Module 1

Introduction to Programming

INTRODUCTION TO PROGRAMMING

Ques 1) What do you mean by programming? Explain the efficient programming.

Ans: Programming
A precise list of steps which dedicated to achieve a particular job is known as program. The set of instructions which guide the computer to perform specific operations is known as computer programs. The instructions are arranged in logical sequence and assembled by the computer programmers.

Computer programming can be considered as engineering process as careful designing, reliability and low cost are the main requirements. It is considered to be an art as well as a literary effort as programmers use their intuition, personalisation, and expertise about the programming languages while designing computer programs.

Efficient Programming
Efficient programming is programming in a manner that, when the program is executed, uses a low amount of overall resources pertaining to computer hardware. A program is designed by a human being, and different human beings may use different algorithms, or sequences of codes, to perform particular tasks. Some algorithms are more hefty and resource-intensive while accomplishing the same task than another algorithm. Practicing to create a low file size and low resource algorithm results in an efficient program.

Ques 2) What is computer language? What are the main features of a good programming language?

Ans: Computer Languages
Computer languages are necessary for communication between user and computer just like a language is required for communication among human beings. The language of communication should be such as that can be understood by the computer. Different types of computer programming languages are developed for instructing a computer to perform certain tasks.

Computer programming language consists of vocabulary, unique set of keywords, special syntax for writing program instructions and set of grammatical rules for monitoring the computer's task.

As computer understands the machine language only, so, it is important to convert a program into machine language.

There are two translators which convert the program to machine language. These are as follows:
1) Compile program, and
2) Interpret program.

The conversion does not depend on the language used for programming.

Main Features of Good Programming Language

1) **Clarity, Simplicity, and Unity:** Programming languages not only provide a method of thinking the algorithm, but also express them accordingly. The language should be clear and simple. Language offers a unified set of concepts and helps programmers in the development of algorithms before actual coding starts. The syntax of a language affects how easily a program is written, tested, reviewed and modified.
2) **Orthogonality:** This refers to the characteristics of being capable to merge different features of a language in many combinations in a meaningful way.
3) **Naturalness for the Application:** A language requires a syntax that helps program structure to mirror the basic logical structure of an algorithm. A program design should be easily transferable into appropriate program statements which show the structure of the algorithm. Different algorithms such as sequential, concurrent and logic algorithms, all have variant structure and can be represented by program in that language.
4) **Support for Abstraction:** Abstraction is an ability to explain and use complicated structures or operations in such a way that unnecessary details are ignored. The degree of abstraction as well as its expression's naturalness is essential when anyone is writing the program.
5) **Ease of Program Verification:** The languages in which programs are written are always a concern. A language that makes program verification easy is always the preferred choice over languages that are difficult to verify even though they may have many more features that make programming easily superficially. Program verification is simplified by how simple the semantic and syntactic structure is.

6) **Programming Environment:** The usefulness of programming language depends on the technical structure. An efficient programming environment ensures that even a technically weak language becomes easier to work with than a stronger language. Reliable, efficient and well-documented implementation of the language is an important pre-requisite. **For example,** Smalltalk was conceptualised around programming environment comprising windows, menus, mouse input and a set of tools for operating programs in Smalltalk.

7) **Portability of Programs:** A language that is widely available and which is independent of the features of a particular machine is useful for the production of transportable programs. **For example,** languages like Ada, FORTRAN, C, and Pascal, all consist of standard definition, permit portable applications implementation. Other languages like ML permit programmer to control over portable attributes of language. It is originated from single-source implementation.

8) **Cost of Use:** Cost is a major component, while reviewing any programming language. It means many different things as:
   i) Program execution cost,
   ii) Program translation cost,
   iii) Cost of program creation, testing, and use, and
   iv) Program maintenance cost.

Ques 3) How many types of computer languages are available?

Or

What do you mean by low level language? Give some example of it.

Or

Compare machine language, assembly language and high level language.

Or

Differentiate high level and machine level language.

Or

Differentiate between High level language and low level language.

Ans: Types of Computer Languages

Programming languages can be classified as under:

1) **Low Level Language:** A low-level language is a programming language that provides little or no abstraction of programming concepts, and is very close to writing actual machine instructions.

   Low-level languages are useful because written in them can be crafted to run very fast and with a very small memory footprint. However, they are considered more difficult to utilize because they require a deeper knowledge of machine language. Languages such as C and C++ are considered "lower-level" because they provide a minimal amount of abstraction at the smallest possible cost to performance and efficiency. Low level languages are of two types:

   i) **Machine Language (1GL)/Binary Language:** Machine language is the first generation language (1GL). It is the basic language that contains the strings of numbers (0s and 1s) defined by hardware design.

   Not only various vendors, but also the same vendor can create different machine languages. **For example,** machine language for 'System z Mainframe' family of IBM is different from the 'IBM's Power System' midrange family. All programs must be translated to machine language so that computer can understand it. A program should be recompiled or reassembled to run on a different computer system.

   Machine language is also termed as binary information because it embodies the basic level of computer information storage.

   A computer only understands its native language of commands in its instruction set which directs a computer to perform functions as loading, storing, adding and subtracting.

   ii) **Assembly Language (2GL):** Assembly languages are associated with second generation language (2GL). These are written by using English words (referred to as mnemonics) which makes it understandable to human beings and are easily convertible to machine language. This conversion is carried out by a special program known as **assembler**.

   Assembly languages are computer and CPU specific. They are the division line between machine language and higher–level languages (3GL, 4GL, etc.). The Assembler translates the Mnemonics to machine language for a specific processor family and environment. Assemblers help in easier debugging of program and launch more advanced programming mechanisms like macro and structured programming.

   Assembly languages are generally used for low-level kernels and drivers implementation. These are also used to implement the applications that are performance-oriented and processing intensive like computer games, graphic manipulation and video editing applications.

2) **High-Level Languages (3GL):** Programming has been made easier with the use of higher level language. It is called so because its syntaxes are closest to familiar words of the human language than assembly or machine language code.

   This language uses operators such as plus and minus sign which are the components of mathematics to express computer operations. Therefore, reading, writing and understanding computer programs is easier with a higher level language. In spite of this, translation of instructions into machine language is still carried out in the computer to understand and execute.

Modern high-level languages are intended to sustain development of programs that reflect the real world problem. **Examples** of these languages are Smalltalk, Simula-1, C, etc.

3) **Fourth Generation Languages (4GL):** Fourth generation languages are non-procedural and are written in a simple English like language. They are more user friendly than 3GLs.

They are machine independent, i.e., they can be used on any type of computer and therefore they are portable. This language is easy to be learnt by end users too, as there are lesser statements to remember than 3GLs.

4) **Fifth Generation Languages(5GL):** Fifth-generation programming language (5GL) is developed for solving problems using constraints given to the program. It does not use an algorithm written by the programmer.

Fourth-generation languages were developed to build specific programs whereas fifth-generation languages are conceptualised to make the computer solve a problem without the help of programmer.

Thus, the programmer's only concern remains what problems need to be solved and what conditions are to be met in place of how to implement algorithm to solve them.

Fifth-generation languages like Prolog, OPS5 and Mercury are mostly used in artificial intelligence research.

Comparison among Machine, Assembly, and High-Level Languages/Difference between Low Level Language and High Level Language

Basis	Machine Languages (Low Level Language)	Assembly Languages	High Level Languages
Understandability	Machine language programs are complex to write, debug and understand.	Assembly language programs are easier than machine language to write, debug and understand.	High-level language programs are easier to write, debug and understand.
Usage	Programmer needs to remember all machine codes corresponding to various operations.	Programmer needs to remember mnemonics corresponding to various operations, which are simpler to remember than machine language.	Programmer needs to remember the syntax for the statements, which are very easy to understand and remember as they are like in English language.
Frequency of Errors	Program writing in machine language is highly prone to errors.	Program writing in assembly language is less error-prone.	Program writing in High-level language is easy with very less possibility of errors.
Tracking Memory Address	Tracking of memory addresses which are in use are necessary for programmer.	Tracking of memory addresses which are in use are necessary for programmer.	Tracking of the memory addresses is not much necessary for programmer.
Machine Details	Programmer must have complete knowledge of the machine to be used.	Programmer must have complete knowledge of the machine to be used.	Programmer does not have to know much more details of the machine.
Portability	Programs are not portable i.e. the same program cannot run on two different machines.	Programs are not portable i.e. the same program cannot run on two different machines.	Programs are portable. Same program can run on different machines.
Maintenance of Code	Any insertion or deletion in the program affects addressing of the succeeding instructions.	Any insertion or deletion in the program affects addressing of the succeeding instructions.	Any insertion or deletion in the program does not affect addressing of the succeeding instructions.
Length of Code	Machine language programs are very long.	Assembly language programs are also very long but smaller than machine language programs.	High-level language programs are very small relative to assembly language programs.
Language Translators	Not needed	Assembler is required.	Compiler or interpreter is required.
Code Execution	Program execution is very fast.	Program execution is slower than machine language programs.	Program execution is slower than assembly language programs.

