellipse :** This is also called as terminal. It is used to indicate the beginning (START), ending (STOP), and pauses (HALT) in the program logic flow. Ellipse is first symbol and the last symbol used in the program logic?

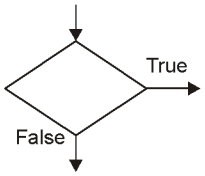
**Parallelogram :** This symbol is used to denote input/output operations in the flowchart.

**Rectangle :** A rectangle is used in a flowchart to represent arithmetic and data movement instructions. Thus, all arithmetic operations of adding, subtracting, multiplying and dividing are shown by this symbol. lt is also used to represent the initialization process.

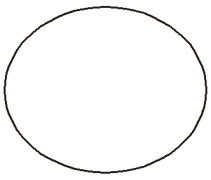
**Flowlines :** Flowlines with arrowheads are used to indicate the flow of operation, i.e.. the exact order in which the instructions are to be executed. The normal flow of flowchart is from top to bottom and left to right.



**Rhombus :** The Rhombus is used in a flowchart to indicate a point at which a decision has to be made and branching is also permitted.




**Connector :** When a flowchart becomes very long, the flow lines start criss-crossing at many stages that create confusion/complexity. When a flowchart becomes too long to complete in a single page in that situation we use connectors to establish the continuity/connection of the flowchart on the next page. A connector symbol is represented by a circle and a letter or digit is placed within the circle to indicate the link.



## Programming Paradigms

Firstly, we need to examine two different programming paradigms :

- Procedural programming
- Object Oriented programming

The important difference between these two is that procedural programming treats every program as a list of instructions to be read in a top-down fashion, while Object Oriented (or OO) programming relies on the interaction of different objects to achieve the programs goal.

Each object, in OO terms, may be composed of procedural code, and some procedural programs are broken up into modules, giving a sense that they are, in some way, object oriented. Hence, the difference is not so much a programming issue as a paradigm.

Historically, the first programs were entirely procedural. Mechanisms were introduced to allow the interpreter of the programs to move around in the code (jump), but there was no sense of independence in the code blocks.

## Subroutine

A subroutine is a piece of code executed upon demand, separate from the main flow of the program. The interpreter will jump to the code, execute it, and return to the main program.

Keywords such as GOSUB and RETURN, from the BASIC language, often used with line numbers, provided a programmer-friendly way to achieve this. The assembly language equivalents were less easy to digest, usually comprising a comparison, jump (or branch) and return, complete with offsets, memory addresses, or, in more modern assemblers, labels.

Subroutines are considered by many to be hard to maintain, difficult to read and digest, and are held in a similar light to the GOTO statement. In other words – reserved for use when nothing else will do.

It is worth noting that a modern language, implementing a modular structure, with named labels rather than line numbers, might be able to include subroutines in an acceptable fashion. However, with functions and procedures available, there may be no call for this approach at all.

Generically, the term *subroutine* can be used to denote a piece of code, separate from the main body, fulfilling a discrete task, in language-neutral terms. For example, a document might refer to the *file handling subroutines*.

## Procedure

A procedure is a named block of code, like a subroutine, but with some additional features. For example, it can accept parameters, which might be input, output, or pass-through.

Traditionally, a procedure returning a value has been called a function (see below), however, many modern languages dispense with the term procedure altogether, preferring to use the term *function* for all named code blocks.

Subsequently, the keyword PROCEDURE exists only in certain languages, and has disappeared from many. It is worth noting that languages like C do not use it, and that BASIC based languages do, whereas Modula and Pascal based languages have both the PROCEDURE and FUNCTION keywords, in which a FUNCTION is a PROCEDURE that returns a value.

## Function

As we saw above, a function is considered in some languages to be a procedure that returns a value. However, it is usually the term used to refer to any named code block. It is worth noting that C based languages use the function keyword exclusively.

Functions can accept parameters, return values, and are usually maintained separately from the main program code. Many programming languages have a special kind of function (in C, the *main* function) designated as the entry point to a program.

BASIC based languages have no such entry point, since they are entirely procedural in that execution begins at the top of the program, and continues until it is told to stop.

## Method

Put simply, a method is a function contained within an object. In object oriented terms, we always speak of objects and their methods and properties. Each method is an action that we can ask the object to perform on one or more of their properties.

