Unit 1: Concept of Algorithm, Flowchart and Languages

Concept of Logic development and Language Fundamental

- Problem Analysis and Needs of Algorithm and Flowchart
- Symbols used to draw flowchart
- Basic examples of Algorithm and Flowchart
- Generation of Computer languages
- High-level and low-level languages
- Translators and editors

Unit – 1

Algorithm

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

Definition of Algorithm

Sequence of instructions that when executed in the specified sequence, the desired results are obtained.

Characteristics of an algorithm

1. Each instruction should be precise and unambiguous.
2. Each instruction should be executed in a finite time.
3. One or more instructions should not be repeated infinitely. This ensure that the algorithm will ultimately terminate.
4. After executing the instructions (when the algorithm terminates), the desired are obtained.

RULES FOR WRITING ALGORITHM:

- START and STOP
- INPUT and OUTPUT
- CONDITION
- LOOP

START and STOP::

- Every algorithm must be begins with a START statement.
- START indicates the starting of an algorithm.
- Every algorithm must be ends with a STOP statement.
- STOP indicates the ending of an algorithm.

INPUT and OUTPUT

INPUT:

An Input (I/P) to the algorithm is indicated by READ statement.

It is used to READ a value of a variable.

E.g. suppose we want to input a number say A to the algorithm is written as

READ A

E.g. suppose we want to input two numbers say A and B to the algorithm are written as

READ A, B

There are two or more variables then its name is separated by comma.

OUTPUT :

An output (O/P) from algorithm is indicated by PRINT statement.

It is used to PRINT the value of a variable.

E.g. suppose we want to display a number say A from the algorithm is by following statement

PRINT A

SEQUENTIAL LOGIC.. Examples

Eg1 . Write an algorithm to find sum of two integers.

1. START
2. READ a,b
3. c= a + b
4. PRINT "Sum is ", c
5. STOP

Eg2. Write an algorithm to find Simple Interest

1. START
2. READ p,r,n
3. si = (p * r * n) / 100
4. PRINT "Simple Interest is ", si
5. STOP

CONDITION.. Examples

- A condition statement is used when there are two possibilities of either YES (TRUE) or NO (FALSE).

- A condition and their statement is represented by IF…THEN…ELSE statement.

Eg3. Write an algorithm to check whether the given number is odd or even

1. START
2. READ n
3. IF N MOD 2 = 0 THEN
   PRINT "Number is Even"
   ELSE
   PRINT "Number is Even"
   ENDIF
4. STOP

LOOP.. Examples

- A looping statement is used when there are possibilities of performing steps for the fixed numbers of times.

- A looping of a statement is represented by DO…WHILE statement.

Eg4. Write an algorithm to PRINT 1 to 5 numbers using DO...WHILE

1. START
2. I = 1
3. DO WHILE I < = 5
   PRINT I
   I = I + 1
   END DO
4. STOP

Eg5. Write an algorithm to PRINT factorial for the given number N.

Input : N = 4

Output : 24 (Hint : 1*2*3*4)

```
Step – 1   START
       2 :   READ   N
       3 :   I = 1, F = 1
       4 :   DO WHILE I <= N
                   F = F * I
                   I = I + 1
             ENDDO
       5 :   PRINT  "Factorial  is ", F
       6 :   STOP
```

Eg6. Write an algorithm to PRINT following series using DO…WHILE

1 2 3 4 5 . . . upto N

1. START
2. READ N
3. I = 1
4. DO WHILE I < = N
   PRINT I
   I = I + 1
   ENDDO
5. STOP

Eg7. Write an algorithm to check whether the given number is Armstrong or not.

Input : N = 153

Output : Armstrong Number

(Hint : 153 is an Armstrong because 153 = $1^3 + 5^3 + 3^3$)

```
Step – 1:   START
        2 :  READ   N
        3 :  SUM = 0, M = N
        4 :  DO WHILE >0
                    R = N % 10
                    SUM = SUM + R * R * R
                     N = N / 10
             ENDDO
        5 :  IF ( M = SUM ) THEN
                     WRITE 'Given number is Armstrong'
             ELSE
                     WRITE 'Given number is NOT Armstrong'
             ENDIF
        6 :  STOP
```

Eg8. Write an algorithm to check whether the given number is palindrome or not.

Input : N = 12321

Output : It is palindrome number.

(Hint : A number is said to be a palindrome number if its original number and its reverse number both are same)

```
Step – 1 :  START
        2 :  READ   N
        3 :  REV = 0, M = N
        4 :  DO WHILE N > 0
                    D = N % 10
                    REV = REV * 10 + D
                     N = N / 10
             ENDDO
        5 :  IF( M = REV) THEN
                     WRITE 'Palindrome'
             ELSE
                    WRITE 'Not Palindrome'
```

```
        ENDIF
    6 : STOP
```

Eg9. Write an algorithm to check whether the given number is prime number or not.

Input : N = 11

Output : It is prime number.

(Hint : A number is said to be a prime number if it is only divisible by 1 and itself)

```
Step – 1 :  START
     2 :  READ  N
     3 :  I = 2, FLAG = 0
     4 :  DO WHILE I < N
          IF ( ( N % I ) = 0 ) THEN
                FLAG = 1
                GOTO STEP 5
          ELSE
               I = I + 1
          ENDIF
     5 :  IF ( FLAG = 0 ) THEN
                PRINT 'Prime Number'
          ELSE
                PRINT 'Not Prime Number'
          ENDIF
     6:   STOP
```

Eg10.Write an algorithm to PRINT Fibonacci series for the given number N.

Input : N = 7

Output : 1 1 2 3 5 8 13

```
Step – 1  START
      :
     2 :  READ  N
     3 :  I = 1, A = 1, B = 0
          DO WHILE I <= N
               C = A + B
               A = B
               B = C
               PRINT C
               I  = I + 1
          ENDDO
     4 :  STOP
```

ADVANTAGES OF ALGORITHM

1. It is easy to understand and easy to write.
2. It is an effective method of solving certain sets of problems.
3. No need to knowledge of a specific programming language
4. It is easy to detect and solved the mistake from the algorithm.