Ques 4) Describe compiler, interpreter, and assembler? Write the names of compiler that are used in c programming.

Or

Discuss the various functionalities of compiler.

Or

Differentiate between Compiler and interpreter.

Or

What is difference between Compiler and assembler?

Ans: Assembler

An assembler is a program which translates computer instructions (combination of mnemonics) written in assembly language into machine instructions (combination of binary digits), so that computer can perform its basic operations.

Or

An assembler is a program which takes an assembly language program (source program) as an input and generates machine language program (object program) as an output along with information for the loader. Macros, assembler directives and instructions are included in assembler programs.

Compiler

A compiler is a system program that converts any program written in high-level language (source program) into program written in machine language (machine program). **Examples** of source languages are C, Pascal, COBOL, etc.

It takes much memory space as well as long execution time to find the errors and limitations present in the program. The **names of compiler that are used in C programming** are like gcc, TinyCC, lcc, etc.

Interpreter

An interpreter is a program that executes the instruction written in high-level language (like C, Pascal, COBOL, etc.).

An interpreter is a program that:

1) Executes the source code directly,
2) Translates source code to intermediate code and immediately executes it, and
3) Explicitly implements stored precompiled code prepared by a compiler which is an element of the interpreter system.

Difference between Compiler and Interpreter

Table: Difference between Compiler and Interpreter

Compiler	Interpreter
It only translates the program.	It translates and executes the program.
Compiler takes entire program as input.	Interpreter takes single instruction as input.
Intermediate object code is generated.	No intermediate object code is generated.
It is used in high level language	It is used in low level language.
Require more memory for execution.	Require less memory for execution.
It is more efficient.	It is less efficient than compiler.
It reads all the instructions at one time and then executes it.	It reads and executes only a single instruction at one time (step by step processing).
It is fast to read the instructions but slow in giving the output.	It is slow to read the instructions but fast in giving the output.
Complier compiles all the errors and gives them at the end.	Unless and until the first error is corrected, the program will not move further.
Example: C compiler, PASCAL compiler	**Example:** Some interpreter languages are LISP , PERL etc.

Difference between Compiler and Assembler

Compiler	Assembler
Compiler is a computer program that reads a program written in one language and translates it in to another language.	An assembler can be considered a special type of compiler which translates only Assembly language to machine code.
Compilers usually produce the machine executable code directly from a high level language.	Assemblers produce an object code which might have to be linked using linker programs in order to run on a machine.
Compiler is used in high-level language.	Assembler is used in low-level Assembly language.
Compiler is more popular than Assembler.	Assembler is less popular than compiler.
Slight difficult to execute.	It gives most efficient executable.

FLOWCHART AND ALGORITHM

Ques 5) What is problem solving? What are the different stages of problem solving? What are the different approaches of problem solving?

Or

What do you mean by an algorithm? Explain the properties of algorithm. What are the advantages and disadvantages of algorithms?

Or

What is flowchart? Discuss the different symbols used in flow chart. What are the main advantages and disadvantages of flowcharts?

Or

Explain the difference between flowchart and algorithm with example.

Ans: Problem Solving

While accomplishing a goal if the progress or the movement is hindered by something then that hindrance is termed as the 'problem'. In routine life, people face various problems but have developed the solutions as well for such problems. Situation determines the nature of the problem.

Getting solution for any problem is both an art and science. The solution to any problem depends upon the attributes of the problem and the human's knowledge level (without learning, human reasoning and experience). Each person has their own perspective in getting the solution for any problem.

Problem solving is the process of working through details of a problem to reach a solution. Problem solving may include mathematical or systematic operations and can be a gauge of an individual's critical thinking skills.

Approaches of Problem Solving

To design and analyse the program, programmers have various programming approaches/tools. Following are the some popular tools:

1) **Algorithm:** The word Algorithm is taken from a Latin word "*algorismus*". The meaning of this word is "calculation method". For data processing and calculation a sequence of finite instructions are used, it is known as 'algorithm'. It contains the well-defined instruction to complete a task.

   An algorithm has an initial state, some precise series of successive states and in the end finish with the end-state.When programmer give a problem to the computer then he/she also select that how to solve the problem. In this situation algorithms are used. It is a method which is used to solve a problem.

   Characteristics / Properties of Algorithm

   Salient characteristics of an algorithm are as follows:

   i) **Finiteness:** After finite number of steps it terminates.
   ii) **Definiteness:** Specified rigorously and unambiguously.
   iii) **Input:** It clearly specified the valid input.
   iv) **Output:** For a valid input it can be proved to give the accurate output.
   v) **Effectiveness:** Simple and basic steps are used.

   For example, the following algorithm finds the multiplication of two numbers. Take the two numbers A and B then the result is stored in the C. Now the algorithm to multiply the two numbers is given as follows:

   Step 1: START
   Step 2: PRINT "ENTER TWO NUMBERS"
   Step 3: INPUT A, B
   Step 4: C = A * B
   Step 5: PRINT C
   Step 6: STOP

   Advantages of Algorithms

   Following are the advantages of Algorithm:

   i) It cases the process of actual development of program code.
   ii) It allows the programmers to use the most efficient solution as per time and space complexity.
   iii) It breaks down the solution of a problem into a series of simplified sequential steps.
   iv) It is a simplified way of representing program instructions enables other programmers to easily understand and modify it.

   Disadvantages of Algorithms

   The disadvantages of Algorithm are as follows:

   i) For large algorithms, it becomes difficult to understand the flow of program control.
   ii) It lacks the visual representation of programming logic as is prevalent in flowcharts.
   iii) There are no standard conventions to be followed while developing algorithms.
   iv) It may take considerable amount of time to write the algorithm for a given problem.

2) **Pseudo Code:** The high-level description of the computer programming algorithm in a compact and informal form is known as pseudo-code.

   It is not proposed for the machine reading rather it proposed for the human reading. It also uses the programming language's structural conventions.

   Descriptions those are not necessary (i.e., variable Declarations, system-specific code, and subroutine.) for the understanding of the human are removed from the pseudo-code. **For example,** consider the following pseudo code example. It finds the maximum elements for a given array of size n.

Pseudo Code: arrayMax (a[1..n], n)

Step 1: max = a[1]
Step 2: For i = 2 to n do
Step 3: If a[i] > max, then
Step 4: max = a[i]
Step 5: Endif
Step 6: Endfor
Step 7: Return max

3) **Flowcharts:** The visual representation of algorithm and pseudo is known as flowchart.

Or

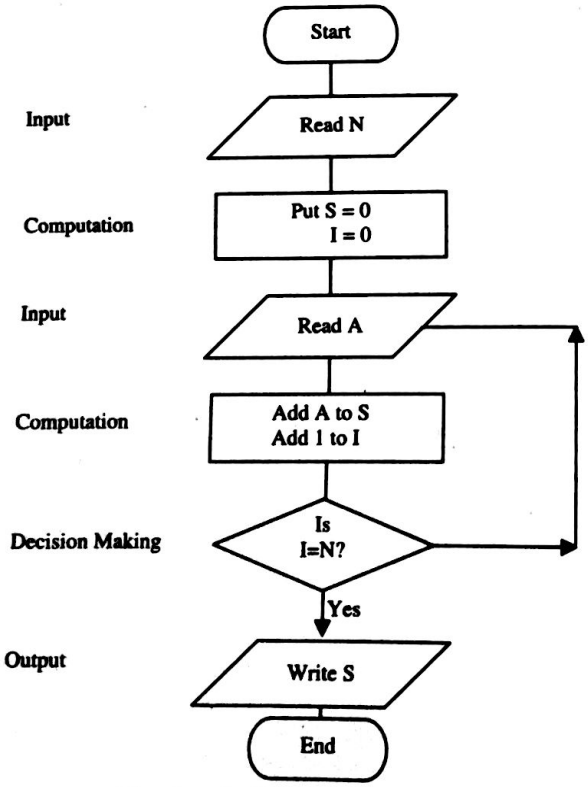
The graphical representation of a program flow and algorithm is known as flowchart.