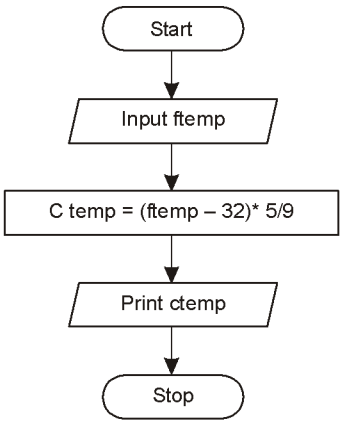
## Difference between Function and Procedure (Sub-routine)

Function by definition is a small executable code which can accept some values and returns a value as a result of the code. Most of the Third Generation Languages provides us a facility to create our own Functions.

Procedure or Sub-routine on the other hand is although a small independent code but can not return any value back to the calling program. A programmer needs to decide as to in which application a Function or a procedure has to be created.

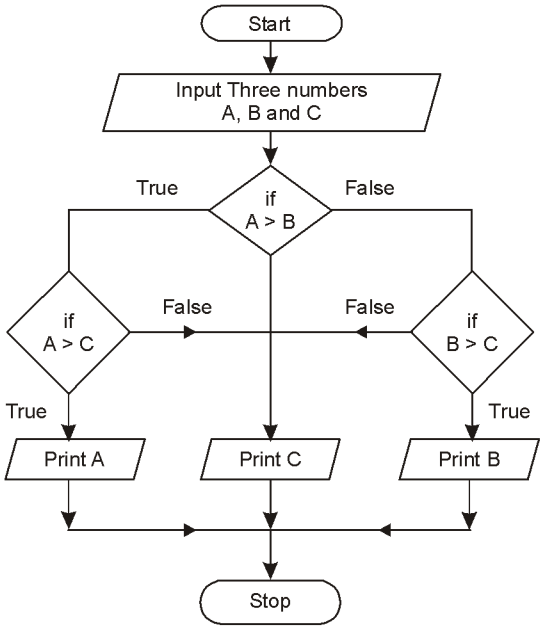
## Solved Examples of Flowcharts

**Example 1.** Draw a flow chart to convert Fahrenheit temperature to Centigrade temperature.
**Solution :** **Flowchart**

