

## Section - 1

# COMPUTER FUNDAMENTALS & HISTORICAL DEVELOPMENT OF C

A computer is a programmable electronic device that accepts data and instruction from input devices, process the data and provides result as information in the output. The electronic device is known as *hardware* and the set of instructions is known as *software*.

Computer accepts raw data and then stores, retrieves, sends, receives, analyzes and synthesizes the data to produce information. Computers are used in wide areas in our daily lives to accomplish tasks quickly and efficiently. Computer has program to complete the tasks called computer program or software. A program is a set of sequenced instructions which causes a computer to perform particular operations.

## 1.1 HISTORICAL DEVELOPMENT OF COMPUTERS

By the date, the historical development of computing technology is summarized below:

Year	Device/Description
3000BC	ABACUS by Chinese
1614	Napier bones and logarithms by John Napier
1620	Slide Rule by Edmund Gunter
1632	Improved Slide Rule by William Oughtred
1642	Pascaline by Blaise Pascal
1671	Leibnitz calculator by Gottfried Wilhelm von Leibnitz
1801	Punch Card loom by Joseph Marie Jacquard
1822	Difference Engine by Charles Babbage
1834	Analytical engine by Charles Babbage
1854	Boolean Algebra by George Boole
1890	Punch Card Machine by Herman Hollerith
1906	Electronic Valve invented by De Forest
1930	Differential Analyzer by Vannevar Bush
1937	Binary adder built by George Stibitz, First Digital Electronic Computer Designed by John V. Atanasoff
1941	First General purpose computer designed by Konard Zuse
1944	First Automatic Computer, MARK I designed by Howard Aiken

1945	ENIAC (Electronic Numerical Integrator and Computer) by John W. Mauchly & J. Presper Eckert, Computer System Elements outlined by Jon V. Neumann
1946	UNIVAC designed principally by J. Presper Eckert & John Mauchly
1947	Transistor Invented by John Bardeen, William Shockley & Walter Brattain
1948	MARK III by Howard Aiken. Also known as ADEC (Aiken Dahlgren Electronic Calculator)
1951	First Business computer UNIVAC(UNIversal Automati Computer) became operational
1952	ERA 1101 became UNIVAC 1101
1953	Introduced 604 Computer by Tom Watson in IBM(International Business Machine).
1956	Second Generation Computer (using transistors) introduced by Bell Laboratory
1959	Integrated Circuits(ICs) demonstrated by Clair Kilby
1964	First third generation computer using ICs developed. IBM produced first Americal Airline teservation tracking system.
1965	First commercial mini computer, PDP-8 by digital equipment corporation
1969	ARPANET (Advane Research Project Agency NETwork) funded by US DOD (Department of Defense)
1971	Intel 4004 Microprocessor designed by Ted Hoff, IBM introduced first 8-inch "memory disk", later it is called floppy disk.1974
1972	Intel made the 8-bit 8008 and 8080 microprocessor.
1974	First fourth generation computer built by Ed Roberts.
1975	First Personal Computer Software created by Bill Gates and Paul Allen
1976	Apple personal computer by Steve Jobs and Wozniak, 5.25-Inch Floopy developed by Shugart
1981	IBM PC introduced in the market
1982	Cray Super Computer by Cray Research Company
1984	Macintosh PC by Apple
1989	Optical Computer demonstrated
1990	Motorola announced 32-bit microprocessor
1991	WWW technology develoed by Tim Berners-Lee at CERN(European Organization for Nuclear Research)
1992	IBM think pad, laptop by IBM
1995	Pentium pro microprocessor by Intel
1996	200Mhz Pentium processor introduced by Intel
1997	Intel Pentium-II microprocessor

1999	Intel Pentium-III microprocessor
2000	Pentium-4 released
2006	Intel Core-2 Duo processor released.
2010	Intel Core-i3 & beyond

## 1.2 COMPUTER GENERATIONS

Each generation of computer is characterized by a major technological development that fundamentally changed the way computers operate, it results increasingly smaller, cheaper, more powerful, more efficient and reliable devices.

### 1.2.1 First Generation - 1940-1956: Vacuum Tubes

#### Characteristics:

- a. Vacuum tubes were used for circuitry system & magnetic drum for memory
- b. Bigger in size, consumes large space, generally a whole room
- c. Much expensive to operate, consumes much electrical power,
- d. Generate too much heat, require excessive cooling
- e. Machine language programming.
- f. Solve single problem at a time.
- g. Examples are: UNIVAC, EDVAC(Electronic Discrete Variable Automatic Computer), ENIAC

### 1.2.2 Second Generation - 1957-1963: Transistors

#### Characteristics:

- a. Vacuum tubes in the circuitry system was replaced by transistors.
- b. Transistors were faster and more reliable than vacuum tubes.
- c. It was a punch card technology
- d. Used assembly language in programming, better in understanding than machine language.
- e. Consume less power, generate less heat and more compact in size as compared with first generation computers. However, it required excessive cooling.
- f. Examples are : IBM 1620, PDP-I, PDP-5. (PDP: Programmed Data Processor)

### 1.2.3 Third Generation - 1964-1971: Integrated Circuits

#### Characteristics:

- a. Numbers of transistors, diodes & registers were integrated into a single silicon chip called Integrated Circuit (IC). It was more efficient and smaller in size of the computer as compared with previous generations.

- b. It required less energy, consumes less space in the room.
- c. Could plug different keyboard, monitors and other IO devices.
- d. Interfaced with operating system which allowed the device to run many different applications at one time with a central program that monitored the memory
- e. Used high level languages in programming to develop application.
- f. Exampes are IBM 370, PDP-11 & CDC 7600

### **1.2.4 Fourth Generation - 1972-Present: Microprocessors**

#### **Characteristics:**

- a. Thousands of silicon chip were built into a single chip called Large Scale Integration (LSI) and Very Large Scale Integration (VLSI).
- b. Magnetic core memories were replaced by semiconductor memories
- c. Very high level programming language like query based language, C, C++ Java were used in programming.
- d. The programs developed were easily portable and expandable.
- e. Operating System with smooth Graphical User Interface system was developed.
- f. Efficiency of the system was drastically improved and the overall size is drastically reduced.
- g. It consumes less it.
- h. Had larger primary and secondary storage memory.

### **1.2.5 Fifth Generation - Present and Beyond: Artificial Intelligence**

#### **Characteristics:**

- a. Based on artificial intelligence, are still in development
- b. Offers Ultra Large Scale Integration (ULSI) technology
- c. PCs are potable, smaller and handy.
- d. Desktop PCs are more powerful, reliable and have fewer possibilities of failure handling.
- e. Embeded artificial intelligence like voice recognition system, parallel processing, super computing, robotics, game playing, expert systems.
- f. Achieve natural language processing with quantum computation and molecular technology will radically change

## **1.3 COMPUTER SYSTEM AND ORGANIZATION**

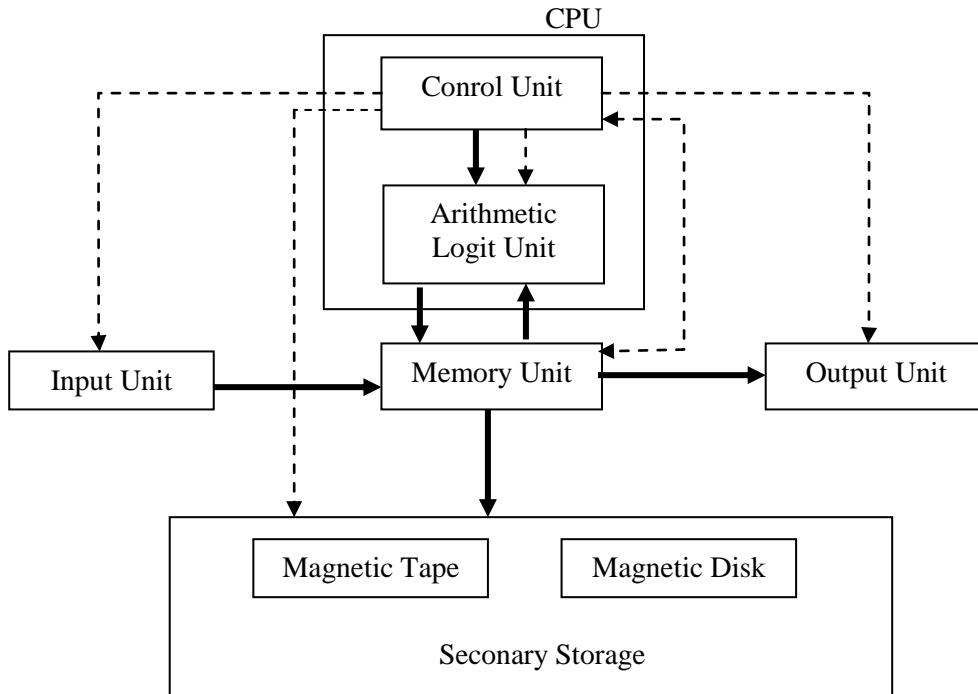
---

Computer System is divided into two fields- hardware and software. The mechanical or the physical parts of computer like CPU, Monitor, and Keyboard etc are computer hardware. Software is a program which makes the computer work and function.