It helps to solve a problem efficiently. It also helps to check the program flow after the coding part. For the mainlining and testing of software it acts as a good tool.

In addition, a flowchart is one that is created by the programmers that represent the sequence of steps involved in solving a problem graphically.

Flowchart is blueprint of the software project which displays the design, general plan, and necessary facts of a planned system.

In a programming sense a flowchart is a logical diagram. **For example,** the following flowchart finds the sum of N given numbers.



Flowchart for Finding Sum of N Numbers

Common Flowchart Symbols

Symbols used in flowcharts are given in the **table below**

Table: Flowchart Symbols

Flowchart Symbol	Name (Alternates)	Description
	Process	An operation or action step.
	Terminator	A start or stop point in a process.
	Decision	A question or branch in the process.
	Delay	A waiting period.
	Predefined Process	A formally defined sub-process.
	Alternate Process	An alternate to the normal process step.
	Data (I/O)	Indicates data inputs and outputs to and from a process.
	Flow line	Connects symbols and shows the flow of the algorithm logic.
	Connector	Connects different flow lines

Advantages of Flowcharts

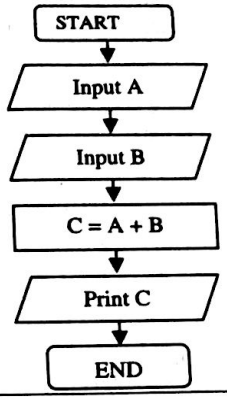
i) **Communication:** In every concern it is a good tool if communicating the logic of a system.
ii) **Effective Analysis:** Analysis is more effective if analysed using the flowchart.
iii) **Proper Documentation:** It is used as good program documentation and used for different purposes.
iv) **Proper Debugging:** In the debugging process it helps a lot.
v) **Efficient Program Maintenance:** It makes program maintenance more efficient and programmer can place efforts more efficiently on a selected part.

Disadvantages of Flowcharts

i) **Complex Logic:** For a complex logic or program flowchart becomes more complicated. A flowchart does not show that why a particular step is executed so to solve this problem a lot of comment boxes are used this will make a flowchart more cumbersome.
ii) **Alterations and Modifications:** Alteration in flowchart required completely re-drawing of that flowchart.
iii) **Reproduction:** Flowchart reproduction is a big problem because symbol used in the flowchart cannot be typed.
iv) **Time-Consuming:** It is a time consuming process because it uses only various box which is not easy as typing.
v) **Subjective:** It is more subjective. So flowcharts created by two different programmers (for the same problem) may not be the same.

Difference between Flowchart and Algorithm

The following **table** illustrates the difference between algorithm and flowchart.

Algorithm	Flowchart
An algorithm is a sequence of instructions used to solve a particular problem.	Flow chart is a pictorial representation of an algorithm.
To be an algorithm, the steps must be unambiguous and after a finite number of steps, the solution of the problem is achieved.	The boxes represent operations and the arrows represent the sequence in which the operations are implemented.
No such information is required.	Knowledge about symbol is required.
It is a bit difficult to understand.	It is easy to understand.
Whenever modification is required, only certain steps are altered.	If alterations are required the flowchart may require re-drawing completely.
For example, let the two numbers be A and B and let their sum be equal to C. Then, the desired algorithm is given as follows: **Step 1)** START **Step 2)** PRINT "ENTER TWO NUMBERS" **Step 3)** INPUT A, B **Step 4)** C = A + B **Step 5)** PRINT C **Step 6)** STOP	**For example,** the following figure shows the flow chart for addition of two numbers A and B and print sum C. 

Ques 6) Write an algorithm and draw flowchart for addition of two integers.

Ans: Algorithm for Addition of Two Integers

Step 1: START

Step 2: Initialize x, y, and z

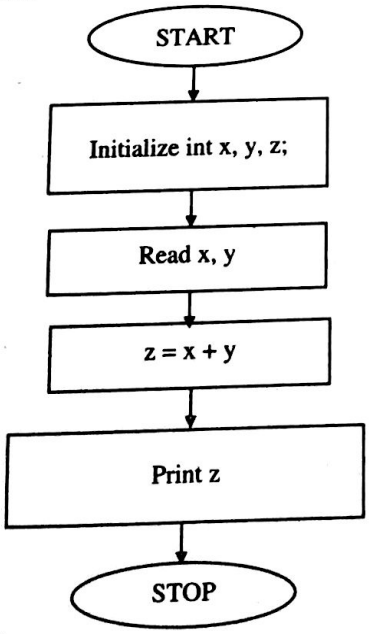
Step 3: Read x, y

Step 4: z = x + y;

Step 5: Print z;

Step 6: STOP.

Flowchart



Ques 7) Write an algorithm and draw a flowchart for finding maximum number out of any given three numbers.

Ans: Algorithm for Finding Maximum Number Out of Any Given Three Numbers

Step 1: START

Step 2: Read a, b, c

Step 3: If a > b

If a > c

Print a;

Else print c

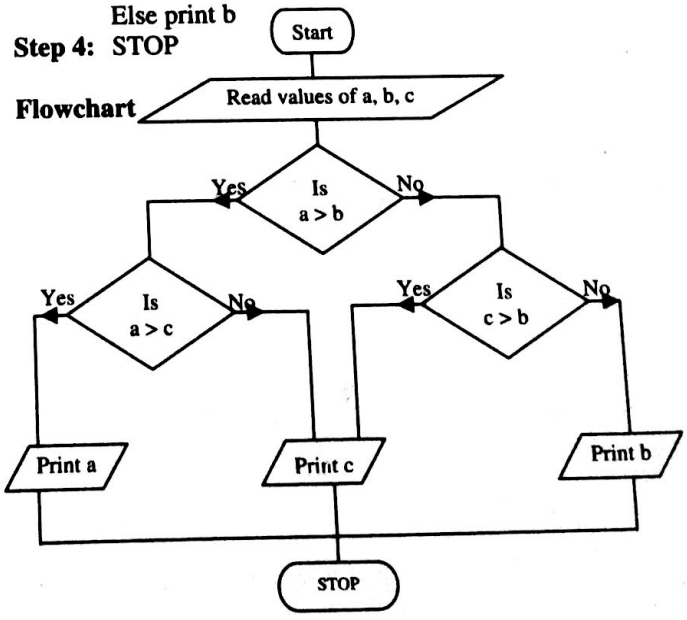
Else if c > b

Print c

Else print b

Step 4: STOP

Flowchart



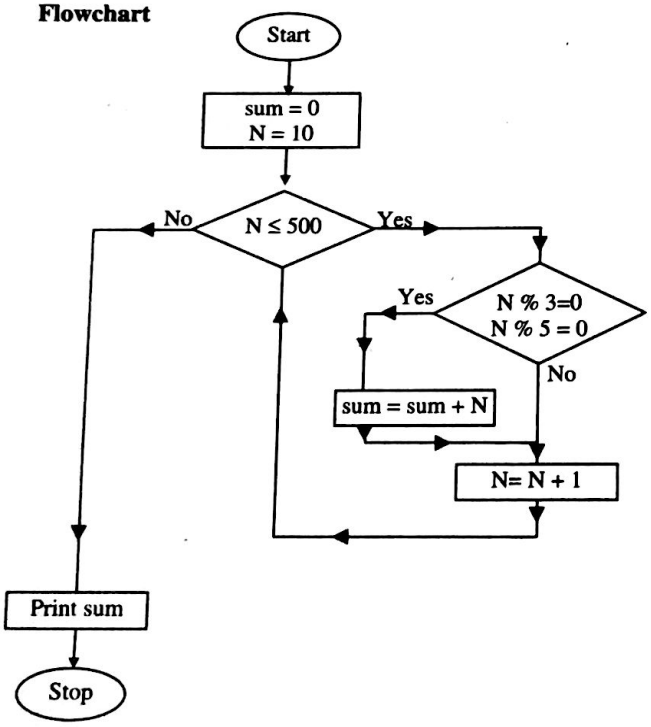
Flowchart for Finding Largest of Three Numbers

Ques 8) Write an algorithm to find the summation of all the integers between 10 and 500 which are divisible by 3 and 5 both. Also draw the flow chart for the same.