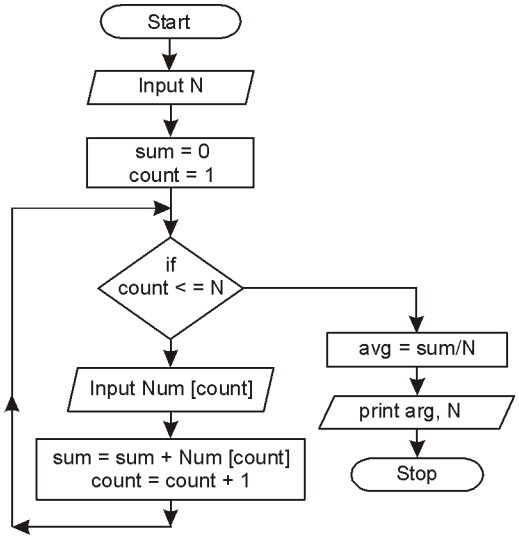



**Note :** In above flowchart
ctemp ~ Temperature in degree Celsius
ftem ~ Temperature in degree Fahrenheit

**Example 2.** Draw a flowchart to find and display largest among 3 numbers.
**Solution :** **Flowchart**

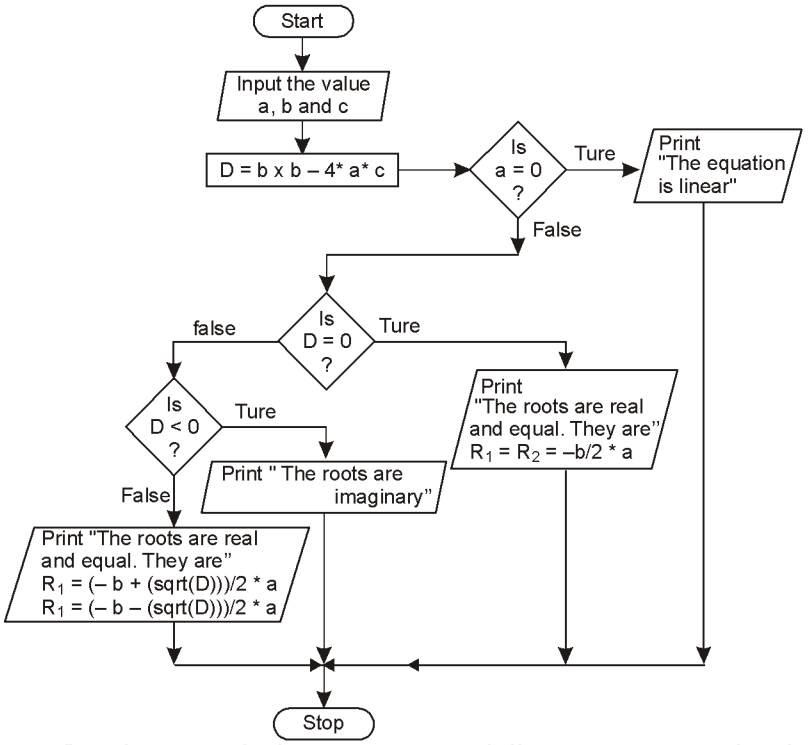



**Example 3.** Draw a flowchart to find average of N numbers and display average & numbers.
**Solution :** **Flowchart**



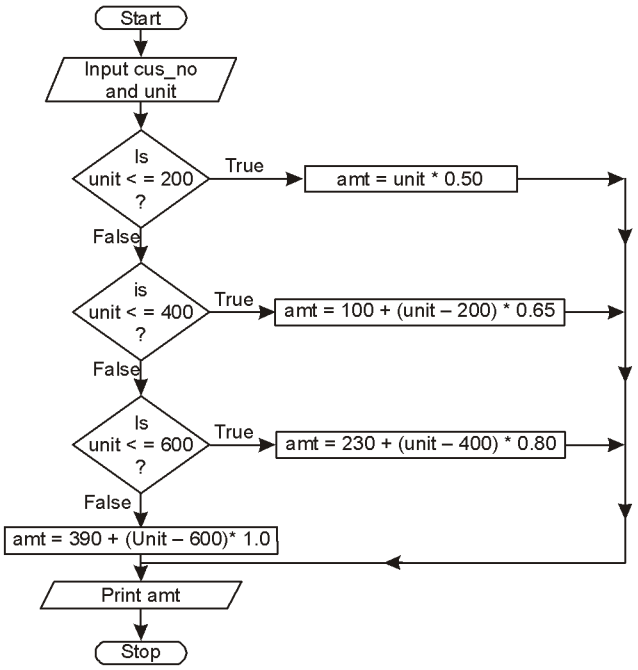

**Example 4.** Draw a flowchart to find the roots of quadratic equations.

**Solution :** **Flowchart**




**Example 5.** Draw a flowchart to calculate a customers bill amount acc. to the electricity company rules.

**Solution :** **Flowchart**


Start
Input cus_no and unit
Is unit < = 200 ?
True
amt = unit * 0.50
False
is unit < = 400 ?
True
amt = 100 + (unit – 200) * 0.65
False
Is unit < = 600 ?
True
amt = 230 + (unit – 400) * 0.80
False
amt = 390 + (Unit – 600)* 1.0
Print amt
Stop

# Basic of C

## Introduction – The C Language

'C' seems a strange name for a programming language. But this strange sounding language is one of the most popular computer languages today. C was an offspring of the 'Basic Combined Programming Language' (BCPL) called B, developed in the 1960's at Cambridge University. B language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972. The new language was named C. Since it was developed along with the UNIX operating system, it is strongly associated with UNIX. This operating system, which was also developed at Bell Laboratories, was coded almost entirely in C.

For many years, C was used mainly in academic environments, but eventually with the release of C compilers for commercial use and the increasing popularity of UNIX, it began to gain widespread-support among computer professionals. Today, C is running under a number of operating systems including MS-DOS. Since MS-DOS is a dominant operating system for microcomputers, it is natural that C has begun to influence the microcomputer community at large.

## Importance of C

The increasing popularity of C is probably due to its many desirable qualities. It is a robust language whose rich set of built in functions and operators can be used to write any complex program. The C compiler combines the capabilities of an assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages. In fact, many of the C compilers available in the market are written in C.

Programs written in C are efficient and fast. This is due to its variety of data types and powerful operators. It is many times faster than BASIC. For example, a program to increment a variable from 0 to 15000 takes about one second in C while it takes more than 50 seconds in an interpreter BASIC.

There are only 32 keywords but its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs.

C is highly portable. This means that C programs written for one computer can be run on another with little or no modification. Portability is important if we plan to use a new computer with a different operating system.

C language is well suited or structured programming, thus requiring the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. This modular structure makes program debugging, testing and maintenance easier.

