- describe the basic concepts of programming
- identify the structure of a C program
- describe the primary elements like identifiers, tokens, constants etc.
- know the basic data types used in C language
- declare and initialize C variables

1.2 INTRODUCTION

Computer programming is defined as telling a computer what to do through a special set of instructions which are then interpreted by the computer to perform some task(s). These instructions can be specified in one or more programming languages including (but not limited to) **Java**, **PHP**, C, and C++. A computer goes through a series of steps whose purpose is to achieve something - a series of steps that are instructed to it in great detail by computer programs. Essentialy, computer programming is the process by which these programs are designed and implemented. There are many advantages to learning the subject of computer programming such as gaining new skills, being able to tell the computer what to do, and becoming better acquainted with computers. So whether you are a computer hobbyist, a student, an IT professional, or are just curious about the subject, learning how to program a computer will be highly beneficial.

This unit is an introductory unit of C programming. This unit includes the types of programming language along with the programming techniques. Basic programming concepts including pseudo code, algorithm and flowcharts are briefly described here. In addition, you will come across of terms that are used in C like identifiers, keywords and constants etc. The elementary data types used in C and the concept of variables are also included as an important part of the unit.

1.3 DEFINING A PROGRAM

A computer program or simply a program is a set of instructions or codes which directs the computer to do some kind of specific functions. The codes or instructions are written using a programming language like FORTRAN, C, C++ etc. For a given problem a particular program or set of instructions can be design to solve it. The method of writing the instructions is called the programming. The person who writes such instruction is called a programmer. An another way of defining a program is : *a computer program is nothing but a combination of algorithm and data structure combined into a single unit which is designed to solve a given problem.*

To get the computer solution of a problem first the problem should be transformed into a computer program. Every program takes input data and manipulates it according to the instructions written in the program and finally produces the output which represents the computer solution to that particular problem.
Some characteristics of a good program are : *accurate, efficiency, reliability, portability, robustness* etc.

Accurate means - the problem that has to be solved by computer must be predefined clearly with specification of requirements. Based on the requirements the programmer designs the program so that it must be accurate to perform the specified task.

Efficiency means - a program should be such that it utilizes the resources of a computer system (e.g.. memory, CPU time etc.) in an efficient manner.

Reliability means - a program should be work its intended function accurately even if there are temporary or permanent changes in the computer system. Program having such ability are known as reliable.

Portability means - a program prepared on a certain type of computer system should run on different types of computer system. If a program can be transferred from one system to another system or one platform to another platform with ease and still it works properly then it is called portable.

Robustness means - a program is expected to continue with its functionalities even at the unexpected errors. It means a program is called robust if it provides meaningful results for all inputs (correct or incorrect). For correct data it will provide desired results and for incorrect data it will give appropriate message with no run time errors.

1.3.1 Program Development Cycles

The process of developing a program comprises of many steps. Before going to coding a program in a particular language, the programmer has to determine the problem that need to be solved. The program development cycle which is consists of six different steps is described in the following where each of the steps has its own significance.

Problem definition : This is the very important phase where the problem is analysed precisely and completely. The problem is defined with respect to the needs of the user it means that all the possible views of the users are collected and defined clearly. The ultimate final quality of the program mainly depends on this stage of program development. If this step is erroneous then it will result in poor quality of final product. The major components of program like input, processing logic, output etc. are clearly defined in this stage.

Analysis : This phase involves the identifying the appropriate algorithm and data structure based on the problem definition done on the above stage. The need of data structure depends on the nature of input, processing and desired output. Proper names for the identifiers can be given at this step of program development.
You will know the meaning of the terms *algorithm, identifier* etc. in the next section of this unit.

Design Phase : After selecting the appropriate solution, algorithm is developed to depict the basic logic of the selected solution. An algorithm depicts the solution in logical steps which is nothing but a sequence of instructions. An algorithm can be represented by flowcharts and pseudocodes. These tools make the program logic clear and helped in coding the problem.

Testing the Algorithm for accuracy : Before converting the algorithms into actual code it should be checked for accuracy. The main purpose of checking the algorithm is to identify the major logical errors that may occur at any steps in the algorithm. Logical errors are often difficult to detect and correct at later stage so it will be helpful if the logical errors can be eliminated in this step.

Coding : The actual coding of the problem is done at this step by using a suitable programming language.

Testing and Debugging : At this step of program development, it is expected to test the complete program. The initial program codes may have errors. Generally, a program known as compiler check the code for syntax errors as well as finally produce the executable format of the whole code. Thus, the results obtained are compared with the expected results. Depending upon the complexity of the program, several round of testing may be required.

Implementation and Documentation : In this step of program development the program that is tested locally is taken to the user's place and installed there. This step will be absent if the program is small and simple. Moreover, the ***manuals or instruction booklet*** of the program is also prepared. The manual helps the user in understanding the program. The documentation of the program contains system flowchart to explain the complete functionality of the program at a glance.

Maintenance : During the operation of the program the users may come across many points where they need some changes in the program. Such types of change of a working program is belongs to the maintenance part of program development. Moreover, proper backup and restore methods also falls into the maintenance part.

The following figure shows the program development cycle :

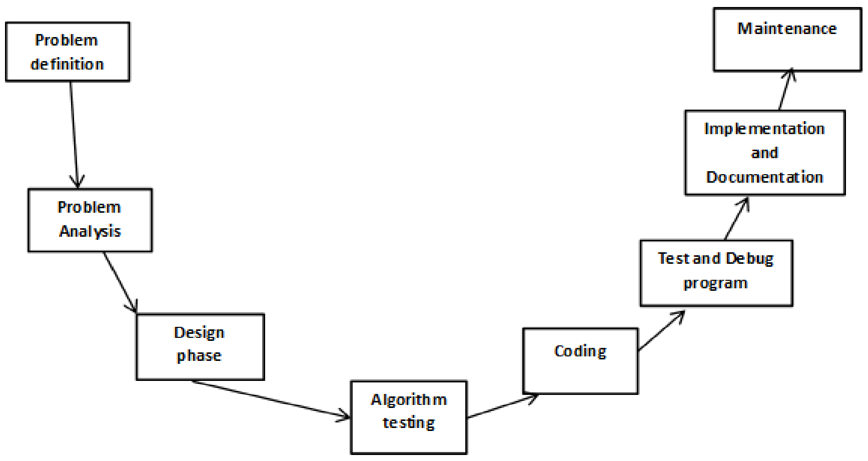


Fig. 1.1 Program development cycles

CHECK YOUR PROGRESS

1. State True or False

a) Computer program is a combination of data structure and algorithm.

b) An accurate program means it is platform independent.

c) In program development cycle appropriate algorithm and data structure is identified in coding phase.

d) Logical errors of a program is identified during algorithm testing phase.

e) The semantic and syntax errors in a program is checked in testing & debugging phase.

1.4 TYPES OF PROGRAMMING LANGUAGE

The programming language is the medium through which the problems to be solved by computer can be represented in the form of programs or set of instructions. The computer executes the set of instructions written in a particular language to produce the desired result. A programming language consists of a set of characters, symbols, and usage rules that allow the user to communicate with computers. In a simple form it can be defined as - *any notation for the description of algorithm and data structure may be termed as a pro-*

gramming language.

The computer understand only the binary language i.e. the languages of 0 and 1, which is also called machine language. In the initial years of computer programming the computer programs were written using machine language which were too difficult to remember all the instructions in the form of 0 and 1s. But with due course of time the other languages were evolved along with the generation of computers. There are different levels of programming language as follows :

- Machine language
- Assembly language
- High level language
- 4GL language

The different levels of programming language is shown in the figure below :

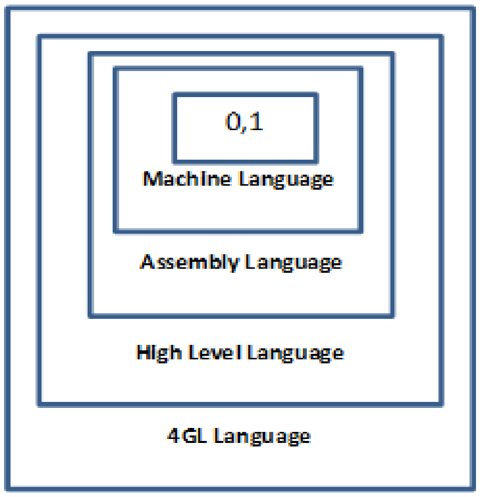


Fig. Levels of Programming Language

i) **Machine language :** The machine language programs were written using the digit 0 and 1. The instruction in machine language consists of two parts. The first part is an operation which tells the computer what operation is to be performed. The second part of the instruction is operand, which tells the computer where to find or store the data after applying the operation.