Ans: Algorithm to Summation of All Integers between 10 and 500 which are Divisible by 3 and 5 both

Step 1) Start
Step 2) let sum = 0
Step 3) Take the number and start from 10 i.e., N = 10
Step 4) Check N ≤ 500
If true
Move to step 4
If false
Move to step 5
Step 5) Check the number divisibility by 3 and 5
If true
Add number to sum i.e., sum = sum + N
If false
Take next number i.e., N = N + 1 move to step 3.
Step 6) Print sum.
Step 7) Stop

Flowchart

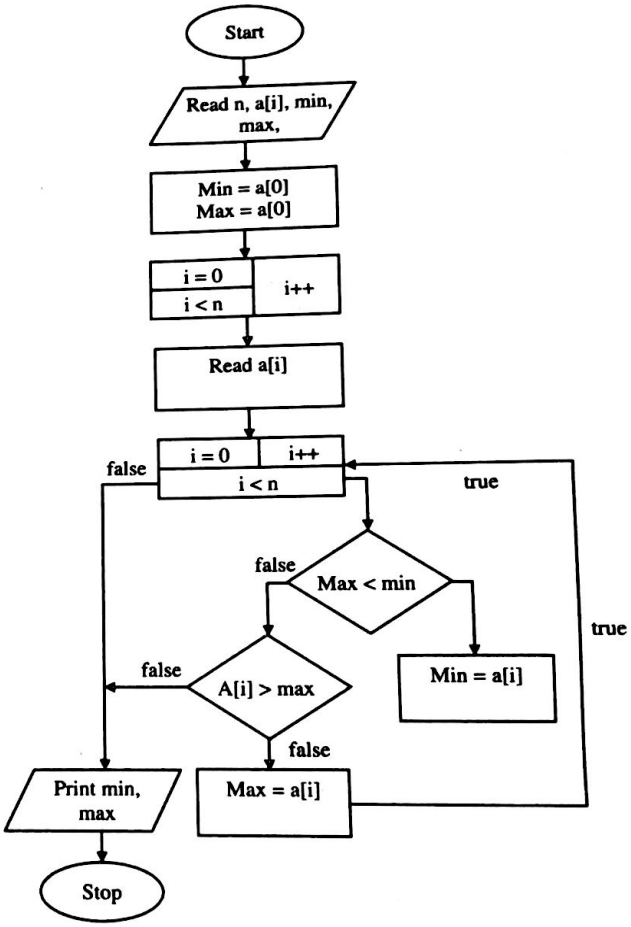


Ques 9) Give the algorithm and flowchart for finding the largest and smallest numbers in a given list of N numbers.

Ans: Algorithm for Finding the Largest and Smallest Numbers in a Given List of N Numbers

Step 1) Enter the elements of the array.
Step 2) Read the elements from the array.
Step 3) Store the first elements of the array as maximum and minimum (max = arr[0]; min = arr[0];).
Step 4) If current element of array is greater than max (i.e., array[i] > max).
Step 5) Set max = arr[i].
Step 6) Increment i by 1. Set i=i+1.
Step 7) Else If current element of array is smaller than min (i.e., arr[i]<min).
Step 8) Set min = arr[i].
Step 9) Increment i by 1. Set i=i+1.
Step 10) Repeat step 4 to 9 till i<size (where size is the size of array).

Flowchart

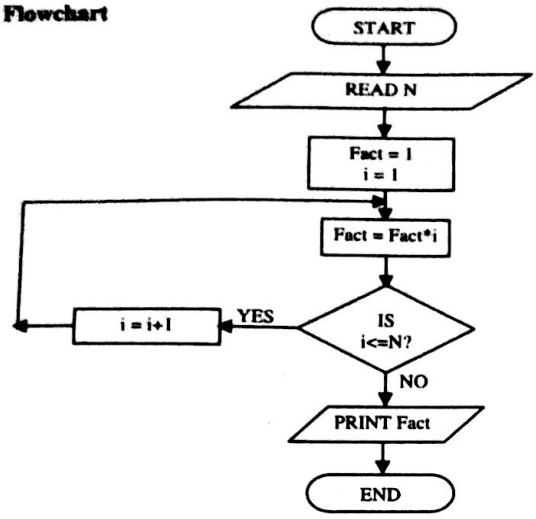


Ques 10) Write an algorithm and draw the flowchart to find the factorial of a number.

Ans: Algorithm of Factorial of Number

Step 1: START
Step 2: Read N
Step 3: Initialize Fact = 1
Step 4: For i = 1 to i < = N
Fact = Fact * i
i = i + 1
Step 5: STOP

Flowchart



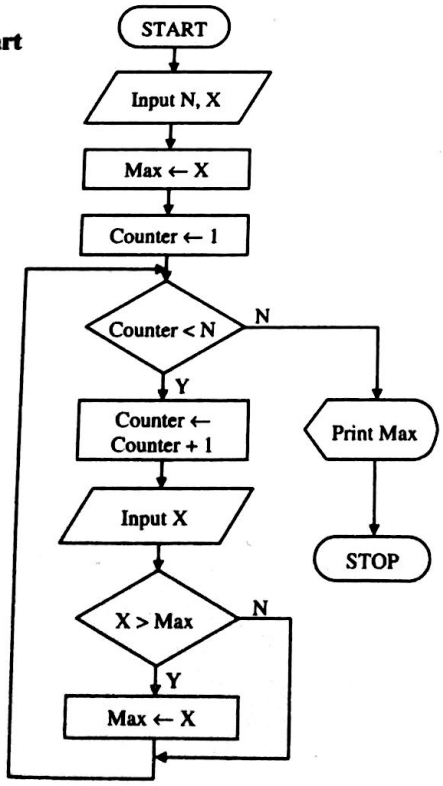
Flow chart for Computing Factorial N

Ques 11) Write an algorithm and draw the flowchart to find the largest among n numbers.

Ans: Algorithm to Find Largest Number

Step 1: START
Step 2: Read N, X
Step 3: Initialize Max = X, Counter=1
Step 4: While (Counter < N) Repeat steps 4 through 6
Step 5: Counter= Counter + 1
Step 6: If (X > Max) then
Max=X
endif go to step 4.
Step 7: Print Max

Flowchart



Ques 12) Write an algorithm to display first 20 Fibonacci numbers and their sum. Also draw a flowchart for same.

Ans: Algorithm for Fibonacci Numbers and Their Sum

Mathematically, the nth term of the Fibonacci series can be represented as:

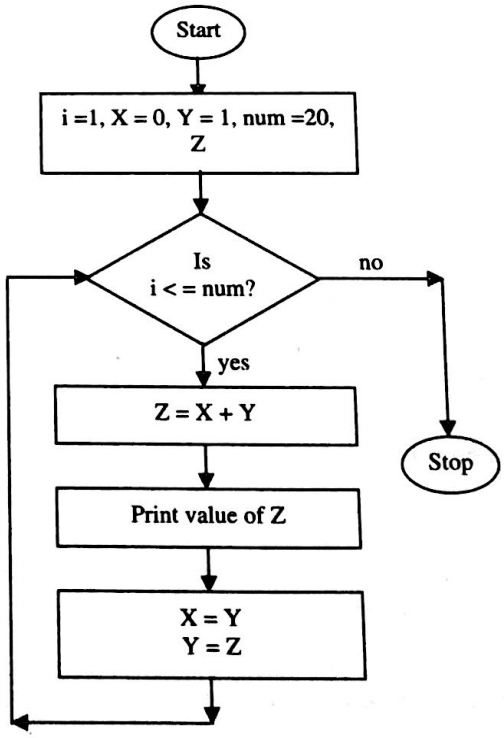
$$t_n = t_{n-1} + t_{n-2}$$

The Fibonacci numbers upto certain term can be represented as: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144..... or 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144....