Another important feature of C is its ability to extend itself. A C program is basically a collection of functions that are supported by the C library. We can continuously add our own functions to the C library. With the availability of a large number of functions, the programming task becomes simple.

## Before Starting C

Every computer program normally consists of three basic steps :

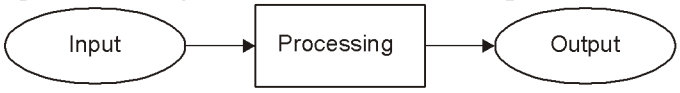



**Fig. 6.1**

For every program, there exists certain INPUT, which after some PROCESSING gives the user a certain OUTPUT. Readers are requested to verify the above statement with the software of their own choice. For any software of your choice you will be able to easily able to identify these three components of programming.

As every language, hardly matters its a general language or a programming language, is always bounded by some rules which" are to be followed. So as we are doing it with any other language, we

should first of all note down all the rules and regulations which are to be taken care while using this language.

In C, an instruction is a group of Alphabets, Numbers, Special symbols, Constants, Variables and keywords. We will discuss all the words from previous sentence one by one.

## Character Set for C

The language uses Alphabets (both Uppercase and Lowercase), digits (0-9) and some special symbols to represent information. The special characters or symbols allowed are :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | - | * | / | = | % | & | # |
| ! | ? | ^ | " | ' | \ | \| | < |
| > | ( | ) | [ | ] | { | } | : |
| ; | . | , | (blank space) | | | | |

Apart from the characters given above the language also uses some combination of characters to represent some special characters like \0 is used to denote a NEW LINE character. Such special combination characters are named as ESCAPE Sequences.

## Identifiers and Keywords

Identifiers are names given to various elements of a program such as variables, function etc.

Identifiers consist of letters and digits : any how the first character has to be an Alphabet.

There are certain words in C which carries a special meaning, these words are called KEYWORDS. A programmer cannot change the use of these keywords. The standard keywords are as follows :

| | | | |
|---|---|---|---|
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | far | for | goto |
| if | int | long | near |
| Register | return | short | signed |
| static | struct | switch | typedef |
| union | unsigned | void | while |

We will learn the use of these keywords as and when required.

## Rules for writing program in C

**(1)** C is a CASE sensitive language that means Abc, abc and ABC will be treated as three different elements of a program i.e. they are not same.

**(2)** Semi Colon (;) indicates the end of a statement.
a=5; b=10; c=20;

**(3)** This seems to be one statement, but in C the above are three different statements as there are semicolons in-between various elements.

**(4)** In C every program there must be at least one function and the name of that function is fixed, main( ). We can say that this is a compulsory function which should be present in every C program.

**(5)** Before using a variable we have to declare the variable so that compiler can allocate the memory space required by that variable.

## Variables and their Data Types

In the last point of previous section we used a word called VARIABLE. Variable by definition is name of a memory location which is used to read or write value at that memory location. These variables hold various values which are processed by the program.

In C, to use a variable we have to declare a variable so that the compiler is able to allocate space for that variable. Here a question arises :

How much space do you require for a variable ?

There are certain keywords in C which are used to indicate various data types, where each data type has its own memory size.

**Syntax for Declaring a Variable :**

**data type variable list;**

/* More than one variable must be separated by semicolons */

Data Types are a kind of identification for every variable, it specifies that the variable can store "what type of values and how much" (minimum and maximum range).

## Data Types

| | | |
|---|---|---|
| int | 2 bytes | Only stores Integer Values |
| char | 1 bytes | Allows to store only single character |
| float | 4 bytes | Allows to store floating point number (decimal/exponent based) |
| double | 8 bytes | Allows to store double-precision floating point number |

## Constants

As the name signifies they are values which cannot be changed at the execution-time of a program. C allows us to use four different types of constants.

**(a) Integer constants :** These are values which do not have any floating (decimal) points in it. We can write Integer constants in three different number systems which are : Decimal Number (Base 10), Octal Number (Base 8) and Hexadecimal Number (Base 16).

**(b) Unsigned and Long Integer Constants :** As the name signifies an Unsigned Integer will only contain integer values which are greater than or equal to zero. (No negative values are allowed)

Long Integer on the other hand can be considered to be Integers of Bigger size which requires more memory as compared to normal integer variables.

**(c) Floating Point Constants :** A floating-point constant will have base 10 and contains decimal point values or values in exponent notation or sometimes both.