Computer hardware refers to the physical parts of the computer system and software is the set of instructions or programs that are necessary for the functioning of computer to perform certain tasks.

### 1.3.1 Computer Hardware

The computer hardware system principally consists of components depicted below in the block diagram with brief descriptions.



*Figure 1.3: Interaction among hardware components*

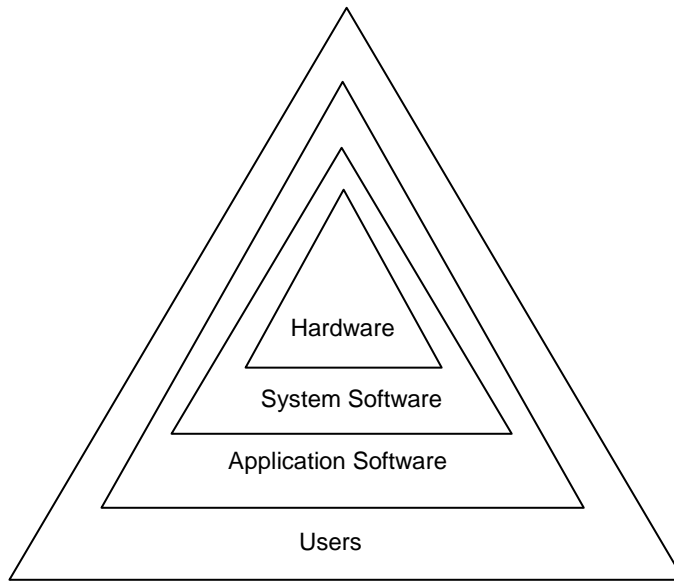
- a. **Input Units:** They are the input devices from which computer accepts data on which the operations are to be performed.
- b. **Processing Unit:** It is also called the Central Processing Unit, used to perform computations and information processing on those data that is entered through input devices. Processing unit consists of Control Unit (CU) and the Arithmetic Logic Unit (ALU).

Control unit stores the instruction sets through which it directs the entire computer system to carry out or execute stored program. Arithmetic Logic Unit (ALU) performs arithmetical and logical operations on the data received through the control unit instructions.

- c. **Output Unit:** They are set of output devices used for providing the output of a program that is obtained after performing the operations specified in a program. Output devices display or print the output results of the operations on the input data.
- d. **Memory Unit:** Memory in a computer is storage area needed to store instructions and data, either temporarily or permanently. There are two types of memory in computer system: Primary memory and Secondary memory. Primary memory stores data and programs while the program is being executed. The secondary memory stores data and programs for long periods of time. It provides large, non-volatile and cheap storage for programs and data.

### 1.3.2 Computer Software

Software is a computer program which is a sequence of instructions designed to direct a computer to perform certain functions. Software is generally categorized as system software and application software. Users operate the software where software operate the hardware to process data and get results. Following diagram shows the relationship among user, software and hardware.



The type of software which is most essential for computer operation and controls the hardware components of a computer are called *system software*. System software refers to the files and programs that make up computer's operating system. The programs that are part of the system software include assemblers, compilers, file management tools, system utilities and debuggers.

The types of software which is used for user specific applications are called application software. These are the applications that most of the users are familiar with, such as Office Word, Excel, Powerpoint and Netscape Communicator.

## 1.4 COMPUTER PROGRAM AND PROGRAMMING LANGUAGES

---

Computer program is a set of instructions that, when executed, causes the computer to behave in a predetermined manner.

Computers generally don't understand natural languages like English or Nepali unless it is instructed. The languages, which are used to instruct the computer to perform certain tasks are called *computer programming languages*. There are many programming languages like C, C++, Pascal, BASIC, FORTRAN, COBOL, LISP, etc.

### 1.4.1 Types or Levels of Programming Languages

Programming languages are categorized in the following:

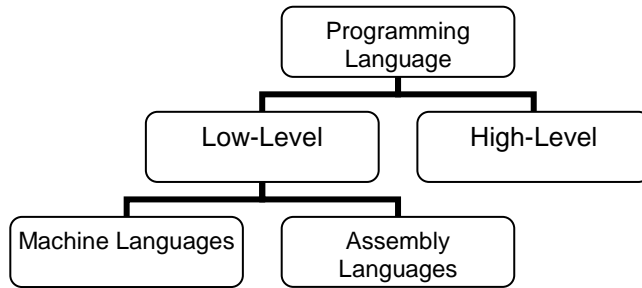


Fig 1.4 .1 Classification of programming languages

Low-level language is a programming language in which each statement or instruction is directly translated into a single machine code.

A high-level programming language is a language that is more user-friendly, to some extent platform-independent and abstracts from low-level computer processor operations such as memory accesses. They are similar to natural languages (like English) and so are easy to write and remember.

### I. First Generation Programming Language: Machine Language

Machine language has the least possible level at which we can program a computer in its own native machine code, consisting of strings of 1's and 0's, and are stored as binary numbers. The main advantage of machine language is that they execute faster than high-level language. However, machine languages are more difficult to write.

### II. Second Generation Programming Language: Assembly Language

Assembly language is categorized as the second generation programming language. it is a symbolic representation (called *mnemonics*) of machine code. Assembly program is closer to plain english words, hence a bit more easier to read and write the program as compared with machine language but the computer can't understand them directly. The assembly-language program must be translated into machine code by a separate program called an *assembler*.

### III. Third Generation Programming Language (3GL): High Level Language

It is a refinement of second generation programming language (2GL). 3GL was introduced to make the language more programmer friendly. High level languages falls somewhere between assembly language and natural languages. It includes languages like FORTRAN, COBOL, BASIC, C, C++.

The programming language that bridges the gap between traditional machines/ assembly language and conventional high level language is called middle level language . C language on one hand supports assembly language and we can access memory directly using pointer, on the other hand C supports high level language features. However C is categorized into high level language, it is easy to create machine code using C. Hence, C is sometimes also called **middle level language**.

### IV. Fourth Generation Programming Language (4GL): Very High Level Language

The higher the generation of language means the more efficient, faster and user friendly programming language. The features of 4GL are:

- a. It is a non-procedural programming language i.e. query language.
- b. Code comprising of English like sentences.
- c. Program is portable and easily expandable.
- d. 4GL code enhances the productivity of the programmers as they have to type fewer lines of code.

A typical example of 4GL is the query language that allows user to extract data from database, data warehouse and big data.

## **V. Fifth Generation Programming Language (5GL): Future Language**

Fifth generation languages are based on Artificial Intelligence. It solves the problem using constraints given to the program. 5GLs are considered to be the wave of the future and predicted to replace all other languages for system development. However, it is a highest level of programming language that meets the visual programming requirements. Examples of 5GL are Prolog, OPS5 and Mercury.

3GL, 4GL and 5GL are in overall categorized into High Level Languages. From the programmers point of view, using high level language is time saving. It is designed to reduce programmer's effort. High level languages are simply classified into following two categories:

- Procedure Oriented (Function oriented) Programming
- Object Oriented Programming

One or more related blocks of statements that perform some complete function are grouped together into a program module is simply called a Procedure. Procedure is just a mini program that performs specific task. The languages that are used to write such procedures to perform the task defined are called **procedure oriented languages**.

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. C++ and JAVA are the examples of **Object Oriented programming language**.

### **1.4.2 Compiler, Interpreter and Assembler**

The compiler takes the source code as input and produces the machine language code (object code) for the machine on which it is to be executed as output.

An interpreter, like compiler, is also translator which translates high level language into a machine level language. The difference between compiler and interpreter is that the interpreter translates and executes the program line by line.

An assembler is a program that takes basic computer instructions (Assembly Language) and converts them into a pattern of bits.

### **Compiling->Linking/Loading-> Executing**

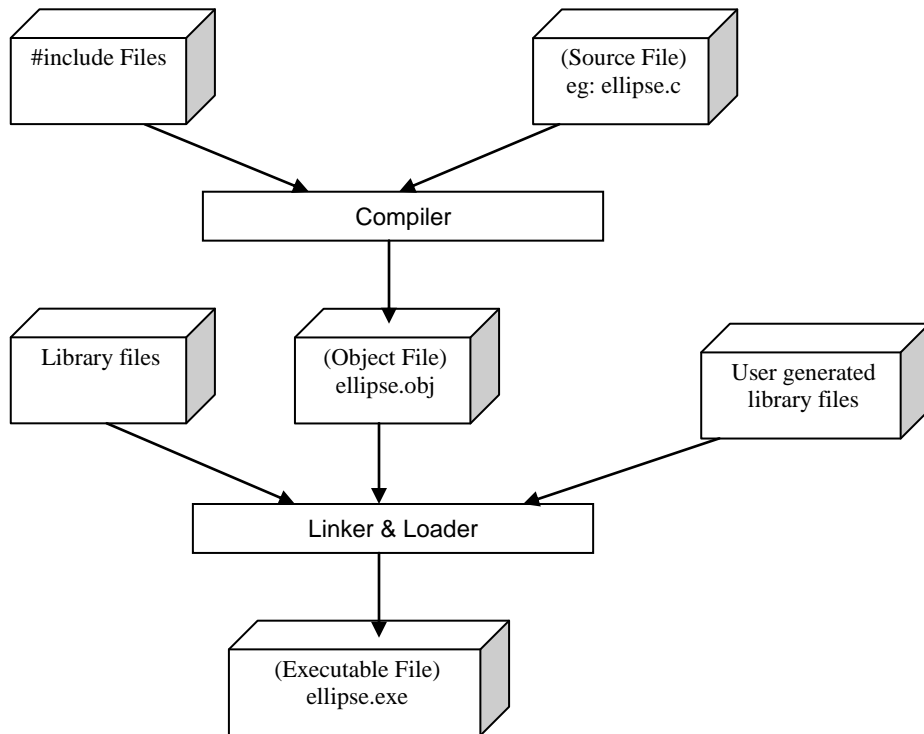
Compiling the program means generating an intermediate kind of file called object file. The object file is required to link with built-in library files and possible other user-generated object files to make the program executable.