The steps of algorithm are as follows:

Step 1: Start
Step 2: Declare variables i, X,Y, Z
Step 3: Initialize the variables, i=1, X=0, Y=1, and Z=0
Step 4: Enter the number of terms of Fibonacci series to be printed
Step 5: Print First two terms of series
Step 6: Check i<= number of terms, repeat step 6 to 11
Step 7: Z = X + Y
Step 8: X=Y
Step 9: Y + Z
Step 10: i =i+1
Step 11: Print Z
Step 12: Stop

Flowchart



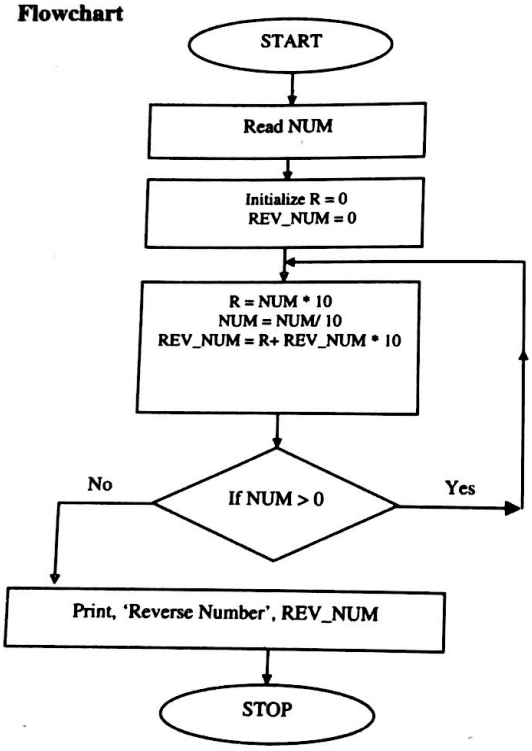
Flowchart to Generate Fibonacci Series and Their Sum

Ques 13) Write an algorithm and draw the flowchart to reverse the digit of a number. Reverse of the digit means write the numbers from right to left. For example, if 521 is a number then 125 is the reverse one.

Ans: Algorithm of Reverse the Digits of a Given Number

Step 1: START
Step 2: Read NUM
Step 3: Initialize Remainder R = 0 and REV_NUM = 0
Step 4: While (NUM > 0)
Begin
R = NUM %10;
REV_NUM = REV_NUM * 10 + R
NUM = NUM/10;
END
Step 5: Print 'Reverse Number' = REV_NUM
Step 6: STOP

Flowchart



Ques 14) Write an algorithm and draw the flowchart to computer the power of a number.

Ans: Algorithm of Computation of Power

If there are two numbers 'a' and 'b' then the power is written as 'a^b'. In this 'a' is the base and 'b' is the exponent (or index or power).

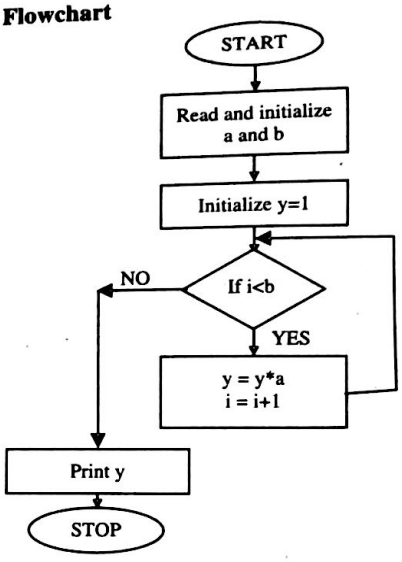
a^b means the number a is multiplied b times. **For example,** If the value of a = 3 and the value of b = 4 then it is written as:

$= 3 \times 3 \times 3 \times 3$

Steps of algorithm are as follows:

Step 1: START
Step 2: Read and initialize a and b
Step 3: Initialize y = 1
Step 4: For i = 1 to b
Step 5: If I < b Then y = y * a
Step 6: Print y
Step 7: STOP

Flowchart



Ques 15) Write an algorithm and draw the flowchart to swap two numbers using third variable.

Ans: Algorithm to Swap Two Numbers Using Third Variable

Swapping is the process in which one can interchange the value of two variables with each other. To perform swapping one needs to use third variable to store the values temporary.

Step in Swapping

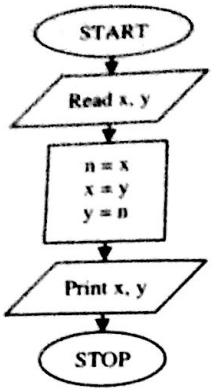
Take the two variables x and y, and one temporary variable temp.

1) Firstly save the value of variable x to the temp variable.
2) Then save the value of y to x.
3) In the last step, save the value x from the temp variable to y variable.

Steps of algorithm are as follows:

Step 1: START
Step 2: Read x, y
Step 3: n = x
Step 4: x = y
Step 5: y = n
Step 6: Print x, y
Step 7: STOP

Flowchart



Ques 16) Write an algorithm and draw the flowchart to swap two numbers without using third variable.

Ans: Algorithm Swap Two Number without Using Third Variable

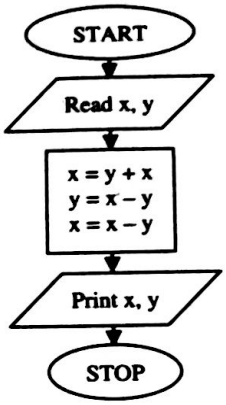
It is not necessary that to swap the two number one needs to use the temp variable. One can also swap number with other methods:

1) Firstly add both the number and save the value in a variable, say num1.
2) Now subtract the value of second variable, say num 2 from the num1 and then save in the num 2.
3) Then subtract the value of num 2 from num1 and store in num1.

Steps of algorithm are as follows:

Step 1: START
Step 2: Read x, y
Step 3: x = y + x
Step 4: y = x – y
Step 5: x = x – y
Step 6: Print x, y
Step 7: STOP

Flowchart



Ques 17) Write an algorithm and draw the flowchart to print the Pascal triangle.