Advantage :

Translation free **:** Machine language is directly understand by the computer so it does not need any translation. Otherwise, if a program is developed in high level language then it has to be translated into the machine language so that the computer can understand the instructions.

High speed : The application developed in the machine language are extremely fast because it does not require any conversion.

Disadvantage :

Machine dependent : Program developed in one computer system may not run on an another computer system.

Complex language : This language was very difficult to understand and programming techniques were tedious.

Error prone : Writing machine language programs, a programmer has to remember all the opcodes and memory locations, so machine language is bound to be error prone.

ii) Assembly language : Assembly language is not a single language, but a group of language. An assembly language provides a mnemonic instruction, usually three letters long, corresponding to each machine instruction. The letters are usually abbreviated indicating what the instruction does, like ADD for addition, MUL for multiplication etc. The assembly language allows the programmer to interact directly with the hardware. Each line of assembly language program consists of four columns called fields. The general format of an assembly language instruction is :

	[Label]	**<Opcode>**	**<Operands>**	**[; Comment]**
e.g.	**BEGIN**	**ADD**	**A, B**	**; Add B to A**

Assembler is a software tool that is used to convert the assembly language program into a machine language program.

Advantage :

Easy to understand and use : Assembly language uses mnemonics instead of using numerical opcodes and memory locations used in machine language, so it is easy to understand and use.

Less error prone : Assembly language is less error prone and it provides better facility to locate errors and correct them.

Faster : It is much faster and use less memory resources than the high level language.

Disadvantage :

- Assembly language programs are machine dependent.
- Assembly language is complex to learn.
- A program written in assembly language is less efficient than machine language because every assembly instruction has to be converted into machine language.

iii) High level language : COBOL, FORTRAN, PASCAL and C are the examples of high level languages. High level languages are similar to English language. Programs written using these languages may be machine independent. A single instruction of a high level language can substitute many instructions of machine language and assembly language. Using the high level language complex software can be design.

Advantages :

- *Readability :* Since high level languages are similar to English language so they are easy to learn and understand.
- Programs written in high level languages are easy to modify and maintain.
- High level language programs are machine independent.
- Programs written in high level language are easy to error checking. Due to the concept of abstraction used in such type of language the programmers don't have to mind on the hardware level representation of programs.

Disadvantages :

Programs written in high level language has to be compile first for execution of the program. This step of compilation increases the execution time of an application. Moreover, the programs occupies more memory space during the execution time.

Another main draw back of HLL programs is - its poor control on the hardwares.

Fourth generation language : 4G languages has the special characteristics that they are closer to human languages so they are easy to learn. 4GL languages are specially designed for some specific applications with limited set of functions, so they are easy to learn. Oracle, VB, SQL, VC++ etc are the example of 4GL languages. Most of the database accessed languages are 4GL language.

1.4.1 Language Translators

The language translators or also called language processors are the programs which converts the high level language programs into the machine language programs for execution. We know that the computer understand only the machine language, so for execution of a program written in HLL must be converted into the machine language codes. The examples of language translators are - compiler, interpreter, assembler etc. Let us see them very briefly.

Compiler : A compiler is a program that can translates an higher level language program to its machine form. This language processor translates the complete source program as a whole into machine code before execution. Here the source program means - the program that is written by the programmer and the translated program is called a object program or object code. Examples: C and C++ compilers. If there are errors in the source code, the compiler identifies the errors at the end of compilation. Such errors must be removed to enable the compiler to compile the source code successfully. The object program generated after compilation can be executed a number of times without translating it again.

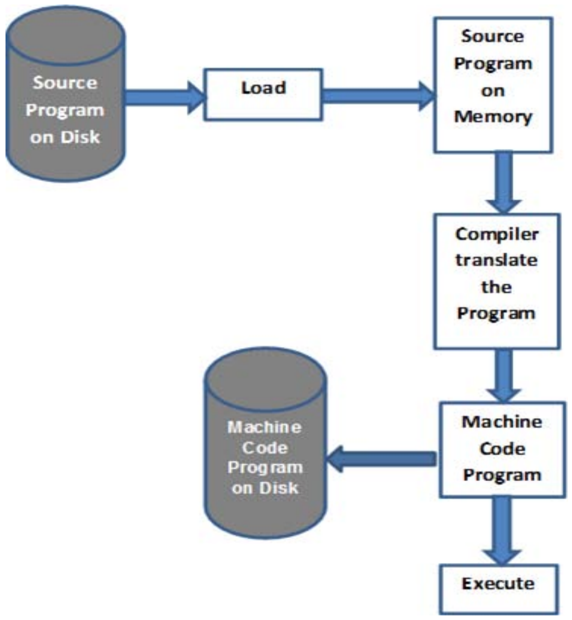


Fig. 1.3 A compiler translate a program

Interpreter : It is a language processor whose working principle is different from compilers. It translates each statement of source program into machine code and executes it immediately before translating the next statement. If there is an error in the statement the interpreter terminates its translating process at that statement and displays the error message. The GWBASIC is an example of interpreter.

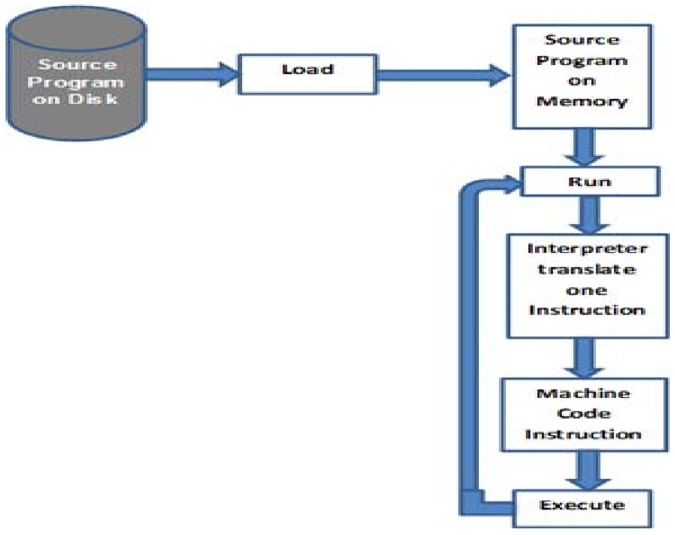


Fig.1.4 An Interpreter translate a program

Assembler : We have already known that - *assembly language* is the symbolic representation of a computer's binary encoding - **machine language** . Assembly language is more readable than machine language because it uses symbols instead of bits.

A program called an **assembler** that translates assembly language into binary instructions or machine language.An assembler reads a single assembly language *source file* and produces an *object file* containing machine instructions.

Linker : A linker is a program that attached or linked several modules of a program. Generally, a program written for a particular application may be consists of thousands or more lines of codes. In such a case, the large program can be broken into a number of smaller program which is called a *module*. Each module will perform its specific task and compiled them separately. Finally each module have to be linked together to construct the complete application. A linker is a program that links a number of modules (object codes, which are generated after the compilation step) and library functions to form a single executable program.

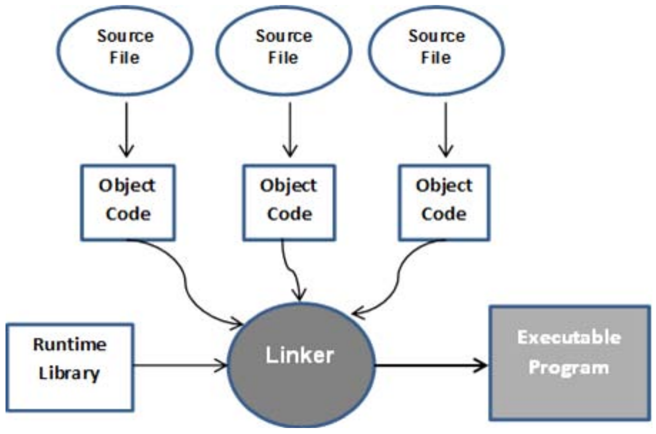


Fig. 1.5 Working of a Linker

Loaders : A loader is a program and that is also a part of the operating system and whose main function is to bring an executable program residing on the disk into main memory and start running it. Generally, a loader performs the function of a linker program also.

The loader performs the following tasks :

Allocation	: It allocates memory space for programs
Linking	: It combines two or more modules and supplies the information needed to allow reference between them.
Relocation	: It prepares a program to execute properly from its storage area.
Loading	: It loads the program into memory.