Bigger programs may have several separate files, some of which may already be compiled. Here the job of linker is to combine these files into a single executable file. Hence from coding to execution the basic steps are:

- i. Program writer (also called coder) creates `.c` source files which is then passed to compiler producing `.obj` object file
- ii. `.obj` object files are passed to the linker/loader which includes all necessary library files to produce `.exe` executable files.
- iii. `.exe` file is the executable file which is then loaded into the memory to be executed by the user. User can run this executable file anytime to perform his/her tasks required.

The relationship between compilation and linking process is depicted below:



**Fig: 1.4.2 Compilation, Linking and Execution**

The executable files will successfully be created if there are no any errors encountered by compiler and linker. The job of loader is to load the executable program (binary program) into memory for execution.

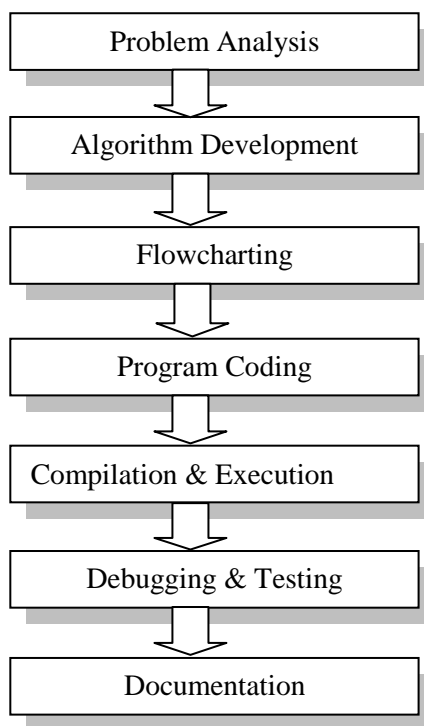
- Compiler translates the block of high level source codes at a time into another file consisting of machine language.
- Assembler translates block of assembly language source codes at a time into another file consisting of machine language.
- Interpreter translates program source code line by line.

## 1.5 PROBLEM SOLVING USING COMPUTER

---

The steps of problem solving are:

- 1) Problem Analysis
- 2) Algorithm Development
- 3) Flowcharting
- 4) Programme Coding
- 5) Compilation & Execution
- 6) Debugging & Testing
- 7) Documentation



*Fig 1.5: Steps in problem solving by a computer*

### 1.5.1 Problem Analysis

**Problem analysis** is the process of decomposing whole or parts of system into smaller parts or modules. Then identify possible inputs, processes and outputs with problems. Basically we do syntactic and semantic analysis in this phase.

Let us consider a problem: "Given the two dimensions (major axis, minor axis) of an ellipse, what is the area?"

**Analysis of the given problem:**

Decomposition of problem description is the first step of solving problem. It could be achieved syntactic analysis like doing it into flowing five steps:

## I. Identify all the nouns in the sentence

Given 2 dimension of an ellipse (major & minor axis), calculate the area

The nouns in the problem specification identify descriptions of information that you will need to either identify or keep track of. Once these nouns are identified, they should be grouped into one of two categories:

**Input** (items either already known or getting it from the user)

**Output** (items that finds out by manipulating the input)

Input		Output
Dimensions	<i>We suppose, these dimensions are given</i>	Area <i>(we need to calculate this)</i>
Major		
Minor		
Pi		
Ellipse		

## II. Eliminate redundant or irrelevant information

There may be some information in the problem description that make it into our input/output chart that we really don't need to solve the problem (that is, not all of the nouns may be relevant). Also, there may be some nouns that appear redundant (Information we already have in our table, just in a different form).

Input		Output
<del>Dimensions</del>	<i>We don't need the noun "Dimensions" because we already have major and minor axis, we also don't need "ellipse" to calculate volume, if we know the dimension</i>	Area <i>(we need to calculate this)</i>
Major		
Minor		
Pi		
<del>Ellipse</del>		

We need to keep the most specific nouns possible in the table. When in doubt, try to piece it together logically: when figuring out the area, which nouns would be the most useful to you?

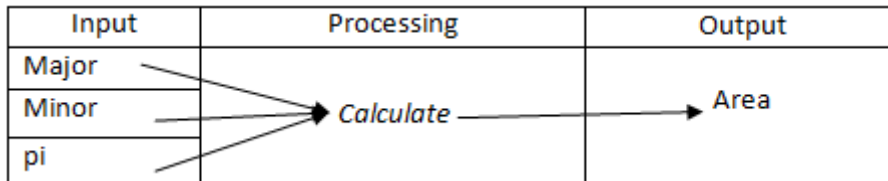
### III. Identify all of the verbs in the sentence

The verbs in the problem specification identify what actions your program will need to take. These actions, known as processing are the steps between your input and your output.

Input	Processing	Output
Major	<i>Calculate (apply mathematical formula)</i>	Area
Minor		
pi		

### IV. Link your inputs, processes, and output

This step is as simple as drawing lines between the relevant information in your chart. Your lines show what inputs need to be processed to get the desired output. In our example, we need to take our major, minor, and Pi (as constant value) and multiply them, to give us our desired area.



### V. Use external knowledge to complete your solution

To solve a problem, we need to know the extra knowledge from mathematics and science. At this point you are required to understand what “calculate” means. In some arbitrary problem, “calculate” could refer to applying some mathematical formula or other transformation to our input data in order to reach the desired output.

## 1.5.2 Algorithm Development

An Algorithm is the step-by-step description of the procedure written in human understandable language for solving given problem.

Let us consider an example of an algorithm for making a pot of tea.

Step1: Start

Step2: If the kettle doesn't contain water, then fill the kettle.

Step3: Plug the kettle into the power point and switch it on.

Step4: If the teapot is not empty, then empty the teapot.

Step5: Place tea leaves in the teapot.

Step6: If the water in the kettle is not boiling, then go to step6.

Step7: Switch off the kettle.

Step8: Pour water from the kettle into the teapot.

Step9: Stop.

It can be seen that the algorithm has a number of steps and some steps involve decision making (Step2, Step4 and Step6) and one step (Step6) involves repetition, in this case the process of waiting for the kettle to boil. The rest of steps are simply sequential.

From this example, it is evident that algorithms show the following three features:

- Sequence (also known as process)
- Decision (also known as selection)
- Repetition (also known as iteration or loop)

### Some conventions used in developing algorithms

- Each algorithm will be enclosed by two statements START and STOP
- To accept data from user, the INPUT or READ statement shall be used.
- To display any user message, the PRINT or DISPLAY statement shall be used.
- The relevant operators shall be used in the expression and condition based on situation described.

For example, to write the algorithm of the problem: "Given the two dimensions (major axis, minor axis) of an ellipse, what is the area?"

During problem analysis, we are now well known about the possible inputs and processing to be carried out to get the output of the problem. The algorithm looks like the following:

Step1: Start

Step2: Define constant variable Pi which holds value 3.1425

Step3: Define other variables for input and output: major, minor & area

Step3: Read major and minor

Step4: Calculate the area of an ellipse:

$$\text{area} = \text{Pi} * \text{major} * \text{minor}$$

Step5: Display area of an ellipse

Step6: Stop

### 1.5.3 Flowcharting

Flowchart is the graphical representation of an algorithm using standard symbols. In other words, flowchart is a pictorial representation of an algorithm that uses boxes of different shapes and connecting lines/arrows to denote different types of instructions.

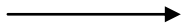
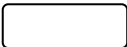
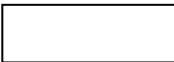
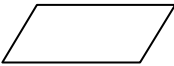
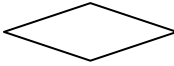



Advantages of Flowcharts

- *Communication:* Flowcharts are a better way of communications.
- *Effective Analysis:* Flowcharts provide a clear overview of the entire problem and its algorithm for solution.
- *Proper Documentation:* The flowchart provides a permanent recording of program logic. It documents the steps followed in an algorithm.
- *Efficient Coding:* Flowcharts show all major parts of a program. A programmer can code the programming instructions in a computer language more easily with a comprehensive flowchart as a guide.
- *Easy in debugging and program maintenance:* Flowcharts help in the debugging process and maintenance of operating program.