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    1   6   15  20  15  6   1
```

Pascal Triangle

Ans: Algorithm to Print the Pascal Triangle

It is created by beginning through a top of 1 and is a triangular array of the binomial coefficients.

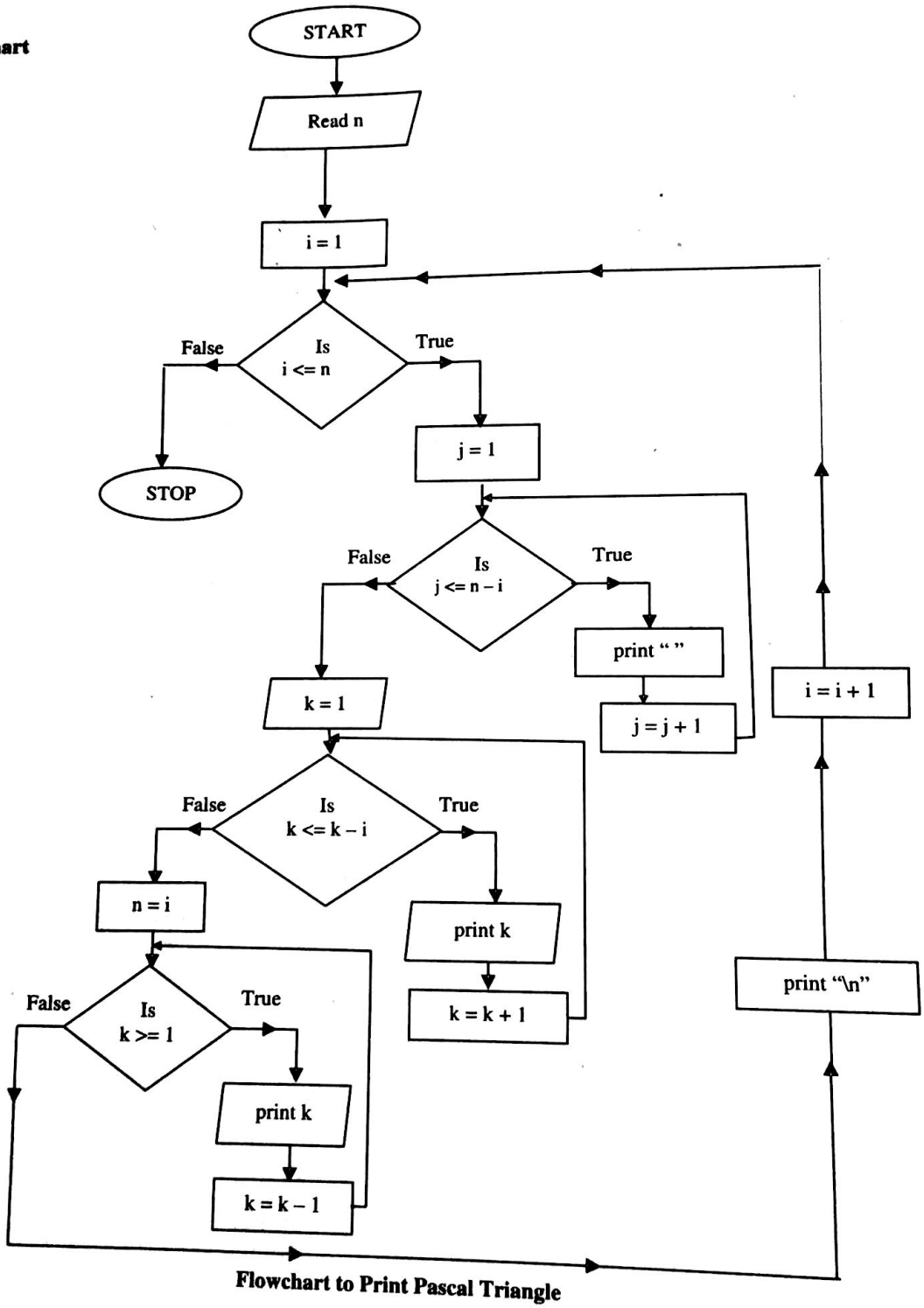
In this triangle each number below in the triangle is the addition of the two numbers crosswise (or diagonally) above it to the left and the right, by places outer the triangle counting as zero. Steps of algorithm are as follows:

Step 1: START
Step 2: Read n
Step 3: For i = 1 to n

```
    Begin
        For i = 1 to (n – i)
            Begin
                Print " "
                i = i + 1
            End
        For k = 1 to (k < i) in steps of 1
            Begin
                Print k
                k = k + 1
            End
```