There are two types of loaders depending on the way the loading is performed : *absolute loader and relocating loaders*.
The absolute loader load the executable code into memory locations specified in the object module. On the other hand, relocating loader loads the object code into memory locations which are decided at load time. The relocating loader can load the object code at any location in memory.

CHECK YOUR PROGRESS

2. State True or False

a) Assembly language instructions are consisting with 0 & 1.

b) High level languages are machine independent.

c) Query languages are fourth generation language.

d) Interpreters generate a complete executable code.

e) Linkers are responsible for moving a program from disk into main memory.

3. What is a compiler? What do you meant by the compilation process?

4. What is a source program and an object program ?

1.5 BASIC TOOLS OF PROGRAMMING

So far, we have came to know that programming is a technique of solving problems using program codes. The program codes are designed by following specific rules and using some programming tools. The general tools used for designing programs are : pseudo code, algorithms, flowcharts etc. In this section, we will discuss briefly about those programming tools.

1.5.1 Pseudo Code

Pseudo code derived from '*pseudo*' which means imitation and '*code*' means instruction (pronounced as soo-doh-kohd) is a generic way of describing an algorithm without using any specific programming language related notations. Pseudocode is an artificial and informal language that helps programmers to develop algorithms. It is a "text-based" detail (algorithmic) design tool. Pseudo code generally consists of short English phrases to express a specific task. But, interestingly, there are no specific rules to write pseudo code. Then why do we need pseudo code? Implementing a problem with proper symbol and syntax is a complicated job. To solve the complicated job, first we need the complete explanation of the job, and then we extract information from the explanation, and roughly design the solution, which will be relevant to a programming construct. This rough work will help in designing the proper solution with proper symbol and syntax.

The general guidelines for developing pseudocodes are :

- Statements should be written in English and should be programming language independent.
- Steps of the pseudocodes must be understandable.
- It should be concise.
- Each instruction should be written in a separate list and each statement in pseudocode should express just one action for the computer.
- The keywords like READ, PRINT etc should be write in capital letter.
- Each set of instruction should written from top to bottom, with only one entry and one exit.

Advantage :

Its language independent nature helps the programmer to express the design in plain natural language.

Based on the logic of a problem it can be designed without concerning the syntax or any rule.

It can be easily translated into any programming language.

It is compact in nature and can be easily modify.

Limitations :

It is unable to provide the visual presentation of the program logic.

It has not any standard format or syntax of writing.

It cannot be compiled or executed.

The following is an example of pseudocode :

```
1. If student's grade is greater than or equal to 60
              Print "passed"
       else
              Print "failed"
```

1.5.2 Algorithm

Algorithm is a sequential, finite, non-complex, step-by-step solution to a problem written in English like statement. It can also be defined as - *an algorithm consists of a set of explicit and unambiguous finite steps which, when carried out for a given set of initial conditions, produce the corresponding output and terminate in a finite time.*

The writing of an algorithm follows the algorithmic notations. The algorithmic notations are the symbolic representation of the steps of the algorithms.

Problem: Design an algorithm to add two integers and display the result.

Solution: First of all you need two integers. There are two ways to get the two integers. First, you can supply the two integers from your side; secondly, you can ask the user to supply the integers. After getting the integers you store it into two containers. You need another container to store the sum of the two integers. You must initialize the third container as zero before you store the summation. Now, perform the addition task and store the result into the third container.

Now, display the content of the third container. This is the solution to your problem.

```
Step 1:  Ask the user to supply the first integer.
Step 2:  Store the first integer into container A.
Step 3:  Ask the user to supply the second integer.
Step 4:  Store the second integer into container B
Step 5:  Initialize container C as zero.
Step 6:  Add the content of container A and B and store the
         result into container C.
Step 7:  Display the content of the container C.
Step 8:  End.
```

The above example can be expressed in a different way :

```
Step 1 : Read a, b, c
         [a, b, c are integers]
Step 2 : c ← 0
Step 3 : c ← a+b
Step 4 : Print c
Step 5 : Stop
```

Here, in the above expression, we have input the numbers in a single line instead of three different line. We have used here a comment line within the **[ ]** brackets. Again we have assign the value 0 to **c**. We have add the values **a** & **b** and store it in **c**. Finally we display the value of **c**.

Always remember :

i) Usually word *Read, Accept or Input* can be used to represent input operation to give values of variables to the computer.
ii) Similarly for the output operation the words Print, Write or Display can be used.
iii) The symbol ← is used in case of assignment (i.e. =) operation. It means the value obtained by evaluating the right side expression is stored to the left side variable.
iv) The comments are non-executable statements written within pair

of square brackets and they are generally used to explain a particular step.

v) The steps are written sequentially and if two statements are written in a single line then they have to separate by semicolon ‘**;**’.

For example :

```
sum←0; a←25
sum←sum+a
Print sum
```

vi) In case of branching or conditional steps - ***If-Then*** or ***If-Then:Else*** is used. The conditional steps are consisting with the relational operators like - <, >, <=, >= etc.

For example :

```
If condition Then :            If condition Then
    Step 1                         Step i
    Step 2                     Else
    ....                           Step j
    Step n                     [ End of If ]
    [ End of If ]
```

vii) The iterative or repetitive steps can be write between **Repeat For** and **[End of For]** as shown below :

```
Repeat For I=1,2,3,....N           I←N
    Step 1                         Repeat While I>0
    ....                 or            Step 1
    ....                               ....
[ End of For I ]                       I←I-1
                                   [ End of While ]
```

For example,

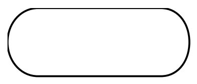
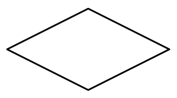
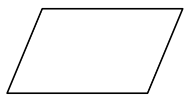
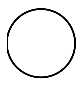
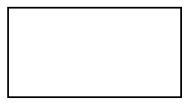
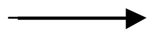
```
Fact←1
Repeat For I=1,2,3,...N
    Fact←Fact*I
[ End of For I ]
Print Fact
```

The following algorithm find the biggest of given three numbers :

```
Step 1 : Read a, b, c
Step 2 : big←a
Step 3 : If b>big Then
            big←b
Step 4 : If c>big Then
            big←c
Step 5 : Print big
Step 6 : Stop
```

1.5.3 Flow Chart

A ***flowchart*** is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instructions. A flowchart acts as a road map for a programmer and guides the programmer as to how to go from the starting point to the final point while writing a computer program. A flow chart will also helps the programmer at the time of analyzing, debugging and testing the programs. The main disadvantage of flowchart is that it is very time consuming and laborious to draw with proper symbol, especially for large complex program. The following symbols are generally used to draw a flowcharts :

Symbol	InstructionType	Symbol	Instruction Type
	Start/Stop		Decision
	Input/Output		Connector
	Process		Flow of Data

The general rules for drawing a flowchart is given below :

a) The flowchart should contain one Start and one Stop terminator.

b) The symbols of the flowcharts are always labeled with simple codes.

c) The codes used in flow chart should not give more than one meaning.

d) The control flows are always represented by arrows.

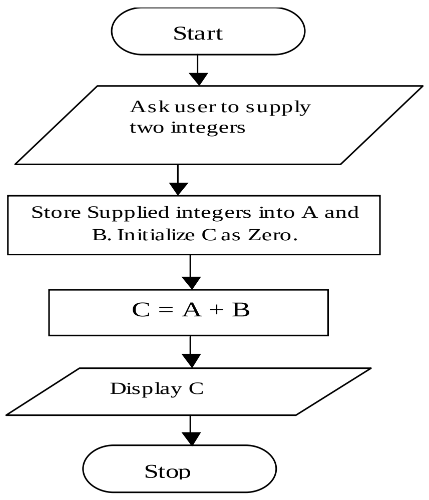
e) The control flow lines i.e. arrows should not cross each other.

f) The arrows moves from either vertically (top to bottom vice versa) or horizontally (left to right vice versa).