**(d) Character Constants :** A character constant is a single character which is enclosed in apostrophes (' ') not in double quotes (" "). Following is the table which is showing various type of constants which are allowed in C language.

| Type of Constants | Example | | |
|---|---|---|---|
| Integer with Base 10 | 743 | 32767 | 9999 |
| Integer with Base 8 | 01 | 0123 | 0777 |
| Integer with Base 16 | 0x0 | 0xabcd | 0x9 |
| Unsigned Integer Constants | 5000U | 0123U | 0XFBAU |
| Long Integers | 123456L | 43L | 12343432L |
| Unsigned Long Integers | 1243UL | 3432UL | 0Xfb324ul |
| Floating Point constants | 0.0 | 432.43 | 1.456E+8 |
| Character Constants | 'X' | 'x' | '5' |

The following table contains various type of constants which may or may not be right according to the C language rules. Please read them carefully and find out the exact reasons if a constant is not allowed.

| Type of Constants | Example | | |
|---|---|---|---|
| Integer with Base 10 | X743 | 3276.7 | 123 |
| Integer with Base 8 | 123 | 0180 | 777 |

| Integer with Base 16 | 0XA | 0xyz | X0123 |
|---|---|---|---|
| Floating Point constants | 0.0 | 432.43 | 1.456E+8.5 |
| Character Constants | "A" | 65 | "Test" |

I hope after giving a close look to the above table you might conclude the following points which must be satisfied when using a Constant in C.

- An integer constant must start by a Number not with any other character like X.
- 0x and 0X are the same thing when we are using a Integer Constant with Base 16 (Hexadecimal notation).
- In an octal number we cannot use number other than (0 to 7) and must begin with 0.
- All the hexadecimal constants must begin with either 0x or 0X.
- For floating point notations the number followed by E must be an Integer Value, not a floating point expression.
- A Character type constant cannot have the values in Double Quotes, doesn't matter if the character in double quotes is only one.

## Before writing the first C Program

Before we start writing the first C program there are some more things which must be known to us so that we can comfortably start writing programs in C.

C is not a language rather it is a group of functions. Every C program must contain at least one function and name of that function is main( ).

Function by definition is a small code, which takes some input and after some processing may or may not return some value. Every function in C can be divided into three parts which are as follows :

(a) Function Declaration (Prototype Declaration)
(b) Function Definition (Actual Function Code)
(c) Function Call (Using a Function)

### Function Declaration

Tells the user as to how the function can be used. In other words, what all parameters are required and of what type. For built-in function the function declaration is there in the Header Files.

If we look at the directory of C language there must a folder named INCLUDE which contains all the header file and the extension of these header file is *.h

### Function Definition

Function Definition of the function is the actual code which is executed at the back of the program when we call a function. The function coding of built-in function is there in the Library Files (files in machine language) which are stored in the files with an extension of *.LIB. In your TurboC Directory these files are stored in a separate folder named LIB.

### Function Call

Function Call is the actual use of the function. To use any function the user ultimately has to make a function call which should satisfy the parameters or prototype specified in the related Header File.

**Functions in C Language can be divided into two parts :**
**(a) Library/Built-in Functions**
**(b) User Defined Functions**

It is too early to talk about the User Defined Functions but certainly we should know about the Library functions. To use the library files, before using a library function we have to include the header file in which the declaration can be found for the function which we are going to use.

**Syntax for using a Header File :**
**(a) #include < headerfilename.h>**
**or**
**(b) #include "headerfilename.h"**

There is a bit of difference between the two methods specified. In case of the first method we are specifying a header file which has to be picked from the default include directory which is specified in the Compiler options.

In case of the second method, the compiler will search for the specified file in the current folder or the folder specified with the file name in the double quotation marks.

**Some commonly used header file and their uses**

| Header File | Type of functions in the file |
|---|---|
| **stdio.h** | Standard Input and Output Header file, consisting functions useful for input and output. |
| **string.h** | File contains function to perform various operations on Strings. |
| **math.h** | Header file consisting most of the mathematical functions. |
| **conio.h** | File consisting functions related to the console. |
| **ctype.h** | Functions related to character manipulation are there in this header file. |
| **dos.h** | Header file containing various functions to perform various DOS related operations. |

## Elementary Functions

There are a couple of function which are very much common and necessary for every program , which is going to read values from keyboard or is going to print output on the console. The functions are described in detail :

**scanf :** Its a library function specified in STDIO.H and is used to input data from the standard input device. Syntax for the function is as follows :

scanf ( "control_string", address_list);

The function gets first parameter as a control_string which is a combination of various percentage characters where every percentage character contains a specific meaning. scanf function is going to read different type of values depending upon these percentage characters.