### Limitations of Using Flowcharts

- *Complex Logic:* A flowchart becomes complex and clumsy when the program logic is quite complicated.
- *Difficulty in alteration and modifications:* If alterations are required; the flowchart may need to be redrawn completely.

## Flowchart Symbols

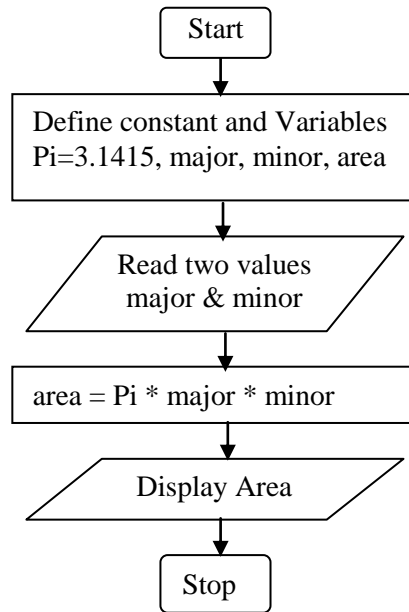
	Arrow	Used to connect flowchart symbols and the direction indicates the flow of logic
	Start/Stop/End (Terminal)	Used to represent the beginning and end of task
	Rectangle	Used for arithmetic and data manipulation operations. The instructions are written inside the symbol.
	Input/Output	Used for input (reading data) and output (displaying data). the data to be read and displayed are written inside the symbol.
	Decision	Used for decision making and branching operations that has two alternatives (true or false, yes or no)
	Connectors	Used to connect different flow lines and remote parts of the flowcharts on the same page
	Function Call	Used whenever you call the function from main or other user defined functions. Function name is written inside the box.
	For Loop	Used to indicate for loop

*Table 1.1 Symbols used for flowchart*

### Guidelines in flowcharting

- Flowcharts should be started from the top of the page and flow down and to the right.
- Only standard flowcharting symbols should be used.
- There should be start and stop on every flowchart.
- The flowchart should be clear, neat, and easy to follow. There should be no ambiguity in understanding the flowchart.
- The direction of flow line of a procedure or system should be from left to right or top to bottom.
- Only one flow line should emerge from a process symbol.
- Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, can leave the decision symbol.
- Only one flow line is used in conjunction with a terminal (Start and Stop) symbol.
- The contents of each symbol should be written legibly. English like language should be used in flow charts, not specific programming language.
- If the flowchart becomes complex, connector symbols should be used to reduce the number of flow lines. The intersection of flow lines should be avoided to make the flow chart a more effective and better way of communication.

Example flowchart of ellipse problem: To write the flowchart of the problem: "Given the two dimensions (major axis, minor axis) of an ellipse, what is the area?"



### 1.5.4 Coding

The coding is the process of transforming the program logic design into a computer language format. This stage translates the program design into computer instructions using some programming languages like c, c++, java etc. During coding of program, the programmer should eliminate all syntax and format errors from the program and all logic errors are detected and resolved during this process.

**Example for writing a source code of an ellipse problem:** To write the code of the problem: "Given the two dimensions (major axis, minor axis) of an ellipse, what is the area?"

```

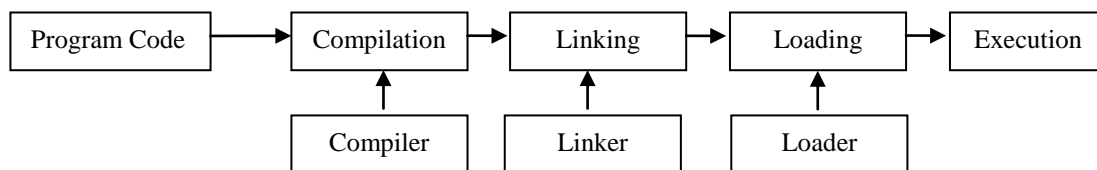
//ellipse.c
/*this code once compiled and executed, calculates
the area of an ellipse after reading major and minor
axis*/
#include<stdio.h>
#include<conio.h>
#define Pi 3.1415
int main (void)
{
float major, minor, area;
//clrscr();
printf("Enter Major and Minor Axis of an Ellipse:");
scanf("%f %f", &major, &minor);
area = Pi*major*minor;
printf("\n The area of an ellipse is: %.2f", area);
getch();
return 0;
}
  
```

The program code is written here just to make understand that "coding" is one of the problem solving steps in computer. Please refer to section 1.9 "structure of C program" to know about the details of program structure and syntax.

**Coding** is the process of transferring paper works like algorithm and flowchart into computer program following the well defined syntax using the programming language like C, C++, JAVA etc..

### 1.5.5 Compilation and Execution

The process of changing high level language into machine level language is known as compilation. It is done by special software, known as compiler. During the execution, the program may ask user for inputs and generates outputs after processing the inputs. Hence, the sequence is:



*Fig 1.5.5: coding, compilation and execution steps*

### 1.5.6 Debugging & Testing

Debugging is the process of discovery and correction of programming errors. Even after taking full care in program design and coding, some errors may remain in the program because the designer/programmer might have never thought about a particular case. These errors may appear during compilation or linking or execution of the program. Program testing and debugging are closely related.

#### ERROR

Error means failure of compilation and execution of the program successfully and/or getting incorrect results. The debugging and testing tasks are for avoiding errors in the program. In the programming, errors are categorized into two as follows:

- Syntax Errors &
- Semantic Errors

Your first attempt of compiling a program won't be successful if you mistakenly typed the program which may not be coded into properly defined syntax. Compiler will generate the syntax error if the program code is not correctly written.

We debug the program generally to avoid syntax errors as well as semantic errors. Semantic or logic errors are more generally captured while testing the output or verifying the result to make sure that your logic is working properly. Additionally there may appear **run time error** during the execution of program like stack overflow and floating point error like divided by zero

### 1.5.7 Program Documentation

“**Program documentation**” is the systematic writing of activities from problem analysis to debugging/testing phases including operational information and presenting it as a report.

There are two types of documentations:

- a) Programmer's Documentation (Technical Documentation)
- b) User Documentation (User Manual)

Programmer's documentation is prepared for future reference to the programmers who maintain, redesign and upgrade the system. The user documentation provides support to the user of the program. This provides instructions for installation of the program and use of it effectively.



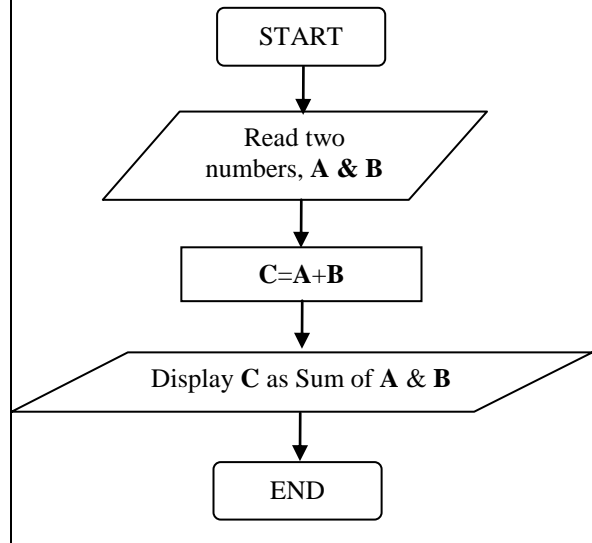
## 1.6 SOLVED EXAMPLES

### 1. Write an algorithm and draw flowchart for finding the sum of any two numbers.

#### Algorithm:

1. Start
2. Display "Enter two numbers".
3. Read A and B
4.  $C=A+B$
5. Display "C as sum of two numbers".
6. Stop.

#### Flowchart:

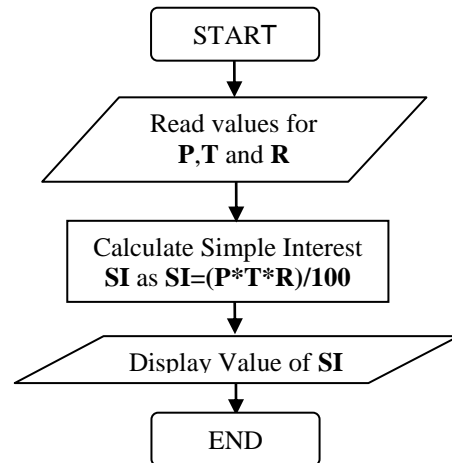


### 2. Write an algorithm and draw a flowchart for calculating the simple interest using the formula $SI=(P*T*R)/100$ , where P denotes the principal, T time and R rate of interest.