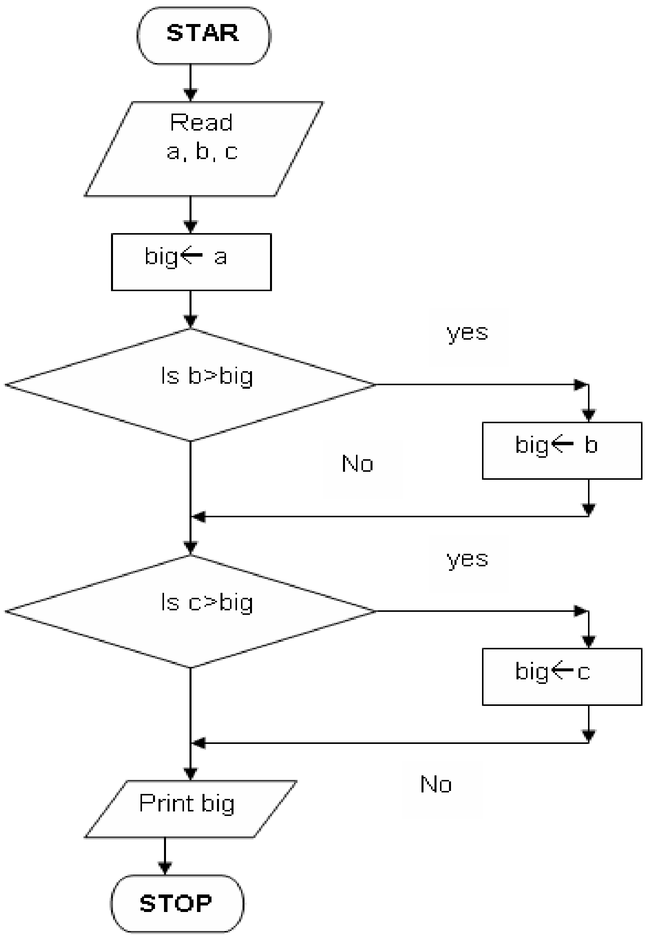
g) Only one flow line comes out from all the boxes or symbols except the decision box.

h) Two lines can flow from the decision b ox if single condition is checked. It means a single condition results in one of the two values TRUE or FALSE.

If we try to draw the flowchart of the algorithm we have written for finding the addition of two integers and display the result, then it will look as shown below :


Start
Ask user to supply two integers
Store Supplied integers into A and B. Initialize C as Zero.
C = A + B
Display C
Stop

The following flowchart is for the program for finding the greatest number from given three numbers.



CHECK YOUR PROGRESS

5. In an examination 50 students obtain marks in their 10 subjects. Draw a flowchart for calculating the average percentage marks of 50 students. The flowchart should show the counting of the number of students that appeared in the examination and the calculation should stop when the number of counts reaches number 50.

6. Write an algorithm to generate a Fibonacci sequence.

1.6 PROGRAMMING TECHNIQUES

The programming techniques refers to the design and writing of programs to solve a problem or task. During the problem definition the programming techniques can be clearly understand. The following are the techniques used in programming :

- Top-down approach
- Bottom-up approach
- Unstructured technique
- Structured technique
- Modular programming

Top down approach : In this approach, the given problem is divided into two or more sub problems, each of which resembles the original problem. The solution of each sub problem is taken out independently. Finally, the solution of all sub problems is combined to obtain the solution of the main problem. The following figure shows the meaning of top down approach.

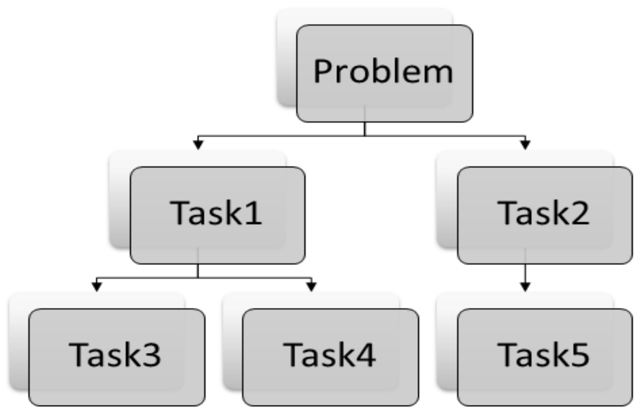


Fig. 1.6 Top-down approach

Bottom-up approach : This technique is just reverse of the top down programming. In this programming technique, the solutions of the independent sub-problems are designed first. Then these solutions are combined or composed in a main module in order to design the final solution of the problem. The following figure shows the bottom-up approach :

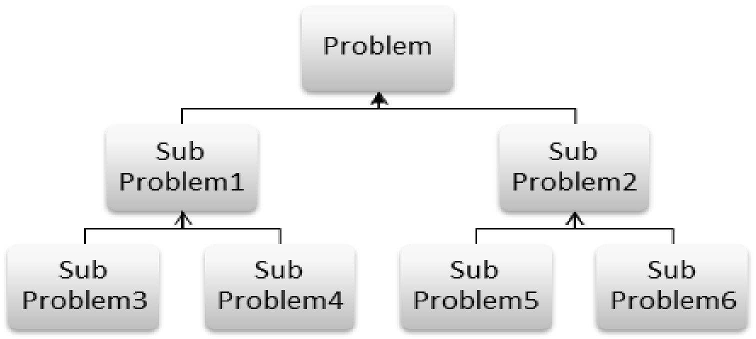


Fig. 1.7 Bottom-up approach

Unstructured Technique : This designing technique is refers to writing small programs using series of statements and having only the main program. All actions such as providing input, processing and output are done within one program only.The program design as well as the logic of the program is simple. To branch from one point to another point within the program is achieved by ***goto*** statement. There are a lots of difficulties seen in this type of technique. Testing and redesigning of the programs take more time.

Structured Technique : In structured technique, a program is broken down into small independent task that are small enough to be understood easily, without having to understand the whole program at once. Each task has its own functionality and perform specific part of the actual processing. Each task is again decomposed into subtask if necessary. The programs designed using the structured technique are well organized. In such type of technique, program design is simple and testing and debugging is easy because of the well defined control structures. The followings are some of the reasons for preferring structured programming :

a. It is easier to write a structured program - Complex programming problems or program are broken into a number of smaller, simpler tasks. Every task can be assigned to a different programmer and/or function.

b. It's easier to debug a structured program - If the program has a bug , a structured design makes it easier to isolate the problem to a specific section of code.

c. Reusability - Repeated tasks or routines can be accomplished using functions. This can overcome the redundancy of code writing, for same tasks or routines, it can be reused, no need to rewrite the code, or at least only need a little modification.

The three basic building blocks for writing structured programming are given below :

a) Sequence structure

b) Loop or Iterations

c) Binary decision structure

Modular Programming :

The technique of breaking down a large problem into a number of smaller program units known as ***module,*** is called a modular programming approach. Each module is designed to perform a specific function. There is one entry and one exit point for each module. Each individual modules can be easily tested and debugged. In case of modular programming, program maintenance is easy as the module showing errors can be easily detected and corrected. A large problem can be easily monitored and controlled using this technique.

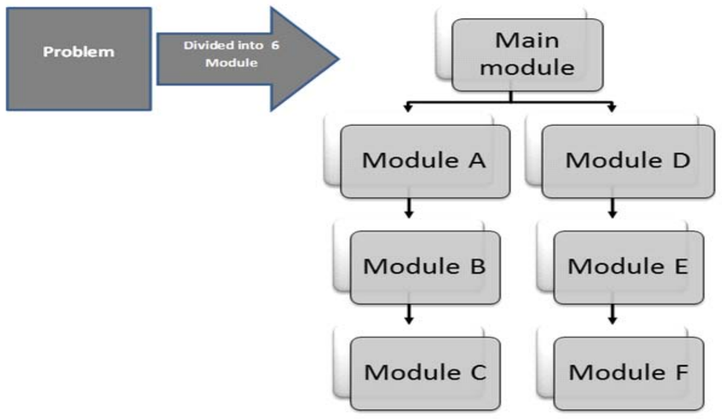


Fig. 1.8 Modular technique

1.7 THE C LANGUAGE

The programming language C is a general purpose computer language. It is a structured, high-level and machine independent language. It was originally created for the specific purpose of writing Operating System software.