DISADVANTAGES OF ALGORITHM

1. It is time consuming process.
2. There is only a manual method to check whether an algorithm is correct or not.

FLOW CHART

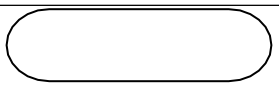
DEFINATION:

A flowchart is a pictorial representation of an algorithm.

Need of flowchart

- While drawing a flowchart the programmer need not pay attention to the details of the elements of the programming language. Hence, can fully concentrate on the logic of the solution to the given problem.
- Flow of operations in the pictorial representation helps the programmer to easily detect any error in the logic of the program.
- Once the flowchart is ready the programmer can concentrate on the operations in each box of the flowchart and ensure error free program development.
- Serves as a document for the computer program.
- Useful for testing and modifications in program.

Symbols used to draw Flow Chart

1. Terminal : 

    - The terminal symbol indicates the beginning (Start), end (Stop), and pauses (Halt) in a program's logic flow.
    - If the program logic has a pause, that is also indicated with a terminal symbol.
    - A pause is normally used in a program logic under some error conditions, or if forms had to be changed in the computer's printing during the processing of the program.

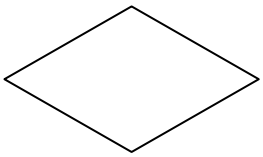
2. **Input – Output :**

- The input/output symbol denotes any function of an input/output nature in a program.
- Hence, all program instructions to input/output data from any type of input/output device (such as keyboard, mouse, scanner, monitor, printer, plotter, etc.) are indicated with input/output symbols in a flowchart.
- Even instructions to input/output data from/to a storage device (such as disk, tape, etc.) are indicated with input/output symbols.

3. **Processing:**

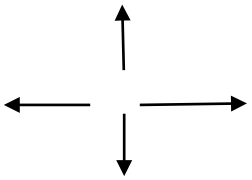
- A processing symbol represents arithmetic and data movement instructions. Hence, all, arithmetic processes of adding, subtracting, multiplying, and dividing are indicated by a processing symbol in a flowchart.
- When more than one arithmetic and data movement instructions are executed consecutively, they are normally placed in the same processing box, and are executed in sequence.

4. **Decision.**



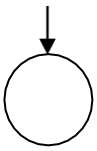
- The decision symbol indicates a decision point, that is, a point at which a branch to one of two or more alternative points is possible.
- The criterion for making a decision is indicated within the decision box.
- Exit paths should be identified and considered.
- During execution, the appropriate path is followed depending upon the result of the decision.

5. **Flow lines.**



- Flow lines with arrowheads indicate the flow of operation, that is, the exact sequence in which the instructions are executed.
- The normal flow of a flowchart is from top to bottom and left to right.
- Arrowheads are required only when the normal flow is not followed. Flow lines are usually drawn with an arrowhead at the point of entry to all flowchart symbols.

6. **Connector :**



- Whenever a flowchart becomes so complex that the number and direction of flow lines are confusing, or it spreads over more than one page, it is useful to utilize connector symbols as a substitute for flow lines.
- This symbol represents an entry from, or an exit to another part of the flowchart.
- A connector symbol is a circle with a letter or digit placed within it to indicate the link.
- A pair of identically labeled connector symbols is used to indicate a continued flow when the use of a line is confusing. Hence, two connectors with identical labels serve the same function as a long flow line.
- If an arrow enters but does not leave a connector it is an exit point, and program control is transferred to the identically labeled connector having an outlet.
- Connectors help in simplifying the task of drawing large flowcharts that spread over several pages.

Rules of drawing flowchart

1. First chart the main line of logic, and then incorporate details.
2. Maintain a consistent level of details for a given flowchart.
3. Do not chart every detail; otherwise, the flowchart will only be a graphic representation of all the steps of the corresponding program. A reader interested in details can refer to the program itself.
4. Use common statements that are easy to understand for words within flowchart symbols. Descriptive titles written in designer's own language should be used, rather than machine-oriented language.
5. Be consistent in using names and variables in the flowchart.
6. Go from left to right and top to bottom in constructing flowcharts.
7. Keep the flowchart as simple as possible. Crossing of flow lines should be avoided.
8. If a new flowcharting page is needed, break the flowchart at an input or output point. Moreover, use properly labeled connectors to link the portions of the flowchart on different pages.

Advantages of Flowcharts

1. **Better Communication.** A flowchart is a pictorial representation of a program. Therefore, it is easier for a programmer to explain the logic of a program to some other programmer, or to his/her boss through a flowchart rather than the program itself.

2. **Proper Program Documentation.** Program documentation involves collecting, organizing, storing, and otherwise maintaining a complete historical record of programs, and other documents associated with a system.

   Good documentation is needed for the following reasons:

   (a) Documented knowledge belongs to an organization, and does not disappear with the departure (resignation/retirement) of a programmer.

(b) If a project is postponed, documented work helps in restarting the project later from the stage at which it was stopped.

(c) If a programmer has to modify a program, documented work provides him/her a more understandable record of what was originally done.

Flowcharts often provide valuable documentation support.

3. **Efficient Coding.** Once a flowchart is ready, programmers find it very easy to write the corresponding program because the flowchart acts as a road map for them. It guides them to go from the starting point of the program to the final point, ensuring that no steps are omitted. The ultimate result is an error-free program developed at a faster rate.

4. **Systematic Debugging.** A flowchart is very helpful in detecting, locating, and removing mistakes (bugs) in a program in a systematic manner because programmer find it easier to follow the logic of the program in flowchart form. The process of removing errors (bugs) in a program is known debugging.

5. **Systematic Testing.** Testing is the process of confirming whether a program will successfully do all its intended jobs under the specified constraints. For testing a program , the program is executed with different sets of data as input to test the different paths in the program logic.

A flowchart proves to be very helpful in designing the test data for systematic testing of programs.

Disadvantages of flowchart

1. Flowcharts are very time consuming and laborious to draw with proper symbols and spacing, especially for large complex programs.

2. Nature of flowchart is of Symbol-String type, so to make any change or modifications in the program logic will usually require a completely new flowchart.

   Redrawing flowchart is a tedious task, programmers may not redraw or modify flowcharts when they modify programs. Due to this programs and flowcharts will be inconsistent. Thus logic of the program and flowchart will not match.

   This defeats the purpose of using flowcharts as documentation support for programs.