| % Character | Meaning |
|---|---|
| %c | Specifies that data item is a Single Character |
| %d | Specifies that data item is a Integer with base 10 |
| %e | Specifies that data item is a floating point value |
| %f | Specifies that data item is a floating point value |
| %g | Specifies that data item is a floating point value |
| %h | Specifies that data item is a short integer |
| %i | Specifies that data item is a decimal, hexadecimal or octal integer |
| %o | Specifies that data items is a octal integer |
| %x | Specifies that data items is a hexadecimal integer |
| %s | Data items is a string followed by blank space character |
| %u | Specifies an unsigned integer. |

So according to the first parameter the scanf function is going to read various values. The number of values to be read depends upon the number of percentage characters given in the control string of the function.

After reading the values from the standard input, its now required to store the values to some memory location so that these values can be reused by other part of the program. For this purpose we have to supply addresses of the variables.

## How to get address of a variable?

In the previous pages you came to know that we have to declare a variable before using it.

At the time of declaration of a variable the variable is allocated a specific unique memory from the primary memory. To get the address of the variable we can use & operator which when used as a prefix with any variable, will give the address of the variable.

**scanf("%d %d %d", &a, &h, &c);**

In the above statement, scanf function is going to read three integer values and will store them at the addresses of variables a, band c respectively. Here a, band c must be earlier declared as integer variables.

**scanf("%d %c %f", &a, &b, &c);**

In the above statement, scanf function will read one integer followed by one character and one floating point value and these values will be stored at the addresses of variables a, band c respectively.

**scanf("%d %f", &a);**

Here although the function is going to read two values, one integer and another float but will only store the integer value in variable a and the other will be lying in the Standard Input Buffer which may be used in the subsequent scanf function.

## Limiting the amount of values in scanf Statement

The normal value separator in C language is a white space which when occurs in input signifies separation of two values. Apart from this separator we can restrict our scanf statement to read a specific number of character or digits for a specific value. Here we can specify the maximum width of digits that will be used for any specific variable.

The value can have fewer characters than the specified field but cannot exceed the limit specified in the control string.

**Example :** Write a program to read a date in the format of "dd mm yyyy".

```
#include<stdio.h>
void main ( )
    {
        int dd, mm, yy;
        printf("Enter your date of birth : ");
        scanf("%2d %2d %4d",&dd,&mm,&yy);
        printf ("Your date of birth is %d-%d-%d", dd,mm,yy) ;
    }
```

**Execution of Program :**

**Input :** Enter your date of birth : 20 8 1976

**Output :** Your date of birth is 20-8-1976

**Input :** Enter your date of birth : 2081976

**Output :** Your date of birth is 20-81-976

**printf :** This function can be used to write the output data on the standard output device.

This function can be used to print any type of data which may include integer, floating point values, character or string type values. Syntax for the function is as follows :

**printf ( "control_string", variable_list);**

The function gets first parameter as a control_string which is a combination of various percentage character where every percentage character contains a specific meaning.

| % Character | Meaning |
|---|---|
| %c | Data item is shown as a single character. |
| %d | Data item is shows as a signed decimal integer. |
| %e | Shows as a floating point value with an exponent. |

| | |
|---|---|
| %f | Shows a floating point value without an exponent. |
| %g | Displays values using either e-type or f-type conversion is depending on the value. |
| %i | Display values as a signed decimal integer. |
| %o | Displays as an octal integer without any leading zeros. |
| %s | Displays a string type value. |
| %u | Displays value as an unsigned decimal integer. |
| %x | Displays as an hexadecimal integer values without a leading 0x |

Unlike, scanf function we do not have to specify the variable address in the printf function.

This function can access the values of the variables by just specifying the variable names of the variables.

## Operators

Whenever we are going to perform any operations then every expression will have a couple of things, first will be the operator which is actually the work that is going to be done and second will be operand or operands on which the operator is going to work.

The operands in C can be any constant values or it can be any variable which might contain some value of its own.

C operators can be classified into a number of categories. They include :

**(1)** Arithmetic operators.
**(2)** Relational operators.
**(3)** Logical operators.
**(4)** Assignment operators.
**(5)** Increment and decrement operators.
**(6)** Conditional operators.
**(7)** Bitwise operators.
**(8)** Special operators.