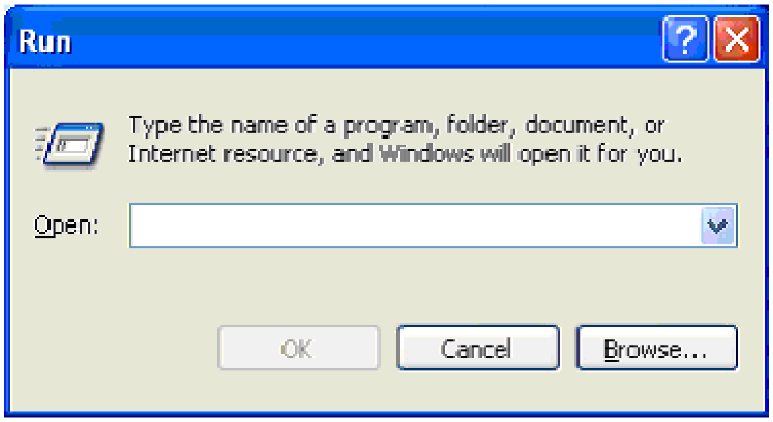
The development in C has seen many evolutionary processes. Like any other programming language, the original version of the C language has undergone a number of revisions. A lot of new features have been added to make it more useful and powerful. C was evolved from ALGOL, BCPL and B by ***Dennis Ritchie*** at the *Bell Laboratories* in 1972. C uses many concepts from these languages and added the concepts of data types and other powerful features. Since it was developed along with the UNIX operating system, it is strongly associated with UNIX. For many years , C was used mainly in academic environments, but eventually with the release of many C compilers for commercial use and the increasing popularity of UNIX, it began to gain widespread support among the computer professionals. Today C is running under a variety of operating systems and hardware platforms. During 1970s, C had evolved into what is now known as "traditional C". The language became more popular after the publication of the book 'The C Programming Language' by **Brain Kerningham and Dennis Ritchie** in 1978. The book was so popular that the language came to be known as "K&R **C**" among the programming community.

To assure that the C language remains standard, in 1983, *American National Standards Institute* (ANSI) appointed a technical committee to define a standard for C. The committee approved a version of C in December 1989 which is now known as ***ANSI C***. It was then approved by the *International Standards Organization* (ISO) in 1990.

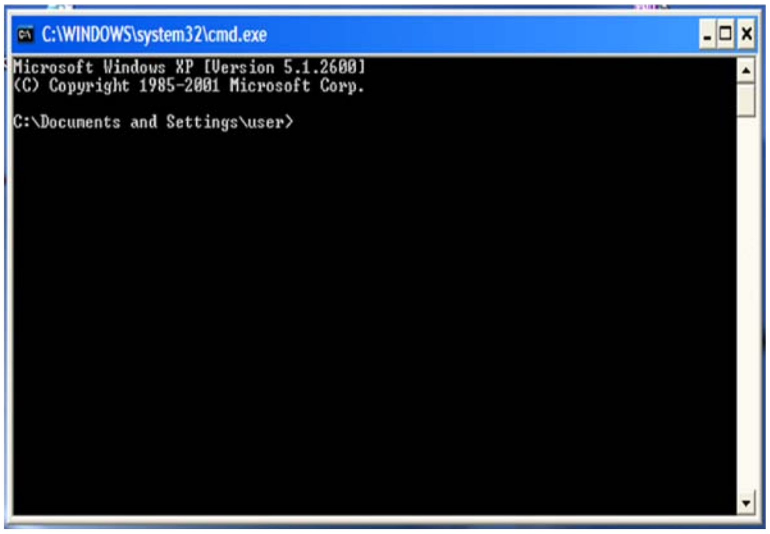
1.8 GETTING STARTED WITH C

The best way to get started with is to actually look at the program. So, let us first create a file "first.c". To create first.c file, we have to follow the following steps:–:

Step 1: Open Turbo C++ Editor from the icon on your desktop. If it is not in the desktop then go to Start and click on Run command from the Pop up Menu. The following command window will appear.

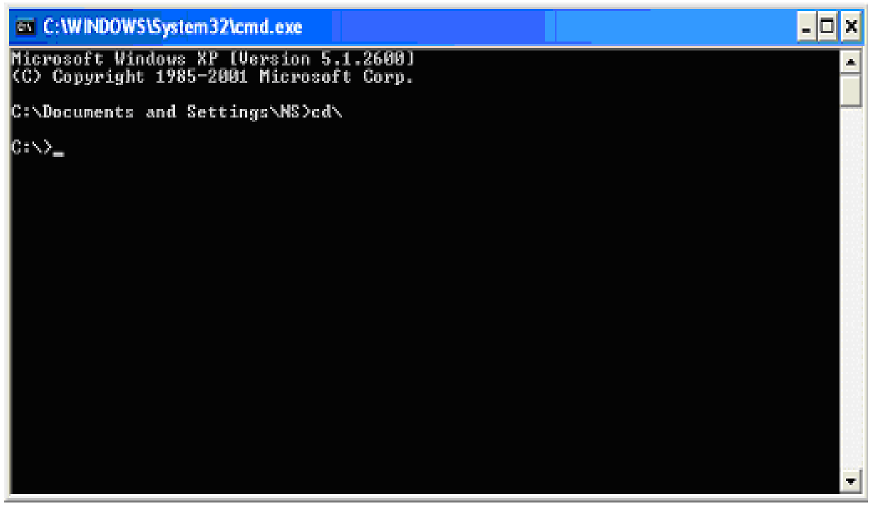

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
OK
Cancel
Browse...

In the open field type cmd and press enter key. The command prompt will appear as an active window.



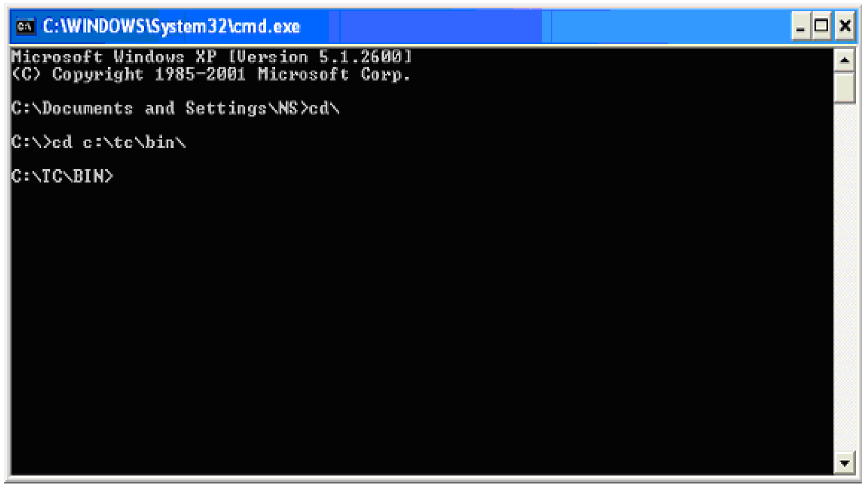
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\user>
```

Now type cd\ . Now file pointer is in root Directory that is in C:\



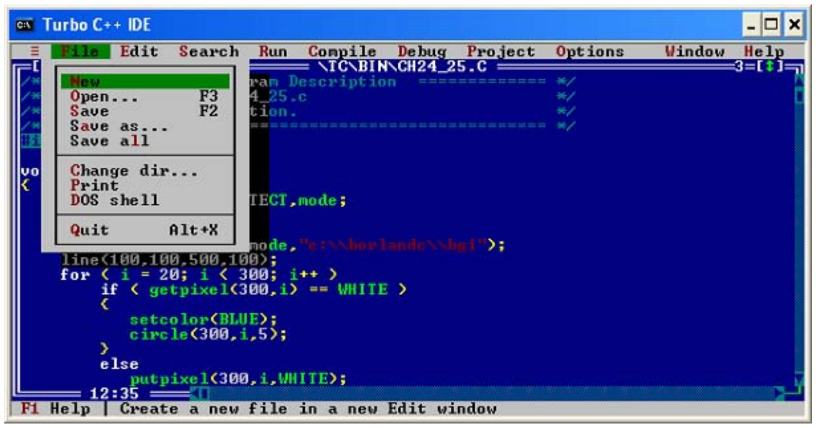
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\NS>cd\

C:\>_
```

Now type the path for Turbo C++ Editor that is cd c:\tc\bin and press enter. The following window will appear.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\NS>cd\