#### Algorithm:

1. Start
2. Display "Enter value of P, T and R".
3. Read P, T and R.
4. Calculate simple interest using formula,  $SI=(P*T*R)/100$ .
5. Display SI as simple interest.
6. Stop

#### Flowchart:

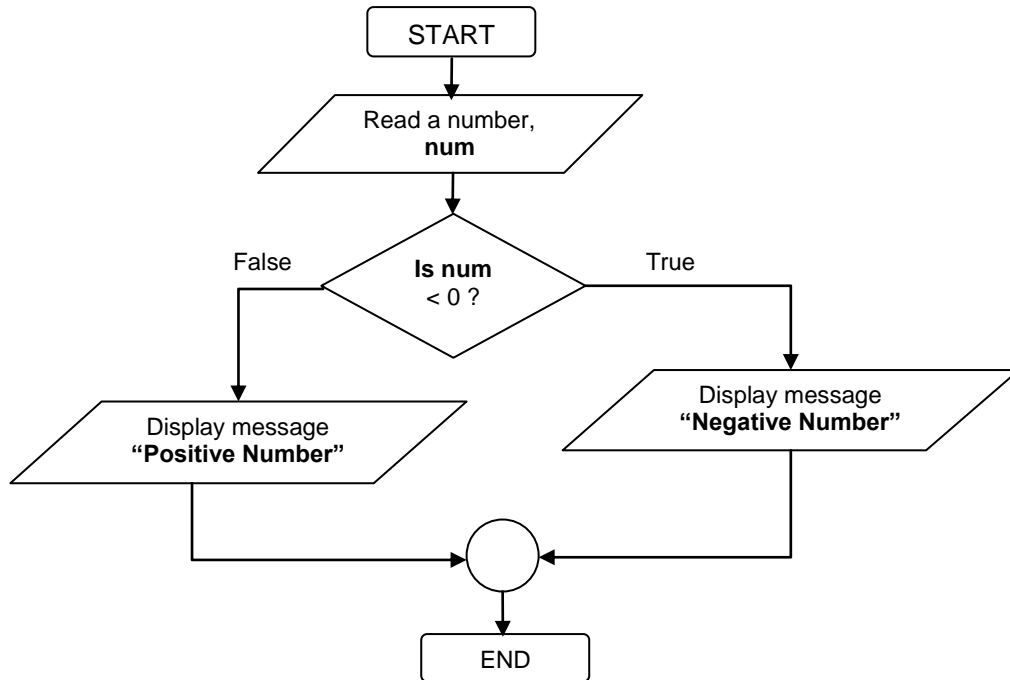


### 3. Write an algorithm and draw flowchart to determine a number whether it is positive or negative.

#### Algorithm:

1. Start.
2. Print "Enter a number which is to be tested for +ve or -ve".

3. Read NUM from keyboard.
4. If  $NUM < 0$ , then Display message “*The number is Negative*” otherwise display message “*The number is positive*”
5. Stop

**Flowchart:**

**Note:** Here zero is assumed +ve number

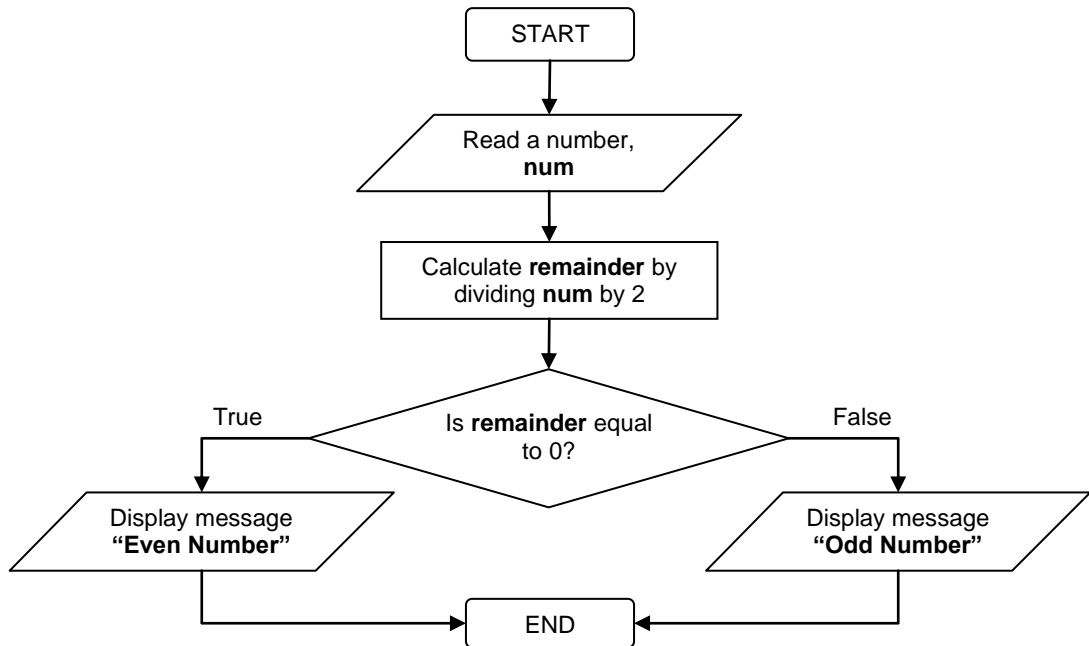
---

**4. Write an algorithm and draw flowchart to test a number for even or odd.**

---

**Algorithm:**

1. Start
2. Display “Enter a number which is to be tested for even or odd”.
3. Read NUM from user.
4. Calculate remainder REM using integer division of NUM by 2.
5. If  $REM = 0$ , then print “The number is even” else print “The number is odd”
6. Stop.

**Flowchart:**

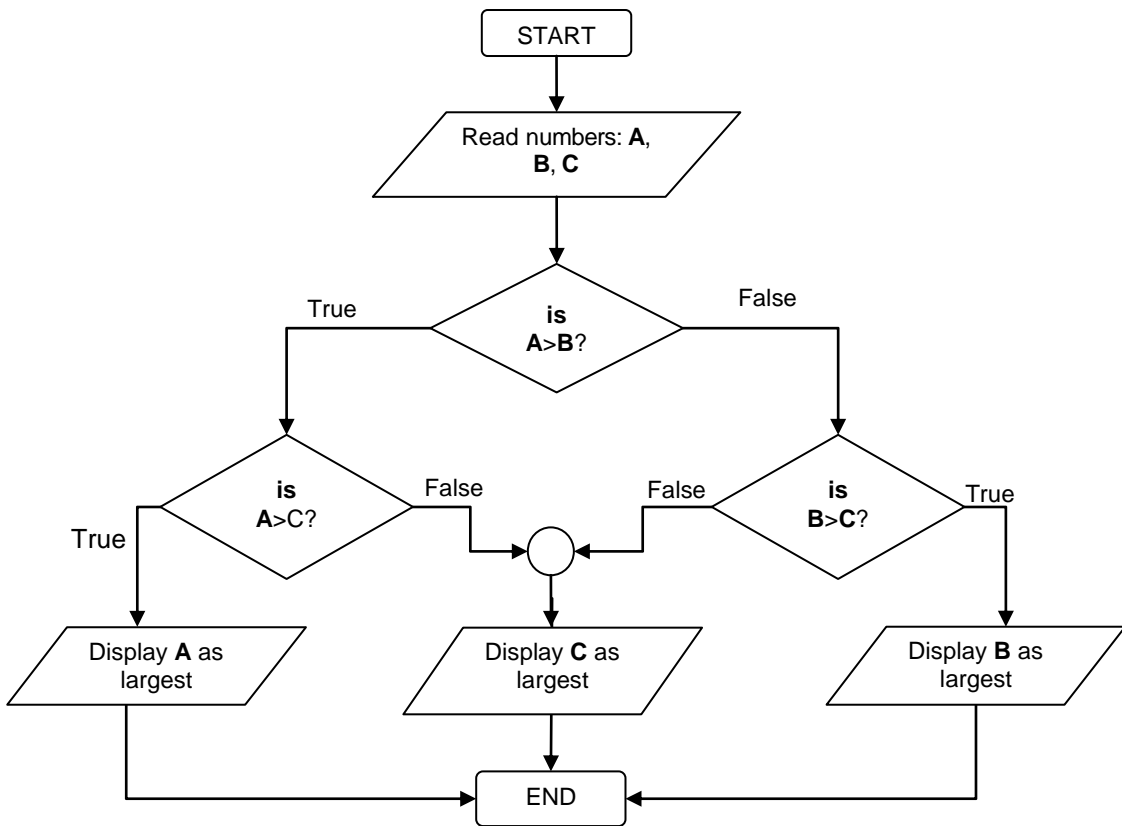

---

**5. Write an algorithm and draw flowchart to find the largest among three numbers.**


---

**Algorithm:**

1. Start
  2. Print "Enter three numbers".
  3. Read three numbers: A, B & C.
  4. If  $A \geq B$  and  $A \geq C$ , then print "A is greatest".
  5. If  $B \geq A$  and  $B \geq C$  then print "B is greatest" else print "C is greatest".
  6. Stop
- Or
1. Start
  2. Print "Enter three numbers".
  3. Read A,B,C
  4. If  $A > B$  then
    - If  $A > C$  then
      - Print "A is greatest"
    - Else
      - Print "C is greatest".
  - Else
    - If  $B > C$  then
      - Print "B is greatest".
    - Else
      - Print "C is greatest"
  5. Stop