```
            For k = i to 1 in steps of 1
                Begin
                    Print k
                    k= k – 1
                End
            Print from new line
                    i = i + 1
        End
STOP
```

Flowchart



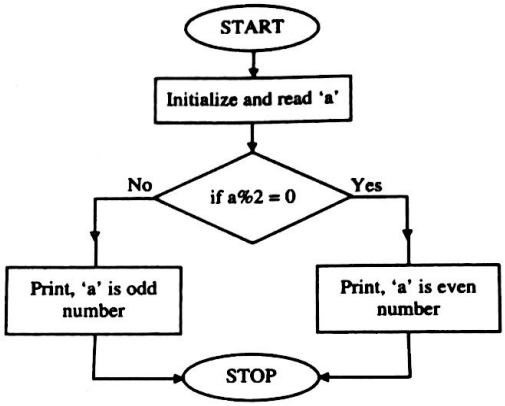
Flowchart to Print Pascal Triangle

Ques 18) Write an algorithm and draw a flowchart to find whether number is even or odd?

Ans: Algorithm to Find Even and Odd of a Number

Step 1: START
Step 2: Initialize int a
Step 3: Read a
Step 4: If (a%2 = 0)
Print 'a' is even number
Else
Print 'a' is odd number
Step 5: STOP

Flowchart



Ques 19) Write an algorithm and draw a flowchart to check leap year?

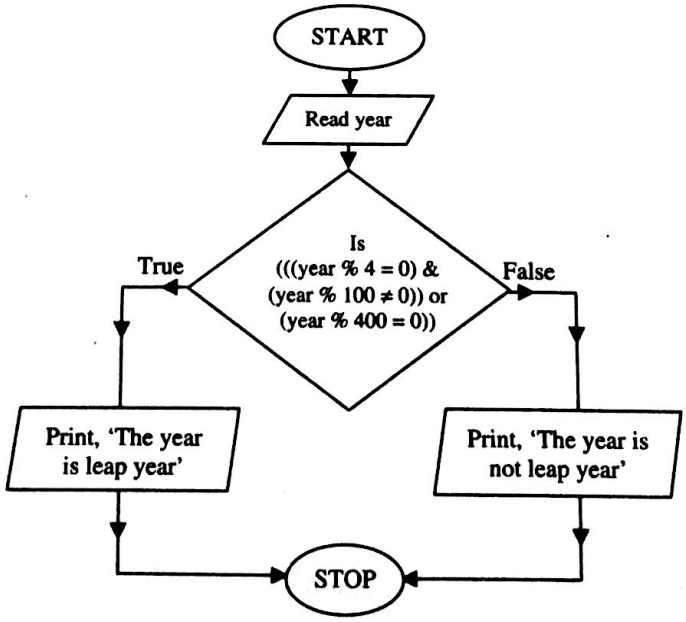
Ans: Algorithm for Checking Leap Year

If a year is divisible by 4 and not by 100, then that year is known as leap year. When a year is divisible by 4 and 100 both then it is not necessary that the year is a leap year. It is leap year if the same year is also divisible by 400. **For example,** year 1900 is divisible by 4 and 100 both, and but not divisible by 400. So, 1900 is not a leap year.

The algorithm for checking leap year is as follows:

Step 1: START
Step 2: Read year
Step 3: If (((year%4 = 0) & (year%100 ≠ 0)) or (year%400 = 0))
Print, 'The year is Leap Year'
Else
Print, 'The year is Not Leap Year'
Step 4: STOP

Flowchart



Flowchart to Check Leap Year

Ques 20) Write an algorithm and draw the flowchart to check the prime number.

Ans: Algorithm to Check Prime Number

A number which is only divisible by the 1 and itself is known as prime number. **For example,** number 3 is prime number because it is only divisible by 1 and 3 itself.

Steps of algorithm are as follows:

Step 1: START

Step 2: Read NUM

Step 3: counter = 0

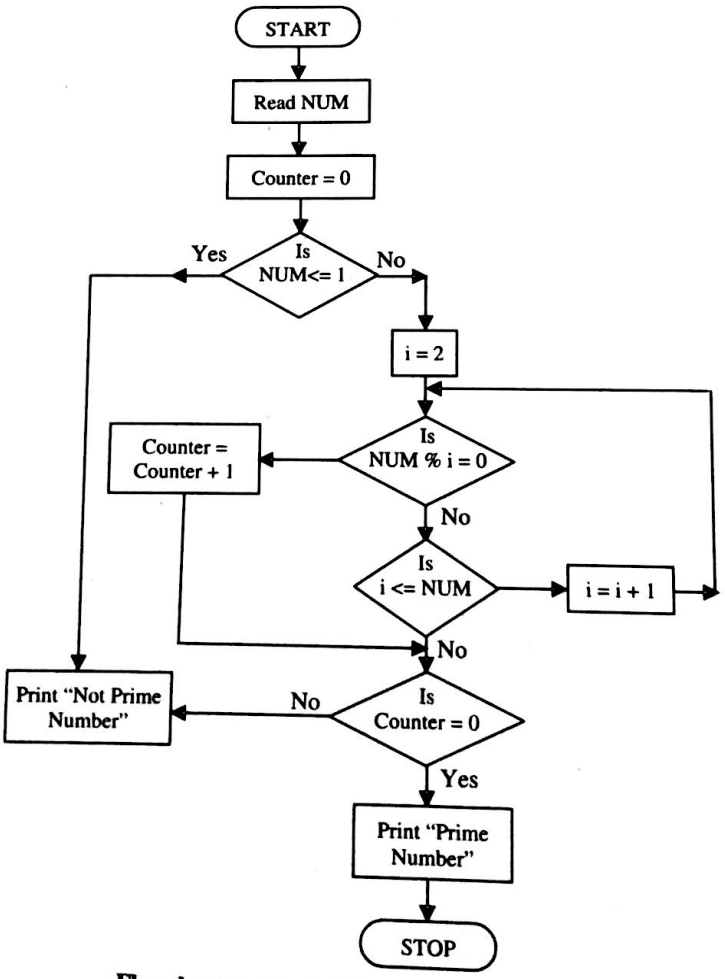
Step 4: If Num <= 1
Print, 'Not Prime Number'

Step 5: For i = 2 to NUM
Begin
If (NUM%i = 0)
counter = counter + 1 and Exit
End

Step 6: If counter = 0
Print, 'Prime Number'
Else
Print, 'Not Prime Number'

Step 7: STOP

Flowchart



Flowchart to Check Prime Number

Ques 21) Write an algorithm and draw the flowchart to find HCF or GCD of two numbers.

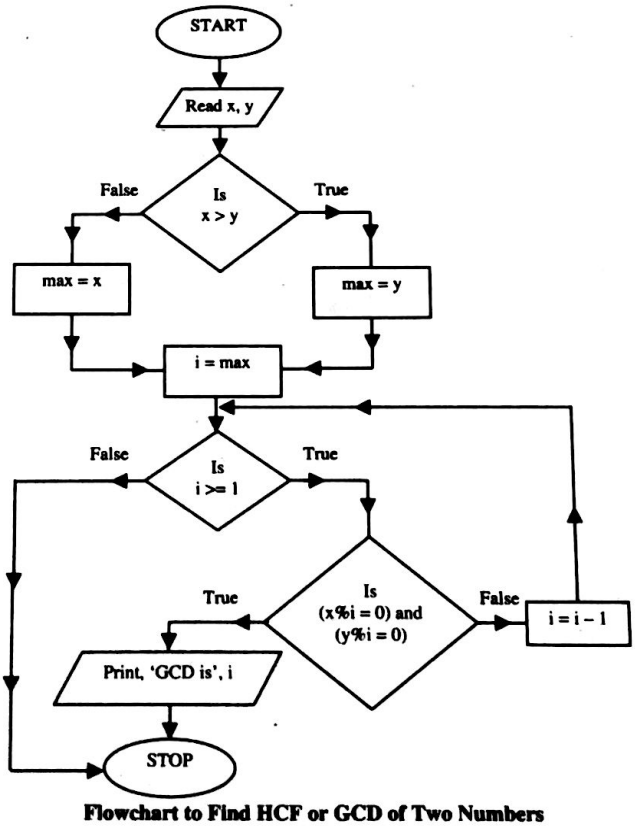
Ans: Algorithm for Find HCF or GCD of Two Numbers

Consider the two or more than two numbers. In this group, the highest number that can accurately divide every one of them is known as HCF (Highest Common Factor) or Greatest Common Divisor (GCD). **For example,** consider the two numbers 4 and 8. So, 2 and 4 are the two numbers that can divide 4 and 8. Now we consider the highest one which is 4 (HCF).

Steps of algorithm are as follows:

Step 1: START
Step 2: Read x, y
Step 3: If (x > y)
max = y
else
max = x
Step 4: For i = max to 1
Begin
if ((x%i = 0) and (y%i = 0))
Begin
Print, 'GCD or HCF of two no', i
go to step 5
End
i = i – 1
End
Step 5: STOP

Flowchart



Flowchart to Find HCF or GCD of Two Numbers

Ques 22) Write an algorithm and draw the flowchart to check that the number is palindrome or not.

Ans: Algorithm to Check Palindrome Number

If a word, phrase, number, or other sequence of symbols or elements whose meaning is same in both directions either forward or reverse direction then that word or phrase is known as palindrome.

For example, consider the string 'Malayalam' this string is same in both the direction.

Steps of algorithm are as follows:

Step 1: START

Step 2: Read n

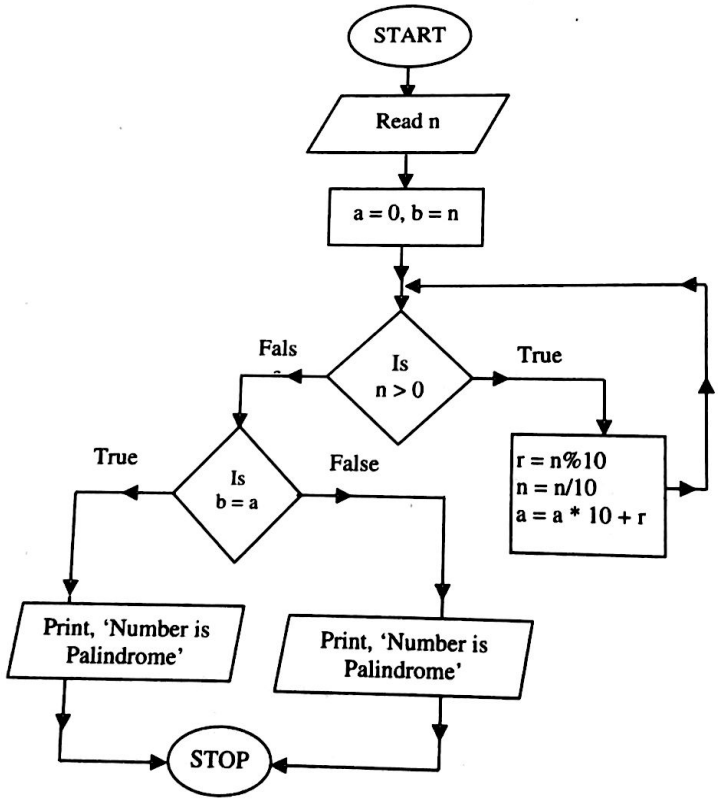
Step 3: a = 0, b = n

Step 4: While (n > 0)
Begin
r = n%10
n = n /10
a = a * 10 + r
End

Step 5: If (b = a)
Print, 'Number is Palindrome'
else
Print, 'Number is Not Palindrome'

Step 6: STOP

Flowchart



Flowchart to Check Palindrome

BASIC ELEMENTS OF C

Ques 23) Give the overview of C language. Explain features of C language.

Ans: C Language

C is a popular general purpose programming language. C language has been designed and developed by **Dennis Ritchie** at Bell Laboratories in 1972.

C is a **structured language**. It is similar in many ways to other structural languages such as Pascal, FORTRAN, etc.

A **structured language** allows variety of programs in small modules. It is easy for debugging, testing, and maintenance if a language is a structured one. Structured languages are easier and most of the developers prefer these languages than the non-structured ones like BASIC and COBOL.

Features of C

1) **Simple:** C is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.
2) **Machine Independent or Portable:** Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.
3) **Mid-Level Programming Language:** C is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language. That is why it is known as mid-level language.
4) **Structured Programming Language:** C is a structured programming language in the sense that one can break the program into parts using functions. So, it is easy to understand and modify.
5) **Rich Library:** C provides a lot of inbuilt functions that makes the development fast.
6) **Memory Management:** It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free() function.
7) **Speed:** The compilation and execution time of C language is fast.
8) **Pointer:** C provides the feature of pointers. One can directly interact with the memory by using the pointers. User can use pointers for memory, structures, functions, array etc.
9) **Recursion:** In c, one can call the function within the function. It provides code reusability for every function.
10) **Extensible:** C language is extensible because it can easily adopt new features.

Ques 24) Explain the basic structure of C program with example.

Or

Explain how to write a C program with example.

Structure of C Programming

The structure of C program is as follows:

1) **Include Header File Section:** C program depends upon some header files for function definition that are used in program. Each header file by default is extended with .h, the file should be included using #include directive as given below:

   #include<stdio.h> or #include "stdio.h"

   In this example <stdio.h> file is included, i.e., all the definitions and prototypes of function defined in this file are available in the current program. This file is also compiled with original program.

Include Header File Section
Global Declaration Section

/* comments */
main() Function Name
{
/* comments */
Declaration Part
Executable Part
}
User-defined Functions
{
}

Figure 1.1: Structure of a 'C' Program

2) **Global Declaration:** This section declares some variables that are used in more than one function. These variables are known as global variables. These variables must be declared outside of all the functions.
3) **main() Function:** Every program written in C language must contain main() function. Empty parenthesis after main are necessary. The function main() is the starting point of every 'C' program. The execution of the program always begins with the function main(). Except the main() function, other sections may not be necessary. The program execution starts with the opening brace '{'and ends with the closing brace '}'. Between these two braces the programmer should declare the declaration and the executable part.
4) **Declaration Part:** The declaration part declares the entire variables that are used in executable part. The initialisations of variables are also done in this section. The initialisation means providing initial value to the variables.
5) **Executable Part:** This part contains the moments following the declaration of the variables. This part contains a set of statements or a single statement. These statements are enclosed between the braces.

6) **User-Defined Function:** The functions defined by the user are called user-defined functions. These functions are generally defined after the main() function. They can also be defined before main() function. This portion is not compulsory.

7) **Comments:** Comments are not necessary in the program. However, to understand the flow of programs, the programmer can include comments in the program. Comments are to be inserted by the programmer. It is useful for documentation. The clarity of the program can be followed if it is properly documented. Comments are nothing but some kind of statements which are placed between the delimiters /* and */. The compiler does not execute comments. Thus, we can say that comments are not the part of executable programs.

The following is an **example** of C program to add two numbers:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num1, num2, sum;
    clrscr();
    /*Taking two numbers from user*/
    printf("Enter the two integer values to be added");
    scanf("%d %d", &num1, &num2);
    sum=num1+num2;
    printf("Addition of %d and %d  is %d\n", num1, num2,
    sum);
    getch();
    return 0;
}
```