C:\>cd c:\tc\bin\

C:\TC\BIN>
```

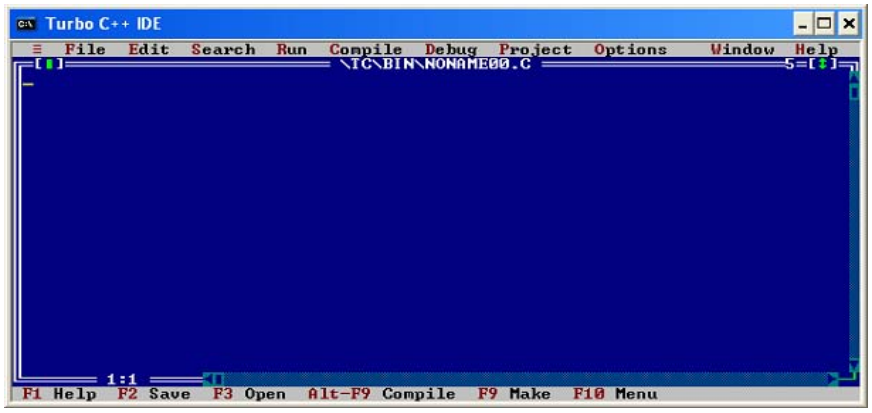
Now Type **TC** and press **Enter**. The Turbo C++ Editor will open. Go to **File** Menu and



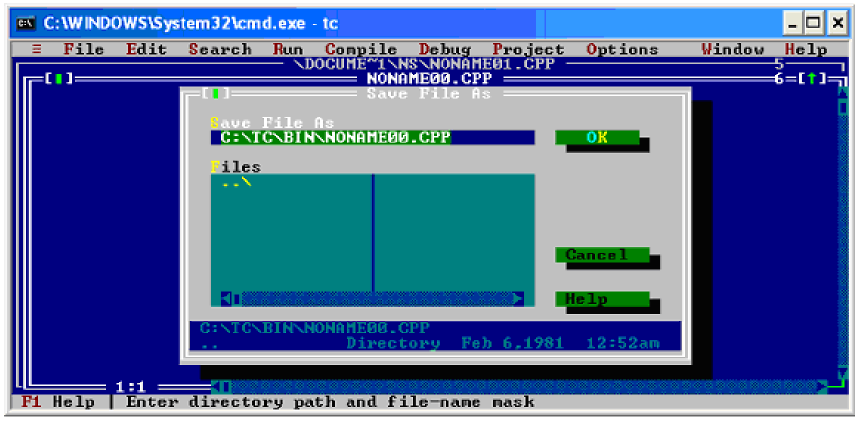
```
ram Description   ============== */
4_25.c                           */
tion.                            */
============================== */

TECT,mode;

mode,"c:\\borlandc\\bgi");
line(100,100,500,100);
for ( i = 20; i < 300; i++ )
     if ( getpixel(300,i) == WHITE )
     {
        setcolor(BLUE);
        circle(300,i,5);
     }
     else
        putpixel(300,i,WHITE);
```

F1 Help | Create a new file in a new Edit window

Step 2: After clicking the New submenu the following window will appear.

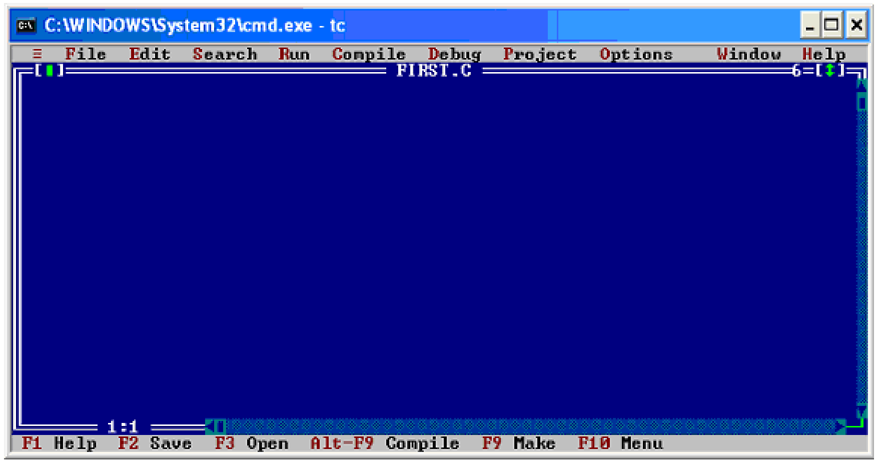

Turbo C++ IDE
≡ File Edit Search Run Compile Debug Project Options Window Help
\TC\BIN\NONAME00.C
5
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

Step 3: Now click on save submenu of file Menu.


C:\WINDOWS\System32\cmd.exe - tc
≡ File Edit Search Run Compile Debug Project Options Window Help
\DOCUME~1\NS\NONAME01.CPP
NONAME00.CPP
Save File As
OK
Cancel
Help
1:1

Save File As
C:\TC\BIN\NONAME00.CPP
Files
..\
C:\TC\BIN\NONAME00.CPP
.. Directory Feb 6,1981 12:52am
F1 Help | Enter directory path and file-name mask

Step 4: Type the file name that is c:\tc\bin\first.c and press **enter**. The following window will appear.


C:\WINDOWS\System32\cmd.exe - tc
≡ File Edit Search Run Compile Debug Project Options Window Help
FIRST.C
6
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

At the end of this step, you successfully create a C source file name first.c and save the blank file. Now type the following code in your first.c program.