**Flowchart:**

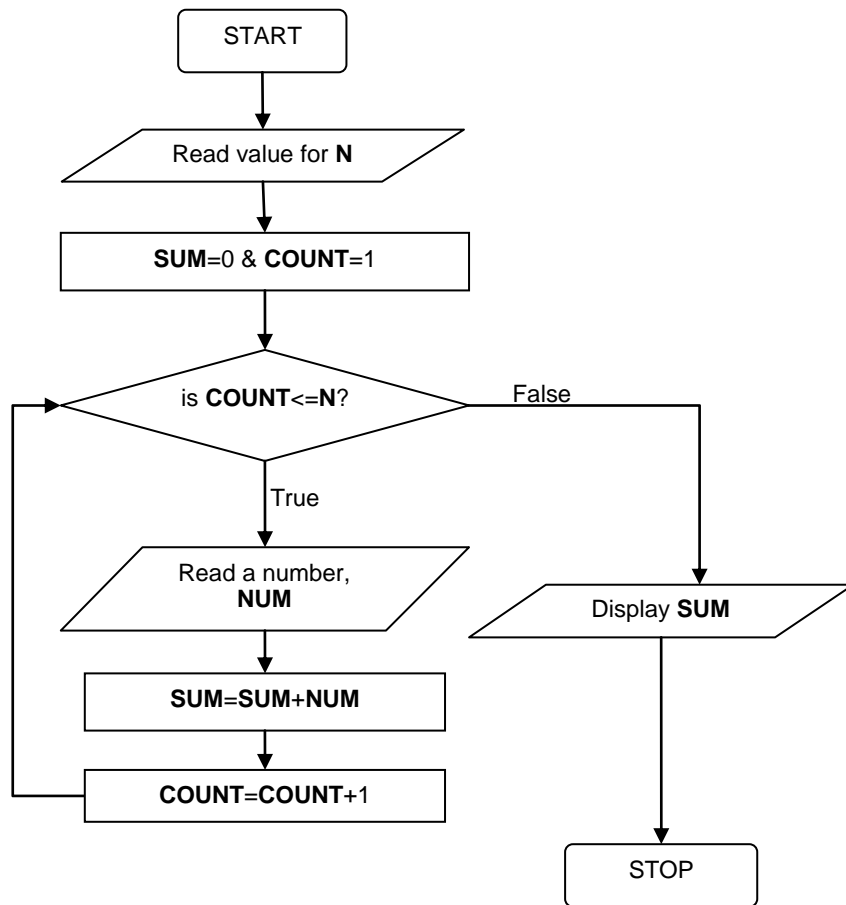

---

**6. Write an algorithm and draw flowchart to read N numbers from user and display sum of all entered numbers.**

---

**Algorithm:**

1. Start
2. Print "How many numbers?"
3. Read N
4. Initialize variables SUM to 0 and COUNTER to 1 i.e. SUM=0 and COUNTER=1.
5. Print "Enter a number".
6. Read NUM.
7. SUM=SUM+NUM
8. COUNTER=COUNTER+1.
9. If COUNTER<=N then goto step 5
10. Print SUM
11. Stop

**Flowchart:**

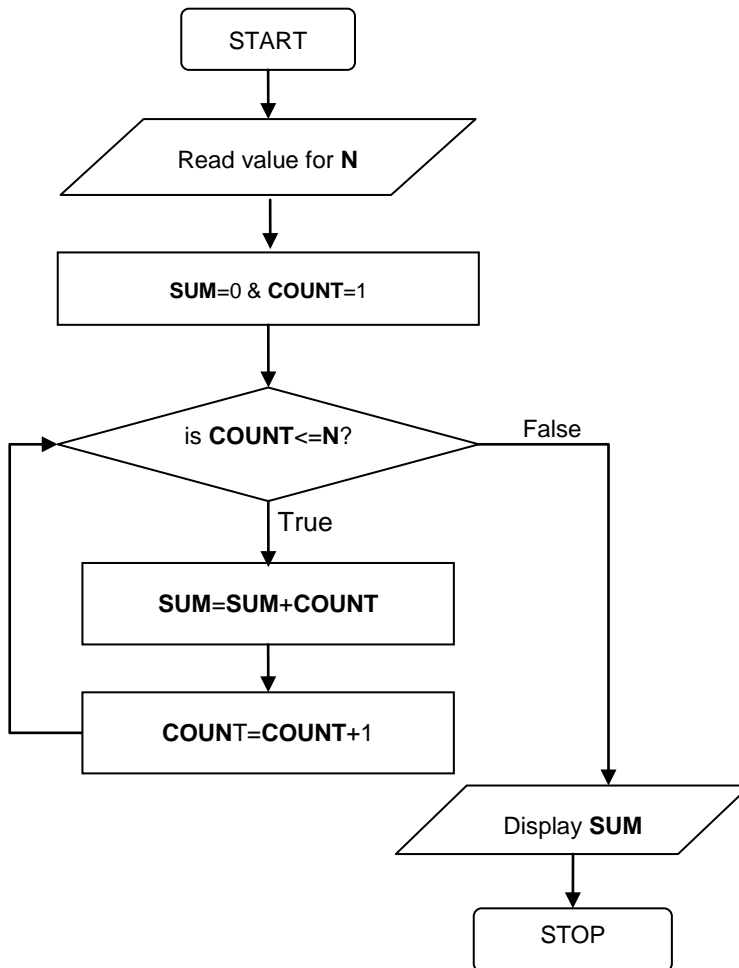

---

**7. Develop an algorithm and draw flowchart for finding the sum of the series  $1+2+3+4+\dots$  upto N terms.**

---

**Algorithm:**

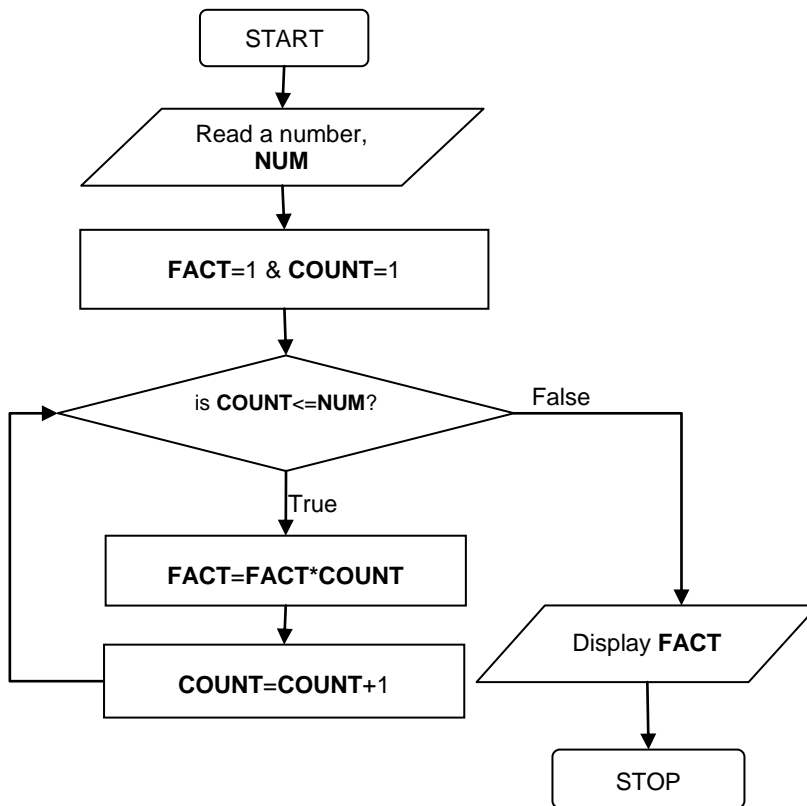
1. Start
2. Print "Enter the value of N".
3. Read N.
4. Initialize variables SUM to 0 and COUNER to 1 i.e.  $SUM=0$  and  $COUNTER=1$ .
5.  $SUM=SUM+COUNTER$ .
6.  $COUNTER=COUNTER+1$ .
7. If  $COUNTER \leq N$  then goto step 5
8. Print SUM.
9. Stop

**Flowchart:**

- 
8. Write an algorithm and draw flowchart for calculating the factorial of a given number N.
- 

**Algorithm:**

1. Start
2. Print "Enter a number".
3. Read NUM.
4. Initialize variables FACT to 1 and COUNTER to 1 i.e. FACT=1 and COUNTER=1.
5. While COUNTER<=NUM  
     FACT=FACT\*COUNTER.  
     COUNTER=COUNTER+1.  
   End of While
6. Print FACT as factorial of the number NUM.
7. Stop

**Flowchart:**

## 1.7 HISTORICAL DEVELOPMENT OF C

The root of all modern languages is ALGOL, It was introduced in the early 1960s and was the first computer language to use a block structure. Martin Richards developed a language primarily for writing system software called BCPL (Basic Combined Programming Language) in 1967. By inheriting the principle features of BCPL, Ken Thomson developed B in 1970. BCPL and B were system programming languages. Inheriting the concepts from ALGOL, BCPL & B, C was developed with added concepts and powerful features by Dennis Ritchie at the bell laboratories in 1972.