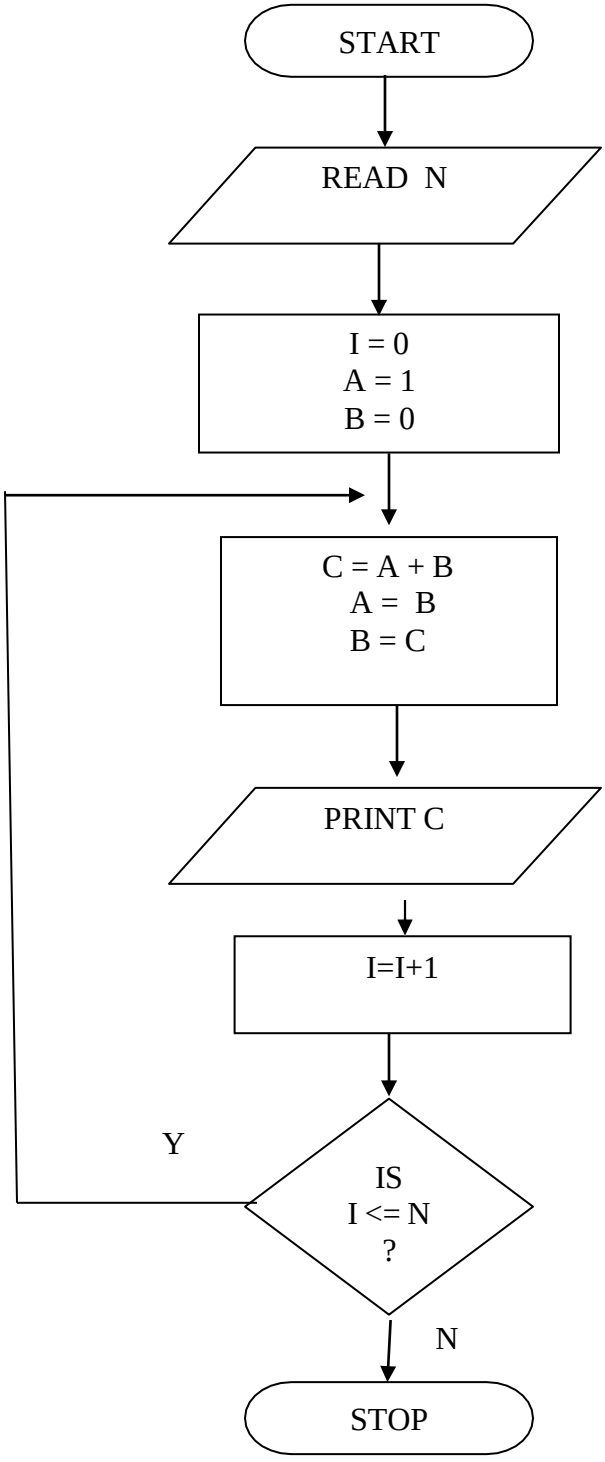
   Due to this problem many companies automatically generate flowcharts directly from the code.

3. There are no standards determining the amount of detail that should be included in a flowchart.

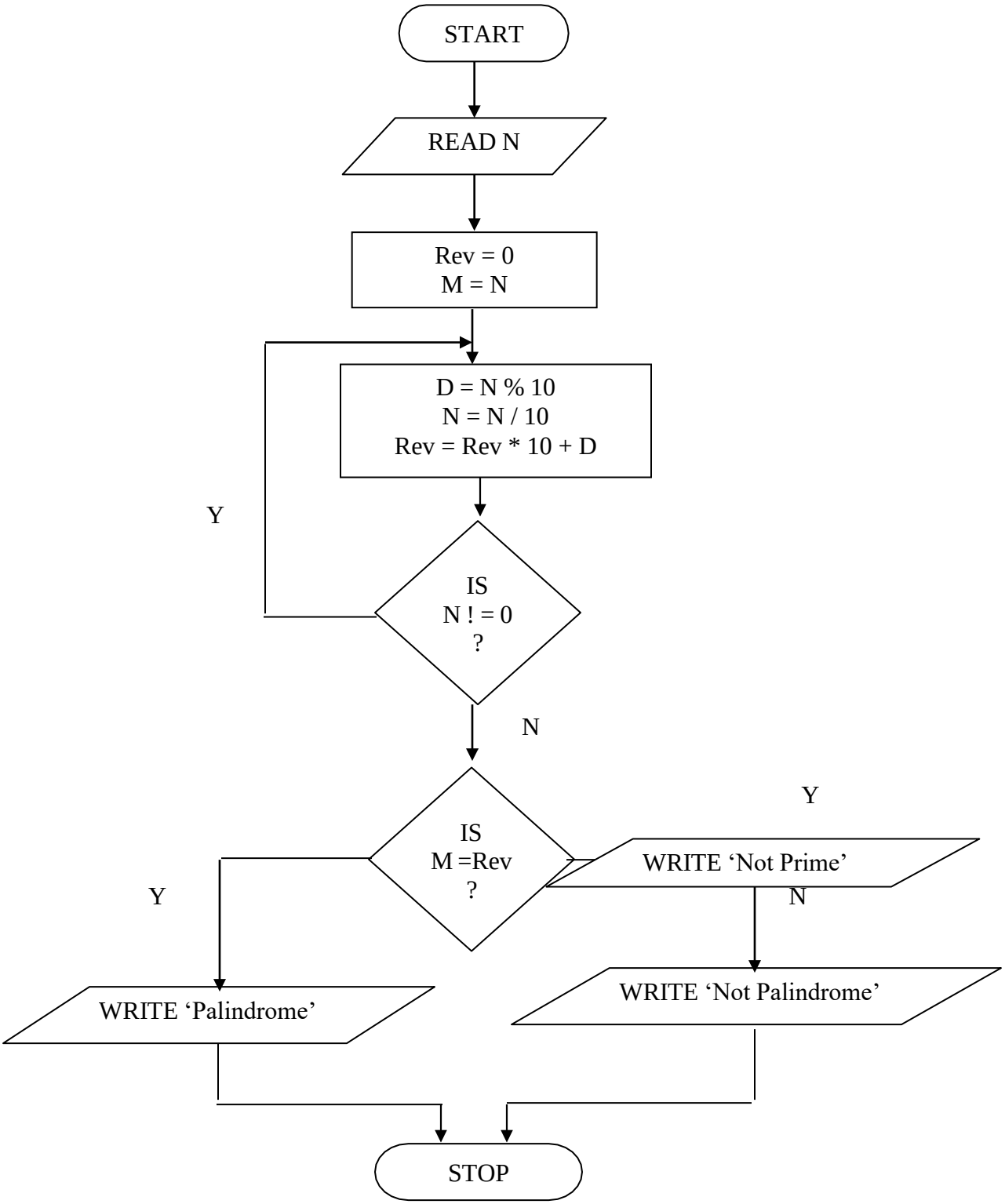
Draw a flowchart to PRINT Fibonacci series for integer N.