```
#include<stdio.h>
   int main()
   {
          printf("This is output from my first program!\n");
          return 0;
   }
```

Let us dissect our first program

- This C program starts with **#include <stdio.h>**. This line **in cludes** the "standard I/O library" into your program. The stand ard I/O library lets you read input from the keyboard (called "standard in"), write output to the screen (called "standard out"), process text files stored on the disk, and so on. It is an extremely useful library. C has a large number of standard libraries like stdio, including string, time and math libraries. A **library** is simply a package of code that someone else has written to make your life easier (we'll discuss libraries a bit later).

- The line **int main()** declares the main function. Every C program must have a function named **main** somewhere in the code. We will learn more about functions shortly. At run time, program execution starts at the first line of the main function.

- In C, the **{** and **}** symbols mark the beginning and end of a block of code. In this case, the block of code making up the main function contains two lines.

- The **printf** statement in C allows you to send output to standard out (for us, the screen). The portion in quotes is called the **format string** and describes how the data is to be formatted when printed. The format string can contain string literals such as "This is output from my first program!," symbols for carriage returns (\n), and operators as placeholders for variables (see below).

- The **return 0;** line causes the function to return an error code of 0 (no error) to the shell that started execution. More on this capability a bit later.

Now to compile this source code press **Alt+F9** key or click compiler submenu of *compiler* menu. If there is error in the source program then the compiler will point out the error with line number. On successful compilation, the compiler will give a message that the source program is error free. Press **Ctrl+F9** (or click submenu run of *run* menu) to execute the compiled source code. This will give the required output. Now to see the output generated, press **Alt+F5** key.

CHECK YOUR PROGRESS

7. What is structured programming ?

8. What is a header file? What is a library function ?

1.9 FUNDAMENTALS OF C LANGUAGE

In this section, we will concentrate on the basic elements of C language. The basic elements of C language can be classified as follows :

- Identifiers
- Keywords
- Constants
- C Character set

1.9.1 Identifiers

Identifiers are names that are given to various program elements, such as variables, functions and arrays. Identifiers consist of letters and digits in any order except that the first character must be a letter.To construct an identifier you must obey the following points :

- Only alphabet, digit and underscores are permitted
- An identifier can't start with a digit.
- Identifiers are case sensitive, i.e. uppercase and lower case letters are distinct.
- Maximum length of an identifier is 32 characters.

The following names are valid identifiers :

x	**y12**	**sum_1**	**temperature**
names	**area**	**tax_rate**	**table**

The following names are not valid identifiers for the reasons stated:

x"	illegal characters (").
order-no	illegal character (-)
total sum	illegal character (blank space)

1.9.2 Reserved Word

Reserved words are the essential part of a language definition. The meaning of these words has already been explained to the C compiler. So you can't use these reserved words as a variable names. Since these reserved words have some special meaning in C, therefore these words are often known as "keyword". In the following shows all the reserved word available in C.

auto	*double*	*if*	*static*
break	*else*	*int*	*struct*
case	*enum*	*long*	*switch*
char	*extern*	*near*	*typedef*
const	*far*	*register*	*union*
continue	*float*	*return*	*unsigned*

default	*for*	*short*	*void*
do	*goto*	*signed*	*++ while*

1.9.3 Constants

A ***constant*** is a container to store value. But you can't change the value of that container (constant) during the execution of the program. Thus, the value of a constant remains constant through the complete program.

There are two broad categories of constant in C, ***literal constant*** and ***symbolic constant***. A literal constant is just a value. For example, 10 is a literal constant. It does not have a name, just a literal value. Depending of the type of data, literal constant is of different type. They include *integer*, *character* and *floating point* constant. Integer constant can again be subdivided into *decimal* (base-10), *octal* (base-8), and *hexadecimal* (base-16) integer constant. One important variation of character constant is *string constant*. Table 1.4 explains the different types of literal constant. Remember that a character constant is always enclosed with single quotation mark, whereas a string constant is always enclosed with a double quotation mark. Another point to remember is that an octal integer constant always starts with 0 and a hexadecimal constant with 0x.

EXAMPLE	**TYPES**
153	Decimal integer constant
015	Octal integer constant
0xA1	Hexadecimal integer constant
153.371	Floating point constant
'a'	Character constant
'1'	Character constant
"a"	String constant
"153"	String Constant

1.9.4 C Character Set

C does not use, nor does it require the use of, every character found on a modern computer keyboard. The only characters required by the C Programming Language are as follows :

Alphabets	A - Z a - z
Digits	0 - 9
Special Symbol	# & | ! ? _ ~ ^ { } [ ] ( ) < > space . , : ; ‘ $ “ + - / * = %

1.10 BASIC DATA TYPES IN C

Now, we have to start looking into the details of the C language. To process with data first of all you must know the type of the data. Data type of C has 3 distinct categories. Figure 1.1 explains the different categories of C data type.

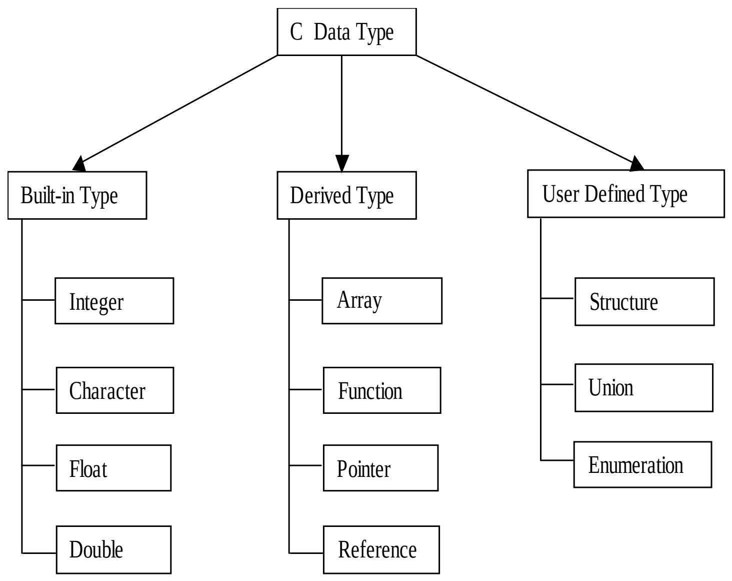


Fig 1.1: Classification of C data type

The first category of data type is the ***built-in*** data type, which are also known as elementary or basic type. Sometime these are called the "primitive" type. These basic data type have several type modifiers, which alter the meaning of the base data type to yield a new type. Table 1.1 lists all combinations of the basic data types and modifiers along with their size and ranges:

Type	**Size (Bytes)**	**Range**
char	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2	-32768 to 32767
unsigned int	2	0 to 65535
signed int	2	-32768 to 32767
short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
signed sort int	4	-32768 to 32767
long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
signed long int	4	-2147483648 to 2147483647
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E-308
long float	10	3.4E-4932 to 1.1E+4932

Table 1.1: Size and range of basic data type and its modifier

Moreover, besides these basic data types another special data type is ***void***. The void type specifies the return type of a function, when it is not returning any value. Sometime void is used to indicate an empty parameter to a function. After all, this data type holds the literal meaning of void. At this point don't worry about the other two categories. All these will be discussed in the subsequent unit.

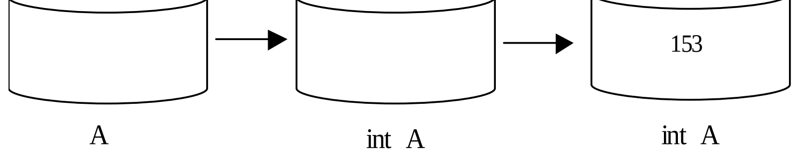
The "%" symbol along with a (special) character in known as ***format specifier*** or ***conversion specifier***. It indicates the data type to be printed or scanned and how that data type is converted to the character that appears on the screen. Format specifiers for usual variable types are shown in Table.1.2.

Format Specifier	**Usual VariableType**	**Display as**
%c	char	single character
%d	int	signed integer
%f %lf	float or double	signed decimal
%e %le	float or double	exponential format
%o	int	unsigned octal value
%u	int	unsigned integer
%x	int	unsigned hex value
%ld	int	long decimal integer
%s	array of char	sequence of characters

Table 1.2: Format specifier for usual variable type

1.11 C - VARIABLES AND THEIR DECLARATIONS

A variable is an identifier to store value. You can resemble a variable with a container which takes different values at different time during the execution of the program. Thus, the value of the variable may change within the program. A variable name can be chosen by the programmer in a meaningful way that reflects what it represents in the program. The naming convention of variable follows the rule of constructing identifiers. Suppose you want to store value 153 to a variable. What you will do is – first create the name of the variable, suppose A. Since 153 is integer so declare the variable A as integer, and then assign 153 to that variable.



Now, in C programming language this can be done using the following statement :

```
int  A ;
A = 153 ;
```

The first statement says that A is a container, where we can store only integer type variable. This means that we can't store value into A other than integer. Therefore this type of statement is known as declaration statement (A declares that A can store only integer type of variable). Thus the general form of declaration of a variable is

data_type variable1, variable2, ... , variableN;

By declaring a variable you tell 3 things to the compiler :

- What the variable name is.
- What type of data the variable will hold.
- and the scope of the variable.

Up to this point container A is empty. The second statement says that the value 153 is stored in A. This means variable A is initialized with 153. Therefore this type of statement is known as variable initialization. A variable must store a value after it has been declared (but before it is used in an expression). You can store values to a variable in two ways

:
- By using assignment statement.
- and by using a read statement.

The first method is used in the above example. In second approach you can make a call of C standard input function (that is *scanf, getch, getc, gets* etc.) to store value to a variable. For example, the above initialization statement can be written as

scanf("%d",&A);

This statement will take an integer type input from standard input device (that is keyword) and store it to A.

The above two statement (program segment) can be written in a single statement.

```
int  A = 153;
```

This type of statement is known as initialization of variable during declaration. As a shorthand, you can declare variables that have the same type in a single line of declaration by separating the variable names with commas. For example, you can declare the variable j and k in a single line as :

```
int j, k;
```

which is the same as the declaration of j and k as :

```
int j;
int k;
```

It is always a good practice to group together declarations of the same data type for an easy reference. For example :

```
int j, k;
float x,y,z;
```

A few examples of variable declarations are shown below :

Variable declaration	**Remarks**
int i = 0, j = 1;	*i* and *j* are declared as integer variables. The variables i and j are initialized with value as 0 and 1 respectively.
float basic_pay;	*basic_pay* is a floating point variable with a real value or values containing decimal point.
Char a;	*a* is a character variable that stores a single character.
double theta;	*theta* is a double precision variable that stores a double precision floating point number.

1.12 SYMBOLIC CONSTANTS IN C

A *symbolic constant* is a name that substitutes for a sequence of characters. The characters may represent a numeric constant, a character constant is a string constant. Thus, a symbolic constant allows a name to appear in place of a numeric constant, character constant or a string. When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence.

Symbolic constants are usually defined at the beginning of a program. The symbolic constants may then appear later in the program in place of the numeric constants, character constants, etc. that the symbolic constants represent.

Symbolic constants are defined using *#define* as given below:

#define<symbolic constant name> <value>

Suppose that you are writing a program which performs a variety of geometrical calculations. For example, using the value π (3.14) for the calculations. To calculate the circumference and area of a circle with a known radius, you could write

```
circum = 3.14 * (2*radius);
area = 3.14 * (radius) * (radius);
```

If, however, you define a symbolic constant with the name PI and assign it the value 3.14, you would write the name PI and the value 3.14 as shown below

```
#define PI 3.14
circum = PI * (2*radius);
area = PI * (radius) * (radius);
```

Some valid examples of symbolic constant definitions are :