The origin of C is closely tied to the development of the UNIX operating system. It was named "C" because many of its features were derived from an earlier language "B". C is a general-purpose, block structured, procedural, imperative computer programming language. Although C was designed as a system implementation language, it is also widely used for applications. C was standardized by ANSI (American National Standard Institute) in 1980.

C was standardized and approved by American National Standard Institute (ANSI) as ANSI C in 1989. It was then standardized by International Standard Organization (ISO) in 1990.

Today, C is running under variety of operating systems and hardware platforms. Some of the most common C compilers are Turbo C/C++ IDE, Borland C/C++, GCC, Microsoft Visual C++, etc.

C++ is an object oriented language based on C was developed in 1990s. Similarly, a pure object oriented language called JAVA was also developed with C++. The few features of C++/JAVA were added to C and was standardized as C99 in 1999.

## 1.8 EXECUTING A C PROGRAM

Executing a program written in C involves a series of steps –

### 1.8.1 Writing a C Program

Computer instructions are written in a text editor to perform certain jobs. These instructions are written using correct syntax of the language we are using. The programming source code can be written using any text editor such as Notepad or text editor of an Integrated Development Environment (IDE) like Turbo C or code::blocks etc. Whichever editor it is, we write the code and save with extension .c like cprogram.c.

```
#include<stdio.h>
int main(void)
{
printf(" welcome to c
programming");
return 0;
}
```

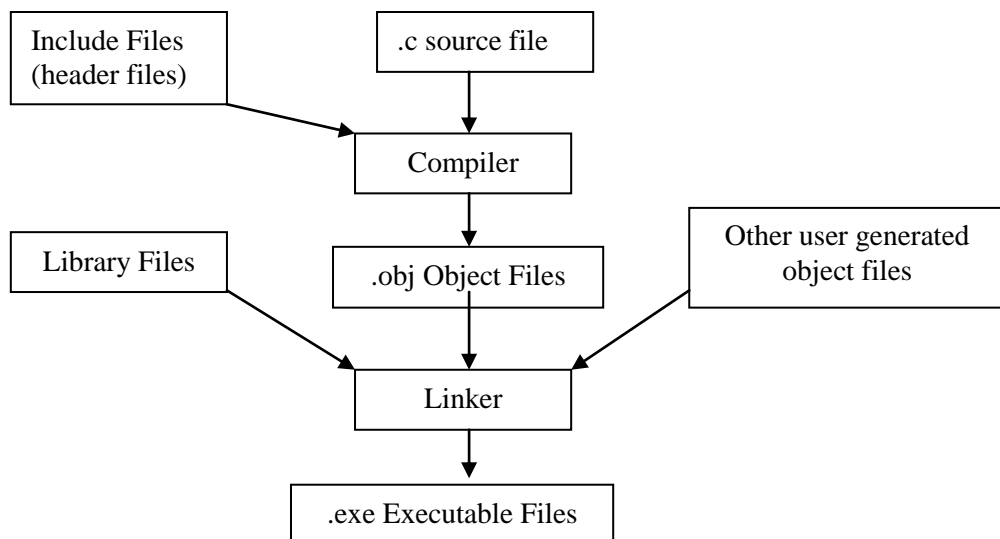
### 1.8.2 Compiling and Linking the program

The computer instructions written in the form of source code are translated into a form that is suitable for execution by the computer. The translation is done by a special program called compiler that processes statements written in programming language and converts them into a machine language or code that a computer's processor uses.

Linker provides instruction to the compiler to link functions with program from the system library. For example, the statement. `#include<stdio.h>` links input/output functions like `printf()` and `scanf()` with the program.

### 1.8.3 Executing the program

Executing the program loads the created executable object code into the computer memory and executes the instruction. Following diagram shows the steps of compilation and linking of C program.



*Fig 1.8.3: compiler and linker relationship*



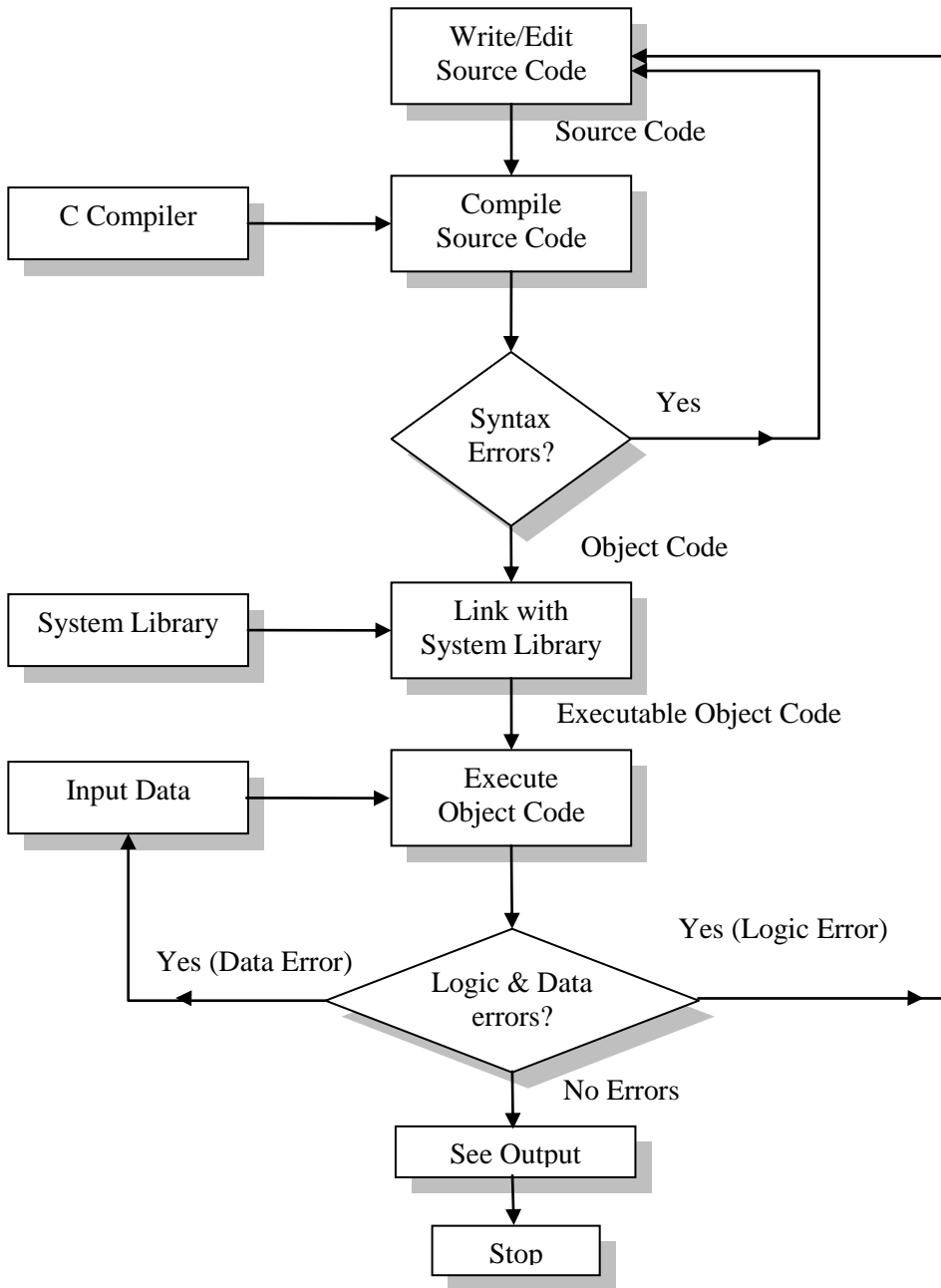
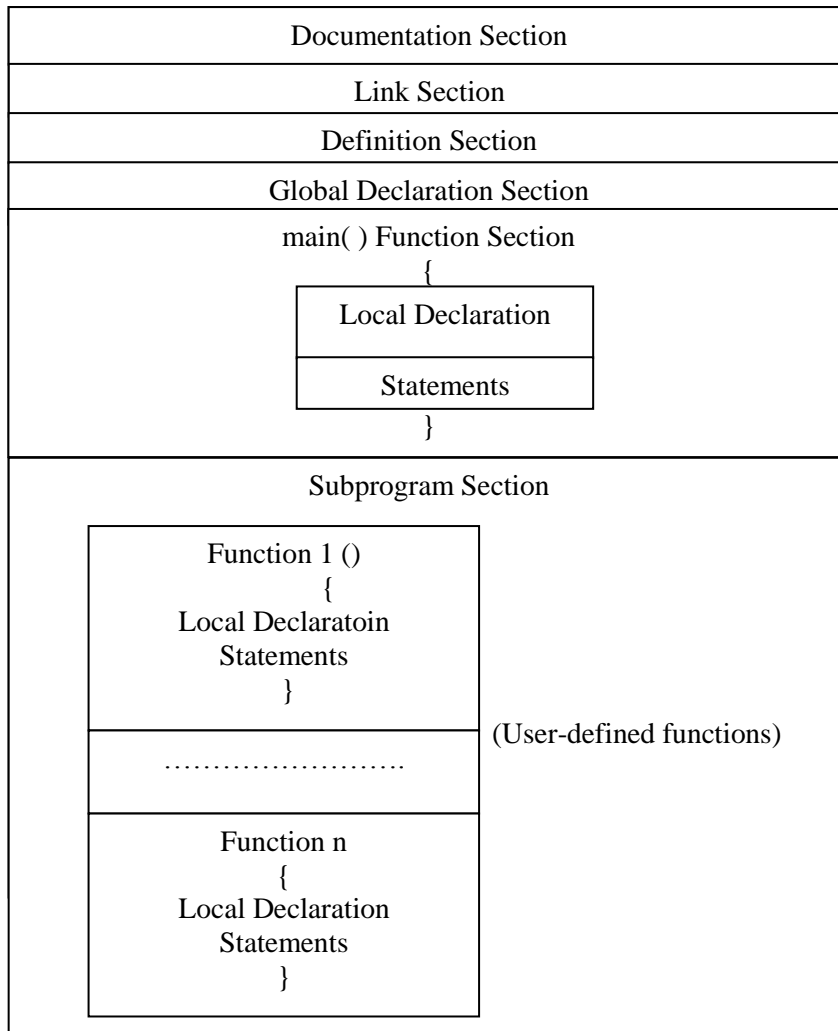