Input : N = 7

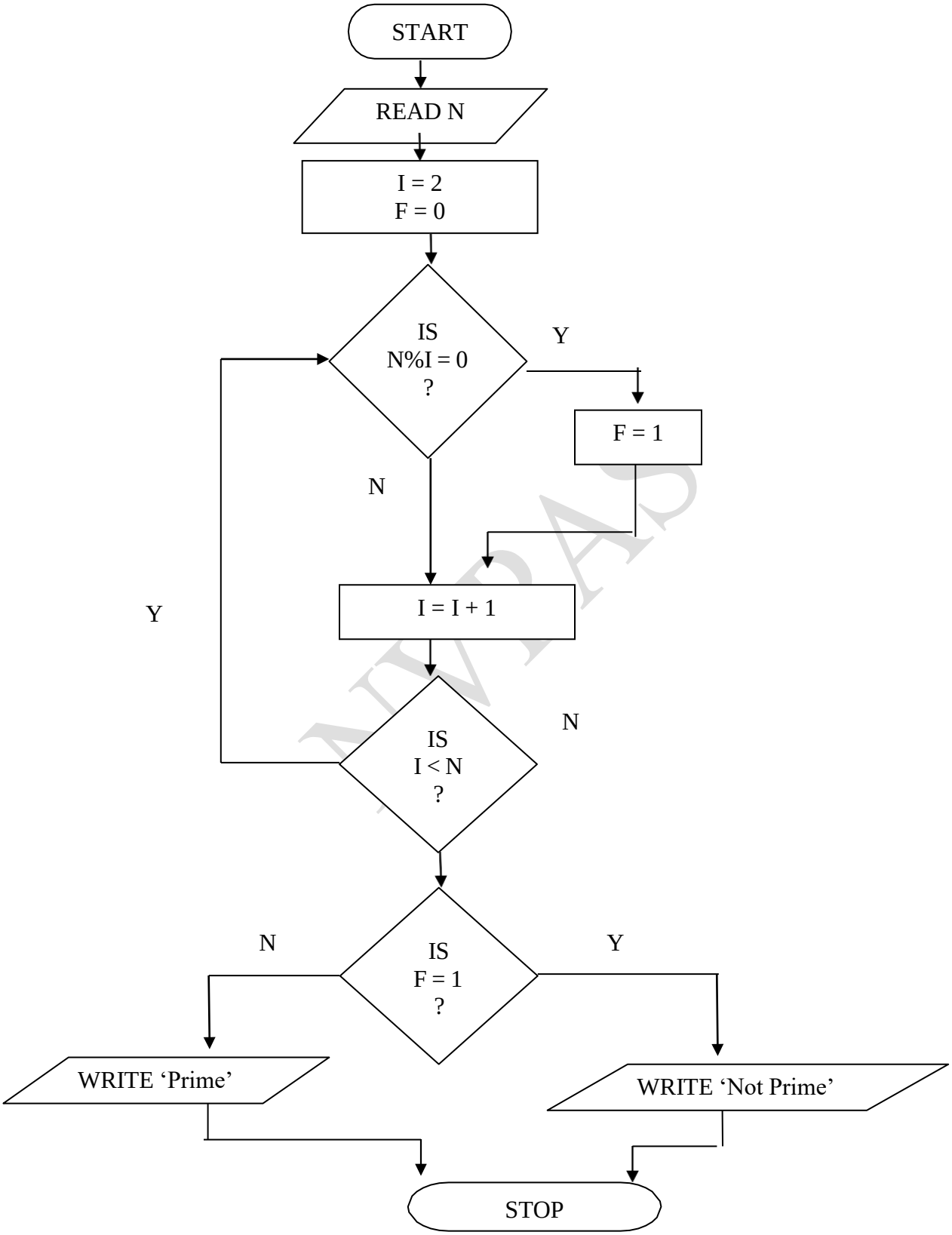
Output : 1 1 2 3 5 8 13



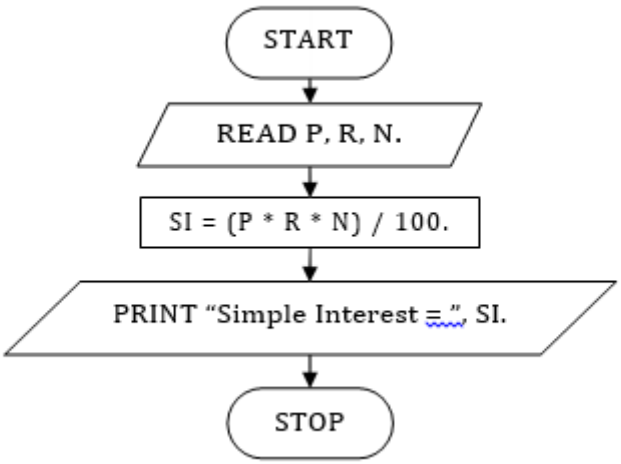
Draw a flowchart to check whether the given number is Palindrome number or not.