**(1) Arithmetic Operators :** C language provides five different types of arithmetic operators which can be used in any valid expressions. These operators fall in the category of Binary operators as they require two operands to work on.

| Operator | Purpose |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | modulus operator gives remainder after division (works only for integers) |

You might feel that in the above list there is one operator which is missing and that is obviously a symbol to be used for exponent for any value. Sorry, C language does not provides any such operator but for that there is a pow() function found in math.h header file of C language.

## Unary Operators

There are operators which are capable of working on single operand and such operators are called Unary operator. Normally these operators precede their single operand on which they are going to work on. The most commonly used unary operator is the minus sign which is used to get negative value.

Here it should be clear that this unary minus sign is entirely different from the sign which is given in the arithmetic operators. There it is used to subtract the second operand from the first where as here it is used to negate any value.

There are two other unary operators : ++ (increment operator) and - - (decrement operator) which is used to increase or decrease the value of the operand by one. These operands are covered in much greater detail in the following chapter.

## Integer Arithmetic

When both the operands in a single arithmetic expression such as a+b are integers, the expression is called an integer expression, and the operation is called integer arithmetic. Integer arithmetic always yields an integer value. The largest integer value depends on the machine, as pointed out earlier. In the above examples, if a and b are integers, then for a = 14 and b = 4 we have the following results :

a – b = 10
a + b = 18
a*b=56
a / b = 3 (decimal part truncated)
a % b = 2 (remainder of division)

During integer division, if both the operands are of the same sign, the result is truncated towards zero. If one of them is negative, the direction of truncation is implementation dependent. That is,

6/7 = 0 and -6/-7 =0
but -6/7 may be zero or -1. (Machine dependent)

Similarly, during modulo division, the sign of the result is always the sign of, the first operand (the dividend.) That is

-14 % 3 = -2
-14 % -3 = -2
14 % -3 = 2

## Real Arithmetic

An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result. If x, y, and z are floats, then we will have :

x = 6.0/7.0 = 0.857143
Y = 1.0/3.0 = 0.333333
z = -2.0/3.0 = -0.666667

The operator % cannot be used with real operands.

## Mixed-mode Arithmetic

When one of the operands is real and the other is integer, the expression is called a mixed mode arithmetic expression. If either operand is of the real type, then only the real operation is performed and the result is always a real number. Thus,

15/10.0 = 1.5

where as

15/10 = 1

**(2) Relational Operators :** We often compare two quantities, and depending on their relation, take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators. We have already used the symbol '<', meaning 'less than'. An expression such as

a < b or 1 < 20

containing a relational operator is termed as a relational expression. The value of a relational expression is either one or zero. It is one of the specified relation is true and zero if the relation is false. For example

10 < 20 is true
but
20 < 10 is false

C supports six relational operators in all. These operators and their meanings are shown in Table :

| Operators | Meaning |
|---|---|
| < | value of operand 1 is less than value of operand 2 |
| <= | value of operand 1 is less than or equal to value of operand 2 |
| > | value of operand1 is greater than value of operand 2 |
| >= | value of operand 1 is greater than or equal to value of |

| | operand 2 |
|---|---|
| = = | value of both the operands is equal |
| != | value of both the operands is not eaual. |

A simple relational expression contains only one relational operator and takes the following form :

**Operand1 relational operation Operand2**

**Examples :**

| | | |
|---|---|---|
| 4.5 <= | 10 | True |
| 4.5 <= | -10 | False |
| -35 >= | 0 | False |
| 10 < | 7+5 | True |
| a+b == | c+d | True |

(only if the sum value of a &b is equals to c & d)

**(3) Logical Operators :** In addition to the relational operators, C has the following three logical operators.

&& meaning logical AND
¦¦ meaning logical OR
! meaning logical NOT

The logical operators && and ¦¦ are used when we want to test more than one condition and make decisions. An example is :

a > b && x = = 10

An expression of this kind which combines two or more relational expressions is termed as a logical expression or a compound relational expression. Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table shown in Table. The logical expression given above is true only if a > b is true and x = = 10 is true. If either (or both) of them are false, the expression is false.

**(4) Assignment Operators :** Assignment operators are used to assign the result of an expression to a variable. We have seen the usual assignment operator, , = '. In addition, C has a set of 'shorthand' assignment operators of the form

v Op= exp;

Where v is a variable, exp is an expression and op is a C binary arithmetic operator. The operator op= is known as the shorthand assignment operator.