Fig. 1.8.4 Process of compiling and running a C program

## 1.9 BASIC STRUCTURE OF C PROGRAM

The structure of C program implies the composition of a program, i.e. it answers questions such as “what are main components to write a C program? How are they organized?” Fig. 1.9 shows the parts that are included in the structure of a C program –



*Fig. 1.9 Basic structure of a C program*

### i. Documentation Section

This section contains a set of comment lines giving the name of program, the author, algorithms, methods used and other details.

For example, the title of a program can be written in this section as,

```
/* This program displays natural numbers from 1 to 10 */
```

**Note:** /\* ... \*/ denotes multiple line comments in C.

## ii. Link Section

This section provides instruction to the compiler to link functions with program from the system library. For example, the statement.

`#include<stdio.h>` links input/output functions like `printf()` and `scanf()` with the program. This section is also called the header declaration section.

### What is a Header File?

There are several built-in C functions like `printf()`, `scanf()`, `clrscr()`, `getch()` defined in C library as a header file having its `.h` extension. Inclusion of header files in the program is the pre-information to the compiler regarding the library functions used and its relevant codes to be executed. This type of pre-information is called compiler directives or preprocessor directives.

## iii. Definition Section

In this section, all symbolic constants are defined. Symbolic constants are defined in later chapters.

## iv. Global Declaration Section

The variables which are used in more than one functions or blocks are called `global variables`. These variables are defined or declared in this section. This section also declares all the user-defined functions.

## v. Main() Function Section

Every C program starts with a `main()` function. Within `main()` function, there are declaration and executable parts. The *declaration part* declares all the variables used in the execution part.

## vi. Subprogram Section

This section contains all the user-defined functions that are called in the main function.

The sections above the global declaration can be categorized into `preprocessor directives` section. This preprocessor directives contain special instructions that include how to prepare the program for compilation. *Include* is a preprocessor command which informs compiler to include some information from the header file.

All the sections except the `main()` function section may be absent when they are not required. These are demonstrated in the following examples –

```

/* A C program to display "Welcome to C Programming"*/
// Program coded by Hari Pd Sharma
} Documentation
Section

#include<stdio.h>
#include<conio.h>
} Link Section

int main(void)
{
    clrscr();
    printf("Welcome to C Programming!!!!");
    getch();
} main( )
Function
Section

```

Output  
Welcome to C Programming!!!!

**Explanation**

The first two lines

```
/* A C program to display "Welcome to C Programming"*/
// Program coded by Hari Pd Sharma
```

are comments. While compiling the program, the comments are not compiled and thus they are not verified for syntax check. In C, we can perform commenting in two ways like by `/* */` and `//`. We can use `/* */` for multiple line comments and double slash (`//`) for single line comment.

The third and fourth lines

```
#include<stdio.h>
#include<conio.h>
```

These are the preprocessor commands which includes two header files `stdio.h` and `conio.h`. these are the header file inclusion section in which other builtin library functions are to be included to successfully compile the program.

The fifth line: `int main(void)` marks the beginning of the program. All C programs must have at least one main function. It is the entry point of our program. Execution of any program starts from the main function and ends at main function. The word before main is the return type of the function. Here the return type is `int` which means the function does return an integer value to the operating system.

The two curly braces `{ }` are used to group the statements in the function body. The function body contains a set of instruction to perform given task.

The statement `clrscr();`

is the use of library function defined under `"conio.h"` header file. This function is used here to clear old content of screen. In some other program execution environment like `Code::Blocs`, the function like `clrscr()`, `getch()` may not be required as it automatically flush the display screen on its consecutive next run.

The statement `printf("Welcome to C Programming!!!!");`

is used to display `"Welcome to C Programming!!!!"` in the computer screen. The `printf` function is library function used for output declared in `"stdio.h"` header file, so it is included in the program.

Again, the statement `getch()` is not compulsory here. If we have not used this function, we have to press `Alt +F5` to see the output of the program in `TURBO C`. Here, it is important to notice that every C statement ends with semi-colon (i.e. `;`).

---

## 9. Write a program to display " This is your first C lab in the university"

---

```
#include<stdio.h>
#include<conio.h>
int main(void) // main function
{
// opening braces opens the main function body
clrscr();
/*The clrscr() is a library function included in conio.h header file. It
clears the display screen of your output window. The requirements of this
function depends on the use of IDE. Like in TURBO C/C++ IDE, we have to clear
the previous output screen and hence this function is required. But in IDE
like CODE::BLOCKS it is not required. */
```

```
printf("This is your first C lab in the university");
/* printf() function is defined in stdio.h header file. Every statement in C
program should be terminated by semicolon (;) */
return 0;
/*The main function is interger type and has to return integer value to OS
hence this statement is required at the end of every main function if it is
of int.*/
} //closing braces close the main function body
```

**10. Write a program to add two numbers.**

```
/* A C program to add two numbers*/
/* Program coded by Hari Pd Sharma */

#include<stdio.h>
#include<conio.h>
int main(void)
{
    int a,b,sum;
    a = 10;
    b = 20;
    sum = a + b;
    printf("The sum is %d",sum);
    getch();
    return 0;
}
```

} Documentation Section

} Link Section

} main() Function Section

**11. Write a program to calculate area of an ellipse.**

```
//ellipse.c
/*this code once compiled and executed, calculates
the area of an ellipse after reading major and
minor axis*/

#include<stdio.h>
#include<conio.h>

#define Pi 3.1415
float area_of_ellipse(float);
int main (void)
{
    float major, minor, area;
    //clrscr();
    printf("Enter Major and Minor Axis of an llipse:");
    scanf("%f %f", &major, &minor);
    area = area_of_ellipse(major, minor);
    printf("\n The area of an ellipse is: %.2f", area);
    getch();
    return 0;
}
//user defined function
float area_of_ellipse(float a, float b)
{
    return (PI*a*b);
}
```

Documentation section. Comments.

Link Section. (Header Declaration)

Definition Section. Function Prototyping

main() function section

Subprogram Section (User defined function)

What is an IDE?

An Integrated Development Environment (IDE) is a common programming platform that allow computer programmer to invoke all the operations necessary to develop a program, including editing, compiling, linking and program execution. IDE helps us to debug the program and it provides help facilities through menu selection. Forexample TURBO C/C++, CODE::BLOCKS IDE

## 1.10 EXERCISES

---

1. Write algorithm and flowchart to cook half KG rice using electric ricecooker.
2. Write algorithm and flowchart to open your desktop computer.
3. Write algorithm and flowchart to wash your clothese by washing machine.
4. Write algorithm and flowchart to fry chicken in a fly pan.
5. Write algorithm and flowchart to start a motorbike.
6. Write algorithm and flowchart to cook daal (lentil) in pressure cooker.
7. Perform problem analysis, Write algorithm and flowchart to calculate area of isosceles right angle and equilateral triangle.
8. Perform problem analysis, Write algorithm and flowchart to calculate area of parallelogram and rectangle.
9. Write algorithm and flowchart to calculate energy of a capacitor (if C is capacitance and V is the voltage then energy of capacitor is  $(E = \frac{1}{2} CV^2)$ )
10. Write algorithm and flowchart to calculate area of pentagon and octagon.
11. Write algorithm and flowchart to fixed RAM into a system unit.
12. Write algorithm and flowchart to replace 500 GB hard disk into your desktop computer.