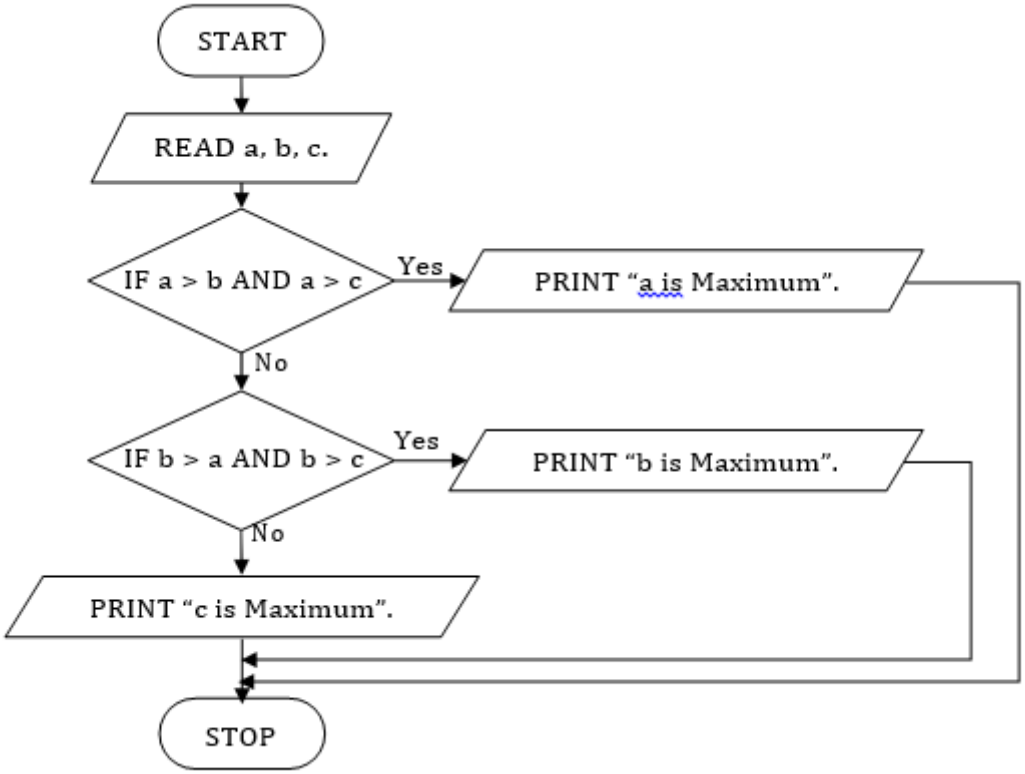
Draw a flow chart to check whether the given number is Prime number or not..



Draw Flow Chart to Find Simple Interest



Draw Flow Chart to Find Maximum of 3 Numbers



--

Write algorithm and draw flowchart for following definitions:

1. To find simple interest. Hint: SI = (P * R * N)/100

2. To find maximum of given three numbers.

3. To find out N! (Factorial of N).

4. To find sum of odd value and even value digits of a given number.

5. To find out minimum from N numbers.

6. To check whether inputted number is prime number or not.

7. To check whether inputted number is palindrome number or not.

8. To check whether inputted number is Armstrong number or not.

Hint: The number whose sum of cube of individual digit is same as number itself.

eg. $153 = 1^3 + 5^3 + 3^3$

9. To print N terms of Fibonacci series.

Input: N = 9

Output: Fibonacci series: 1 1 2 3 5 8 13 21

10. sum=$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 + \ldots$ and so on.

GENERATION OF COMPUTER LANGUAGES (1GL, 2GL, 3GL & 4GL)

1st Generation (1GL):

- The period for 1GL was 1950-1958.
- Uses Bottom – UP Technology.
- Languages : FORTRAN, FlowMatic (Data Processing) & IPl5 (List processing) were used.
- High-level language concepts of simple action (control mechanism) like statements & expressions were introduced with FORTRAN I. It also introduces iteration (Do loops), conditional flow control transfer GOTO m, If and use of labels. FORTRAN I allowed subroutines to be complied independently. The language was so created to generate efficient machine code.
- Static storage allocation i.e. size of variable was reserved.

2nd Generation (2GL):

- The period was 1957 to 1961.
- A Top-Down approach was used i.e. emphasized the needs of programming from programmer's viewpoint.
- Concept of nested block structures, scope rule of variables, explicit type declaration for variables, call by value/name for passing procedure parameters, If-Then-Else selection statements, For & While, Switch concept, own variables and three data type int, real and Boolean.
- Languages : ALGOL 60 in 57 to 60, COBOL (COmman Business Oriented Language) in 59-61 - business oriented, it was divided into divisions, it was machine independent and has its own compiler. LISP (LISt Processing) 59-61, it was (an appreciative language) designed for non-numeric processing. Used widely in Artificial Intelligence applications, FORTRAN II.
- It introduced the use of dynamic storage allocation and recursions.

3rdGeneration (3GL):

- The period was 1960 to 1970.
- Languages : PUI, ALGOL68, Pascal, SIMULA69, BASIC, C were developed. PUI 64 – 69.
- It is a general-purpose language *to* satisfy for commercial areas, system programmers, and scientific needs. It includes ideas of separate compilation, subroutines, sharing common data, Do Loop from FORTRAN. It took i/o facilities, report generating facilities from COBOL. It adopts arrays, recursion from ALGOL 60. PL/I introduced Paralleled processing.
- ALGOL 60: served as basis for ALGOL W, ALGOL 68, SIMULA67, and Pascal. It removed use of labels as parameters of procedure and CASE statement was introduced.
- ALGOL 68 (1963-69): It supports pointers, records, and Union type. It allows the user to define their own data types.
- Pascal: Around 1970 - apart from other features included range types and enumerated types, allowed passing procedures as parameters.
- SIMULA67: Process oriented language in 1967. It introduced concept of Class and co-routines.
- Others were APL- Math's applications, SNOBOL4 - string manipulation language, BASIC – 1st language for beginning programmers (encouraged structure programming).

4th Generation (4GL):

- 1970s most 4GL's evaluated from Pascal.
- Languages were CLU, ALPHARD, EUCLID, MODULA,ADA.
- Main features of these languages were modularity by means of abstraction. Abstraction involves hiding implementation details from users. Modularity permits the specification of well-defined interfaces among program modules. They also support concurrency for parallel processing.

Fourth generation languages allows users to create programs with less effort than is required by high level languages. Also known as non-procedural languages.

The objectives of 4GL include:

1. Increases the speed of developing programs.
2. Minimize user's efforts to obtain information from computer.
3. Decrease skill level required of users.
4. Reduce errors or make programs that are easy to change.

Depending on the language, the difficulty of 4GL varies widely. These languages are usually used in combination with a database.