```
#define TAXRATE 0.55
#define TRUE 1
#define FALSE 0
```

CHECK YOUR PROGRESS

9. What is an identifiers ?
10. What is a keyword in C ?
11. What are the types of constants ?
12. What is the built in data types in C ?
13. What is a variable ?

1.13 LET US SUM UP

A computer program is a set of instructions that directs the computer to perform some specific task.
Characteristics of a good program are - accurate, efficiency, reliability, portability, robustness etc.
The program development cycle consists of the phases : *problem definition, analysis, design phase, testing the algorithm for accuracy, coding, testing and debugging, implementation and documentation* and the last one is *program maintenance*.

The programming languages can be classified as: machine language, assembly language, high level language and 4GL language.

The compiler, interpreter and assembler are the language translator programs.

A linker is a program that attached or links many modules of a program.

The loader brings a program residing on disk into the main memory of computer and run it.

Pseudocode is a program-planning tool that allows programmers to plan program logic by writing program instructions in an ordinary natural language, such as English.

The term *algorithm* refers to the logic of a program. It is a step-by-step description of how to arrive at a solution to a given problem.

Pictorial representation to depict clearly the flow of control to arrive at the solution of a problem is called flowchart.

The programming techniques used for designing a program are: *top-down technique, bottom-up technique, unstructured technique, structured technique and modular programming technique*.

C is a general purpose, high-level programming language developed by **Kerningham** and **Ritchie** at AT & T Bell Labs.

C program logic is a combination of statements. Statements are always found between { } braces called the body of a function. Each statement performs a set of operations. Simple statements are terminated by semicolon ' ;'.
Every C program is required to have a special function called *main*. This function is the entry point of the program.

Identifiers are the name given to the various program elements - variables, functions, arrays etc.

C character set includes uppercase and lowercase alphabets, digits and several special characters. Altogether there are 93 valid characters allowed in C.

There are 32 keywords (reserved words) in C. They cannot be used as variable names.

A C variable is an entity whose value may vary during program execution. C makes it compulsory to declare the type of any variable name that a programmer wishes to use in a program before using it. The basic data type that can be used for such declaration are *int, float, double* and *char.*

Symbolic constants are generally defined at the beginning of a program.

1.14 FURTHER READINGS

1. Balagurusamy, E: *Programming in ANSI C*, Tata McGraw-Hill publication.
2. Gottfried Byron S: *Programming with C*, Tata McGraw-Hill publication.

1.15 ANSWERS TO CHECK YOUR PROGRESS

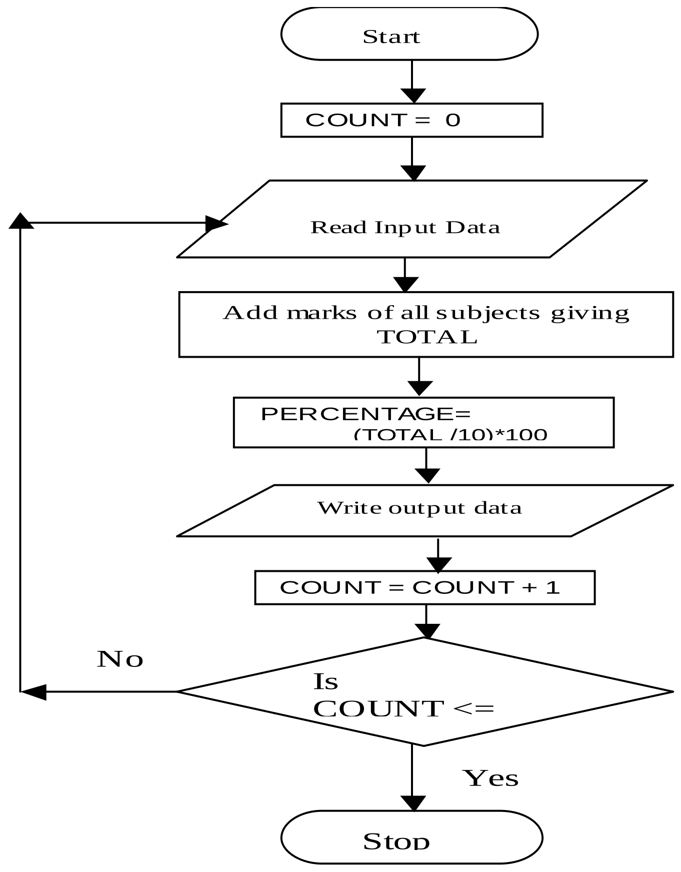
1. a) T, b) F, c) F, d) T, e) T
2. a) F, b) T, c) T, d) F, e) F
3. A **compiler** is a computer program (or set of programs) that translates text written in a computer language (the *source language*) into another computer language (the *target language*).

A program that is written in a high level language must be translated into machine language before it can be executed. This is known as compilation process.

4. The language in which a programmer writes programs is called source language. It may be a high level language or an assembly language. A program written in a source language is called a source program.

The language in which the computer works is called object language or machine language. When a source program is converted into machine language by an assembler or compiler it is known as an object program. In other words, a machine language program ready for execution is called an object program.

5.



6. In a Fibonacci sequence, the previous two numbers are added to generate the next Fibonacci number.

f1 = 1 Ist number

f2 = 2nd number

f3 = f1 + f2 = 1+2 = 3

f4 = f2 + f3 = 2+3 = 5

f5 = f3 + f4 = 3+5 = 8, and so on.

f1	f2	f3	f4	f5	f6	f7 ....
1	2	3	5	8	13	21 ....

To get next fibonacci number, we have to do sum of previous two numbers in the series.

Algorithm :

1. Assign sum =0, A=0, B=1, i=1

2. Get the number of terms upto which you want to generate the

Fibonacci number, i.e. n.

3. Add A and B to get the next Fibonacci number
4. Assign the value of B to A i.e. A=B.
5. Assign the value of sum to B i.e. B = sum
6. Write the value of sum to get next Fibonacci number in the series.
7. Increment i with 1 i.e. i = i+1 and repeat step 3,4,5,6 with the last value of i = n (n is the number of terms up to which we want to generate Fibonacci number series.)
8. Stop

7. In structured programming a program is broken down into small independent task and each task has its own functionality and perform specific part of the actual processing. These task are developed independently without the help of the other. When these task are completed, they are combined together to solve a whole program.

8. In computer programming, particularly in the C and C++ programming languages, a header file or include file is a file, usually in the form of source code, that is automatically included in another source file by the compiler. Typically, header files are included via compiler directives at the beginning (or *head*) of the other source file.

The **C standard library** (also known as **libc**) is a now-standardized collection of header files and library routines used to implement common operations, such as input/output and string handling, in the C programming language. Unlike other languages such as COBOL, Fortran, and PL/I, C does not include built-in keywords for these tasks, so nearly all C programs rely on the standard library to function.

9. Identifiers are names that are given to various program elements, such as variables, functions and arrays. Identifiers consist of letters and digits in any order except that the first character must be a letter.

10. Keywords are also called the reserved words in C. They have specific meaning to compiler. These words should not be used for naming any other variables.

11. There are two broad categories of constant in C, ***literal constant*** and ***symbolic constant.***

12. The built in data types are *integer, character, float, double*.

13. A variable is an identifier that is used to represent a single data item i.e. a numerical quantity or a character constant. A given variable can be assigned different data items at various place within thin a program.

1.16 MODEL QUESTIONS

1. What is a program ? Explain the characteristics of a good program.
2. Describe the various stages of program development.
3. Write down the characteristics of high level language ?
4. Write the difference:
   a) Machine language and Assembly language
   b) High level language and Machine language
   c) Compiler and Interpreter
   d) Linker and Loader
   e) Compiler and Assembler
   f) Pseudocode and Algorithm
5. What do you understand by compilation and execution of a program.
6. What is an algorithm ? Why is it necessary to write an algorithm before program coding ?
7. How flowchart is more effective than algorithm ? Explain with algrithm.
8. What do you mean by programming technique ? Compare modular and structured programming techniques giving example.
9. Give the merits and demerits of modular and structured programming technique.
10. What are the major differences between compilation and

interpretation ? Which process does take more time on repeated processing and why ?

11. Write down a few characteristics of ‘C’ language.
12. Write an algorithm to find the area of a triangle.
13. Write an algorithm to find the sum of a set of number.
14. Write an algorithm to test whether the given number is a prime number.
15. Write an algorithm to find the factorial of a given number.
16. What is the difference between a keyword and an identifier ?
17. List the rules of naming an identifier in C ?
18. Name and describe the four basic data types in C ?
19. What is a variable ? How can variables be characterized ?
20. Draw a flowchart to print the sum of numbers between 1 and the entered number. For example if you enter 5, then it will find the sum of 1+2+3+4+5 = 15
21. Draw a flowchart to input three numbers and print the largest number.
22. Write an algorithm that reads a year and determine whether it is a leap year or not.
23. Write an algorithm to sort an array in the descending order.