The assignment statement

**v op= exp;**

**is equivalent to**

**v = v op (exp);**

with v evaluated only once. Consider an example

x += y+1;

This is same as the statement

x = x + (y+1);

The shorthand operator + = means 'add y+ 1 to x' or 'increment x by y+ 1'. For y =1=2, the above statement becomes

X += 3;

and when this statement is executed, 3 is added to x. If the old value of x is, say 5, and then the new value of x is 8. Some of the commonly used shorthand assignment operators are illustrated in Table

| Statement with simple assignment operator | Statement with shorthand operator |
|---|---|
| a = a + 1 | a += 1 |
| a = a - 1 | a -= 1 |
| a = a * (n+ 1) | a *= n+1 |
| a = a / (n+1) | a /= n+1 |
| a=a%b | a %= b |

The use of shorthand assignment operators has three advantages :

1. What appears on the left- hand side need not be repeated and therefore it becomes easier to write :

2. The statement is more concise and easier to read.
3. The statement is more efficient.

**(5) Increment and Decrement operators :** C has two very useful operators not generally found in other languages. These are the increment and decrement operators :

++ and --

The operator + + adds 1 to the operand while - - subtracts 1. Both are unary operators and take the following form :

++m; or m++;
--m; or m--;

+ +m; is equivalent to m = m+ 1; (or m + = 1;)
--m; is equivalent to m = m-1; (or m -= 1;)

We use the increment and decrement statements in for and while loops extensively.

While + +m and m+ + mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement. Consider the following :

```
m= 5;
Y = ++m;
```

In this case, the value of y and m would be 6. Suppose, if we rewrite the above statements as

```
m= 5;
y = m++;
```

then, the value of y would be 5 and m would be 6. A prefix operator first adds 1to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

Similar is the case, when we use + + (or - -) in subscripted variables. That is, the statement

```
a[i++] = 10;
```

is equivalent to

```
a[i] = 10;
i = i+ 1;
```

The increment and decrement operators can be used in complex statements. Example :

```
m = n++ - j+10;
```

Old value of n is used in evaluating the expression. n is incremented after the evaluation. Some compilers require a space on either side of n++ or ++n.

**(6) Conditional Operator :** A ternary operator pair "?:" is available in C to construct conditional expressions of the form

**exp1? exp2 : exp3;**

where expl, exp2, and exp3 are expressions.

The operator 1 : works as follows : expl is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the expression. If expl is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expressions (either exp2 or exp3) is evaluated. For example, consider the following statements.

```
a = 10;
b = 15;
x = (a > b) ? a : b;
```

In this example, x will be assigned the value of b. This can be achieved using the if..else statements as follows :

```
If (a > b)
x = a;
else
x = b;
```

**(7) Bitwise Operators :** C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or shifting them right or left.

Bitwise operators may not be applied to float or double. Table lists the bitwise operators and their meanings.

| Operator | Meaning |
|---|---|
| & | bitwise AND |
| \| | bitwise OR |

| | |
|---|---|
| ^ | bitwise exclusive OR |
| << | shift left |
| >> | shift right |
| ~ | One's complement |

**(8) Special Operators :** C supports some special operators of interest such as comma operator, sizeof operator, pointer operators (& and *) and member selection operators (. and -». The comma and sizeof operators are discussed in this section. Two preprocessor operators known as "string-izing" and "token-pasting" operators (# and ##).

## The Comma Operator

The comma operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated left to right and the value of right-most expression is the value of the combined expression. For example, the statement

value = (x = 10, Y = 5, x+y);

first assigns the value 10 to x, then assigns 5 to y, and finally assigns 15 (i.e. 10+5) to value.

Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

Some applications of comma operator are :

In for loops :

for (n =1, m = 10; n <= m; n++, m++)

In while loops :

while(c = getchar(), c != '10')

Exchanging values :

t = x, x = y, y = t;

## The size of Operator

The size of is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier.

**Examples :**

m = sizeof(sum);
n = sizeof(long int) :
k = sizeof(235L);

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

## Associability of Operators

In C language various operators can be used as a mixture in a single expression. In that case it should be clear to the programmer as to which operator is going to work first and what will be direction of implementation. (It can be Left -> to -> Right or Right -> to -> Left)

### Hierarchy of operator precedence's :

| Operator Category | Operator | Associatively |
|---|---|---|
| Unary Operators | – ++ – ! size of (type) | Right -> Left |
| Arithmetic | * / % | Left -> Right |
| Arithmetic | + - | Left -> Right |
| Relational Operators | < <= > >= | Left -> Right |
| Equality Operators | == != | Left -> Right |
| Logical AND | && | Left -> Right |
| Logical OR | II | Left -> Right |