4GL include: data query languages, report generators and application generators.

A database query language permits formulation of queries that may relate to several records from one or more files. The appropriate records can then be printed or displayed in a suitable format. E.g. IBM's SQL.

A report generator allows data from a database to be extracted and formatted in to reports. It also allows substantial arithmetic and logic operations to be performed on the data before they are displayed or printed.

An Introduction to Computer Languages :

A Language that is used to communicate between people and the computer is known as Computer Language.

It is a language that is understood by the computer.

All computer languages can be classified following three categories:

1. Machine Language
2. Assembly Language
3. High-level Language

Machine Language

These language could only run on the machine for which they are designed.

A typical machine language instruction consist of 3 components:

1. An operation code.
2. One or more register numbers and
3. One or more data addresses.

In general, an instruction prepared in machine language has a two - part as shown below.

OPCODE	OPERAND
Operation Code	Address / Location

Fig. Instruction Format

The first part is of the instruction is OPCODE i. e. Operation Code. It tells the computer what function to perform. Every computer has an opcode for each of its instructions.

The second part of the instruction is the operand. It tells the computer where to find or store the data or other instructions that are to be manipulated.

Since program written in this language is made up of simply O's and 1l's, it is not easy language to learn because there are thousands of binary instruction codes to remember. And program written using this language is very difficult to read and understand.

The operation code tells the control unit what data processing operation is to be performed. Eg: transfer of data from one storage location to another the register number of instruction signifies which register(s) to use when computational procedures such as addition or multiplication of 2 numbers are required. When writing programs in machine language, it is programmer's

responsibility to keep track of register usage. The purpose of data addresses is to computer a specific data address for finding a particular data item in memory.

Machine language instructions are expressed directly in binary code i.e. 0's and 1's or indirectly in binary code using octal or hexadecimal number system. The use of binary representation makes machine language program easy to store since the primary memory of all digital computers store binary digit.

Machine languages are primitive, requiring the programmer to write separate instruction for each computational step. Eg: x= a * (b-c)/d.

1. Transfer the value of b in register 1.
2. Subtract c from register 1.
3. Multiply register 1 by a.
4. Divide register 1 by d.
5. Store the results in x.

Advantages:

1. **Makes efficient use of computer storage:** A programmer completely controls the language instructions and their storage in computer memory. Thus. Language instructions are not duplicated.

2. **No need of Translation or compilation :** The instructions of machine level languages are immediately executable. They require no complication or translation step, as required by other programming languages.

Disadvantages:

1. **They are machine dependent:** Because operation codes and register usage differ form computer to computer, making standardization impossible. An internal design of a computer is different from one computer to another computer. So, a program written in machine language can not properly work on all the computers.
2. **Poor Productivity :** Programming in machine language usually results in poor programmer productivity. This is due to level of detail required

and tedious use of binary, octal, hexadecimal number system when writing machine language instruction.

3. **Use of Registers :** Machine language requires programmers to control the use of each register in the computer's ALU. This is very cumbersome task.

4. **Direct Addressing :** Computer storage locations must be addressed directly not symbolical. i.e. for any given computer program, the programmer must remember the physical address of potentially hundreds of data values.

5. **Difficult to program requires High Skill :** Machine language requires high level of programming skill because it is necessary for the programmer to remember the dozens of OPCODE and OPERAND numbers.. This increases programmer training cost.
6. **Error prone:** A program written in machine language is only consists of series of 0 and 1. There are so many possibilities of errors in such programs. It becomes very difficult for programmers to concentrate the logic of the program. This results in program errors.
7. **Difficult to modify**: It is difficult to correct or modify machine language programs. It is very difficult to locate errors from the program because it contains only a sequence of digits.
8. **Time Consuming :** writing a program in machine language is so difficult and time consuming that it is rarely used today.

The first generation language was machine language. The idea of writing programs in binary code was given by famous mathematician John Van Newmann.

Assembly Language:

Machine language programs are easily stored in the memory of computers they are not easily programmed by humans. Writing instruction is 0's and 1's is boring, tedious and prone to error. Thus to improve programmer's productivity, second generation programming language called assembly language were developed. This programming language substitute alphabetic or numeric symbols for the binary code of machine languages, thereby enabling programmers to express computer instructions more easily.

Assembly language instructions consists of 4 components:

1. **A name of label field** – It is optional. It is used to mark a place in program to which computer control can transfer.
2. **An operation code** – it tells the control unit what data processing task to perform.
3. **An Operand -** It identifies the location of data values in primary memory. Its functions is same as of data address of machine language.
4. **A Comment field** – It enables a programmer to write notes and therefore document the program. These comments are optimal and computer ignores it while execution.

Assembly language use symbols rather than absolute address to represent memory locations. Eg: letter A is used to represent the memory location of one value in the formulae. Similarly a programmer can use the notation "R1" and "R2" ,"Reg1" & "Reg2" or "ACCUM1" & "ACCUM2" to represent the register in the computers ALU.

Assembly language use mnemonics for the operation code- i.e single letters or short abbreviations that help programmers remember what the code represents.

Name	Operation Code	Operand	Comments
Begin	L	R1,B	Transfer the value of B to register 1.
	S	R1,C	Subtract C from register 1.
	M	R1,A	Multiply register 1 by A.
	D	R1,D	Divide register 1 by D
	ST	R1,E	Store the results of register 1 in E

Here, L stands for load to a register form primary memory and M stands for multiply a register by storage value. The use of mnemonics makes Assembly language much easier to code instructions and debug computer programs then in machine language as a result programmers make fewer errors and are able to work longer without tiring.

Assembly language programs are not immediately executed by a computer so the instructions must first be translated into machine language by another program called assembler. So assembler transforms the source program written in assembly language to object programs which is a machine language program.

Advantages

1. **Easy to use :** Assembly language is easier to use than machine language because here symbolic storage names and mnemonics are used instead of binary codes. Easier coding increases programmer's productivity.

2. **Check error easily :** An assembler is useful for checking programming errors. As the assembler performs its translation process, it can also check for proper grammatical usage and language consistency.

3. **Symbolic Addressing :** Programmers do not have to remember the addresses of data values. A programmer simply uses familiar symbols to represent data values, and the assembler takes care of assigning physical address in memory to those symbol at the time the program is translated into machine language.

4. **Modular programming :** Assembly language encourages programming in modules. i.e. A large program can be divided into smaller pieces or modules and a separate programmer can be assigned to each module.

Disadvantages:

1. **Translation is required :** Assembly programs are not immediately executable, so they must first be translated into machine language by an assembler. This process takes time.

2. **Step by step instructions :** Assembly language program instructions must be expresses separately. i.e. the programmers have to think like calculators. So it must break formulas and other data processing tasks into simple steps and program each step separately.

3. **Machine dependent :** Assembly language is machine dependent. Assembly program written for one type of computer will not run on another type of computer.

4. **High Skill :** Programming in assembly language requires high level of programming skill.

- It is difficult to remembers all the symbols used in assembly languages. A new language is introduced that is a high level language that is English like language.
-
-

High level language

- A **high level language** has some rules and regulations for writing, which is known as syntax. A compiler is used to check the syntax of the program, written in high-level language.
- A program written in high level language cannot understand directly by machine. So, this program has to translate into its equivalent machine code, so that machine can execute the instructions. This translation can be done by compiler.
- The program written in high level language is called as source program.
- A source program has been converted into machine language by compiler is known as object program.

- Examples of high level languages are Pascal, Fortran, Cobol, Basic, C, C++, etc.

Advantages:

- High-level languages are Machine Independent. In other words, a program written in a High-level language can be run on many different types of computers.
- It is easy to learn and use.
- It is very easy to locate and correct errors.
- Less Time and Effort is required for writing program.

Disadvantage:

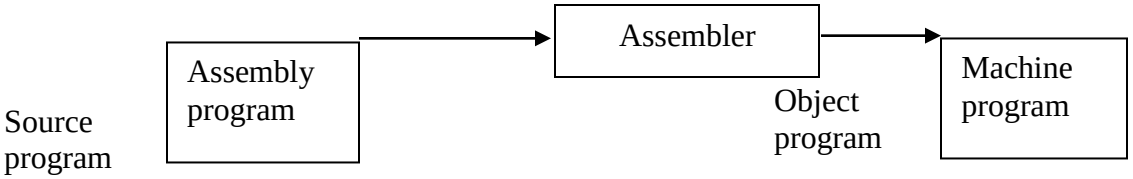
- Program written in a high-level language requires more time to run compare to machine and assembly languages program.
- Slower execution compare to machine and assembly language program.

Translator:

- Computer understands only one language, i.e. Machine Language. Therefore, before executing a program written in a high-level language or an assembly language, it is necessary to translate them into machine language. This will be done by translator.
- **Definition: A translator is a program that converts the high level language program or assembly program into machine level language program.**
- Examples of translator are:
  - Assembler
  - Compiler
  - Interpreter.

Assembler

- The assembly language code cannot be executed by a machine directly as it is not in a binary form.
- **What is Assembler ? - An Assembler is translator that converts an assembly language program into machine language program.**
- The process of translating an assembly language program into its equivalent machine language program with the use of an assembler is illustrated in the below figure:



- A symbolic program written by the programmer in assembly language is called a **source program.** After the source program has been converted into machine language by an assembler, it is referred to as **an object program.**
- The input to an assembler is a source program written in assembly language and its output is an object program which is in machine language.

- Since the assembler translates each assembly language instructions into its equivalent machine language instructions, there is one-to-one correspondence between the assembly instructions of object programs.
- When we write a program in symbolic language, we first run the assembler program.

Compiler

- The high level language program cannot be executed by a machine directly as it is not in a binary form.
- **What is a Compiler ? : A translator program that translates a high level language code into the computer's machine code is called a Compiler.**
- The program written in high level language is called as **source program.**
- A source program has been converted into machine language by assembler is known as **object program.**
- Working of compiler is illustrated in below figure:

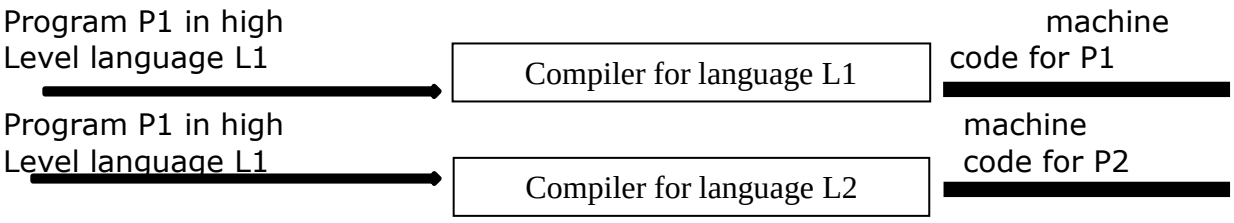
Source program Object program

As shown in the above figure the input to the compiler is a source program written in a high-level language and its output is an object program, which consists of set of equivalent machine language instructions.

The object program will be executed by the computer's processor.
The compiler can translate only those source programs, which have been written in the language for which the compiler is meant.

For e.g.: a FORTRAN compiler is only capable of translating source program, written in FORTRAN .Therefore each computer requires a separate compiler for each high-level language that it supports.

This is shown in bellow figure:



Compilers are large programs which reside permanently on secondary storage. When the translation of a program is to be done, they are copied into the main memory of the computer.

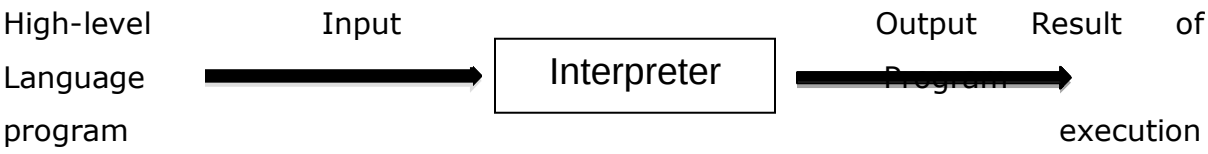
- The compiler, being a program, is executed in the CPU. While translating a given program, the compiler analyzes each statement in the source program and generates the sequence of machine instructions which, when executed, will precisely carry out the computation specified in the statement.
- A compiler cannot detect logical errors. It can only detect grammatical errors in the program.

Interpreter:

Definition : An ***interpreter*** is another type of translator, which is use for translating programs written in high-level languages.

- It takes one statement of high-level language program, translates it into machine language instructions, and then immediate executes the resulting machine language instructions. That it, in case of an interpreter, the translation and execution processes alternate for each statement encountered in the high-level language program.

- This differs from a compiler, which just translates the entire source program, but unlike a compiler, its output is the result of program execution, instead of an object program.



- After compilation of the source program, the resulting object program is permanently saved for future use, and is used every time the program is to be executed. Hence, repeated compilation is not necessary for repeated execution of a program. However, in ease of an interpreter, since no object program is saved for future use, repeated interpretation of a program is necessary for its repeated execution.

Since an interpreter translates and executes a high-level language program statement-by-statement must be reinterpreted every time it is encountered during program execution. For example, during the execution of a program, each instruction in a loop will have to be reinterpreted, every time the loop is executed.

Advantages of Interpreters

- Interpreters are easy to write, because they are less complex programs than compilers.
- They also require less memory space for execution than compilers.
- Syntax error in a program statement is detected and brought to the attention of the programmer as soon as the program statement is interpreted. This allows the programmer to make corrections during interactive program development. Therefore interpreters make it easier and faster to correct programs.

Disadvantages of Interpreters

- They are slower than the compilers when running of finished programs. This is because each statement is translated every time it is executed from source program.

Difference between Compiler and Interpreter

	Compiler		Interpreter
1	In this approach higher level language program is stored , scanned and the whole program is translated into an equivalent machine level language program.	1.	In this approach it translates one statement of higher level language at a time into machine instruction which is immediately executed.
2	Compiler is the complex program as compared to interpreter.	2.	Interpreter is the smaller program and easy to write.
3	Compiler occupy large amount of memory.	3.	Interpreter doesn't require *large* memory.
4	A complier is 5 to 25 times faster than interpreter.	4.	It require more time to execute on a computer compared to complier.
5	Object program produced by compiler is saved for future reference.	5.	The object code for the statement produced by the interpreter is not save for future reference.
6	Example : C , C++	6.	Example: Pascal

INTRODUCTION TO EDITOR

Definition :

Editor is a special program, which helps in creation and modifications of the simple text files.

It is also define, as "Editor is a program used to interactively review and alter text material and other program instructions"

Editors are used to create a file. Normally a text files. The file may be a programming code or any other text based like report, index etc.
One can modify the created file afterwards also using the editor.

Editors are used to create a text file, which does not include any graphics where as in Word processor we can include text and graphics both. Compilation of programs in some programming language is provided by some of the Editors, where as such facility is not given by any Word processor. Word processors are basically used to create a various kinds of documents not for writing programs.

Characteristics of Editor:

Normally each editor has a menu bar, which contains menu(s), each menu will have item(s). Most of the editors provide the facility of working with more than one file, using windowing technology. Menu bar is always at top of the screen. The highlighted/underlined key in menu item shows the shortcut to perform the operation. Some menu items will have direct short cut keys, i.e. without invoking the menu one can execute the command. To go to any menu, normally a highlighted/underlined key of the menu with 'alt' key is used. All most all editors provide a facility of copying and pasting of text. Find and replace, printing.

Categories of editors and its examples:

Line Editor : ed (for UNIX OS) , edlin (DOS)

Full screen editor : Turbo (Pascal Editor), NE (Norton Editor),EDIT (DOS Editor),vi (Unix Editor), Notepad (Windows Editor), tc (C/C++ Editor)

Linkage editor

1. LINE EDITOR:

This is the most basic type of editor provide means of editing text files on a line by line being identifies by a line editor,

Example: ed(on UNIX), edlin(DOS)

This editor is suitable for typing in small amount such as Batch files. Typical commands include:

LINE NUMBER ED	EDIT THE LINE SPECIFIED
A	Appends the line
D	Delete line
I	Insert text
L	List Text
E	Ending Editing

2. FULL SCREEN EDITOR OR TEXT EDITOR:

This is the most widely used for editing text files. Modern text editors normally display the file's text on the screen. Text editor does not contain text and paragraph formatting commands. They are used to type simple letters. Also text editor are used to create and edit source language programs, data and text files.

EXAMPLES : Turbo editor ,NE(Norton Editor),EDIT(DOS Editor), VI(Unix Editor), Notepad(Windows Editor).

3. LINKAGE EDITOR :

The Linkage Editor is utility Program, which adapts a program that has just been assembled or compiled into a particular environment. It links cross references between separate program modules. And it links the program to various libraries containing prewritten subroutines. The output of linkage is a program ready to run in the computer.

TURBO C++ EDITOR

Borland International Inc. develops this editor with inbuilt compilation feature. This editor will compile C as well as C++ programs. In addition to this, the debugging feature is also provided. It has inline assembler feature also, with which on can write 8086/8087- assembler code directly. A rich help on C/C++ functions is provided.

The Turbo C++ offers everything you need to write, edit, compile, link, run, manage and debug C/C++ programs. The menu bar at the top of the screen is the gateway of the menus.

To go to the menu bar, there are three different ways;

1. Press F10 key OR
2. Press Alt + ch, Where ch is the first character of the menu options OR
3. Click anywhere on menu bar (If mouse facility is available).

Once you open this editor, it has following menu Options.

File Menu: The File menu provides commands for creating new files, opening existing files, saving files, changes directories, printing files, shelling to DOS & quitting Turbo C++.

Edit Menu: The edit menu provides commands to cut, copy and paste text in edit windows. You can also undo changes and reverse the changes you have just undone.

Search Menu: The Search menu provides commands to search for text, function declarations and error locations in your files.

Run Menu: The Run menu provides commands to run your program and to start and end debugging sessions.

Compile Menu: The Compile menu provides commands to compile the program in the active edit window, or to make or build your project. .

Debug Menu: The Debug menu provides commands to control all the features of the integrated debugger.

Project Menu: The Project menu contains all the project management commands to do the following

- Create or open a project
- Add or delete files from your project
- Set options for a file in the project
- View included files for a specific file in the project, etc.

Options Menu: The Options menu contains for viewing and changing various default setting in Turbo C++.

Windows Menu: The Options menu contains window management commands.

Help Menu: This Help menu provides access to the online help system.