Explanation: Following is the step-by-step program description:

1) The lines between the /*.....*/ are comments which are written for the programmer they are not compiled.
2) C Program starts with #include<stdio.h>: this line includes the "Standard I/O Library" into the program. The standard I/O library lets to read input from the keyboard called "standard in", write output to the screen called "standard out", process text files stored on the disk, and so on.

   It is an extremely useful library. C has a large number of standard libraries like conio, including string, time and math libraries. A library is simply a package of code that someone else has written to make coding easier.
3) The line void main() declares the main function. Every C program must have a function named main somewhere in the code. At run time, program execution starts at the first line of the main function.
4) In C, the '{' and '}' symbols mark the beginning and end of a block of code.
5) The variable initialization is done in the format data type var_name.
6) 'scanf' takes the input from the end user.
7) The print of statement in C allows sending output to standard out 'generally', the screen. The portion in quotes is called the format string and describes how the data is to be formatted when printed.
8) The return 0; line causes the function to return an error code of 0 'no error' to the shell that started execution.

Ques 25) What are the basic elements of C programming? Explain them with suitable example.

Or

What is variable? What is the rule of naming variables?

Or

Explain keywords, constants, identifiers with suitable example.

Ans: Basic Elements of C Programming

1) **Character Set:** The C character set includes alphabets of both upper (A-Z) and lower case (a-z) letters. As C is case sensitive, upper and lower case letters are treated differently by the compiler. It also includes the numerals (0-9).

   The C character set is given in **table below**:

Table: Character Set in C

Nature	Symbols
Alphabets	A-Z, a-z
Numerals	0-9
Arithmetic	+ – * / % = ?
Logical	! > < | &
Parentheses	( ) [ ] { }
Punctuation	; : , .
Special	= ' " # \ ^ ~ (blank)

2) **Keywords:** There are certain words, which are reserved for doing specific tasks. These words are known as keywords. These keywords have standard, predefined meaning in C. There are following keywords available in C:

Table: Reserved Keywords in C

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

3) **Identifiers:** An identifier is a sequence of characters that represents an entity such as a function or a data object.

Syntax:

identifier nondigit
identifier digit
identifier dollar-sign

For example, Thakur_publisher

4) **Variables:** Variables are used to store a value. It can be modified by program. Variable is the symbolic name given to particular portion of memory.

The values of the variables may be changed during execution of the program but one at a time. Variables must be declared before using within the program.

Variable is the symbolic name consists of the different character such as lowercase letters, upper case letters and underscore (_).

Syntax:
<data type> <variable name>;

Here data type may be int, float, char, double etc.

For example,
int basic;
int a, b, c;

Here basic, a, b, c are the variable names which are declared as data type int.

Rules for Naming Variables
The naming rules of variables are given below:
i) Letter or underscore is used to represent the first character of a variable in specific cases.
ii) Keyword cannot be used for naming variable.
iii) White spaces are not permitted for variable names.
iv) Uppercase and lowercase letters are treated different in C language. **For example,** thakur, Thakur, THAKUR shows the different variables. Lowercase letters are preferable for the naming variables. The length of variable can be arbitrarily long.

5) **Constants:** Constant is a value that can be stored in the memory and cannot be changed during execution of the program. There are two types of constants:
i) **Numeric Constant:** Numeric constants are numeric digits which may or may not have decimal point (.).

Types of Numeric Constant
a) **Integer Constant:** Integer constants are whole numbers, which has no decimal point (.).Some valid integer constants are:
123
3705
b) **Real (Floating Point) Constant:** Floating point constants are the numbers, which hold the decimal point. Some valid floating point constants are:
0.5
5.3

ii) **Character Constant:** A character constant is a single character that is enclosed within single quotes (''). **For example**, 'a', '9', '$', etc.

6) **Data Types:** Data type is defined as a finite set of values along with well-defined set of rules for operations that can be performed on these values.

7) **Comments:** Comments are also statements which are used for understanding the program. The comments are used for documentation. Comments are given by starting with **/*** and ending with ***/**. It can be of one or many number of lines.

Ques 26) What do you understand by data-type? Explain the different types of data-types in C.
Or
Write the short notes on following:
1) Data type,
2) Integer, and
3) Double

Ans: Data-Type
Data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. Data types are used to define a variable before its use.

Different Types of Data Types
1) **Integer Type:** It is used to store the integer value. The size of int is either 2 bytes or 4 bytes.

Syntax:
int <variable name>;

For example,
int num1;

Here, **num1** is a variable of type integer.

2) **char:** Keyword char is used for declaring the variable of character type. It is used to store any single character. The size of char is 1 byte. The character data type consists of ASCII characters. Each character is given a specific value.

Syntax:
char <variable name>;

For example,
char var1= 'e';

Here, **var1** is a variable of type character which is storing a character 'e'.

3) **float:** Variables of floating types can hold real values (numbers) such as: 2.34, –6.443 etc. Keywords either float or double is used for declaring floating type variable.

The size of float (Single precision float data type) is 4 bytes and that of double (Double precision float data type) is 8 bytes. Floating point variables has a precision of 6 digits whereas the precision of double is 14 digits.

Syntax:
float <variable name>;

For example,
float num1=3.302;

Here, **num1** is floating type variable.

Ques 27) What is operator? What are the different types of operator in C language?

Ans: Operator

An operator is a symbol that tells the processor to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. The data itself is called 'operand'.

To execute some mathematical or logical manipulations processors used symbols. These symbols are known as 'operators'.

To manipulate the variables and data operations are used. **For example,** if there are two variables and one operand (say '+') then it can be written as:

variable1 + variable2

Variables used in the statement are also known as 'operand' so one writes the above statement as:

Operand1 Operator Operand2

Operators operate on operands. C programming language has 45 distinct operators. Generally, operators are the part of mathematical or logical series, which is known as 'expression'.

Types of Operators

C language is rich in built-in operators and provides the following types of operators:

1) **Arithmetic Operators:** These operators generally include arithmetic operands. These include:

+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder)

Each of these operators can work with **int**, **float** or **char**.

For example, let consider the following expression:

n % 4

This gives the remainder when n is divided by 4.

2) **Relational Operators:** Relational operators are used to find the relation between two variables. They are used to compare the values of two variables in a C program.

Relational operators are sometimes called **comparison operators**. Expressions that contain relational operators are called relational expressions. Relational operators in C include:

= =	Equal to
!=	Not equal to
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater than or equal to

For example, let consider the following expression:

x < 5

This means that x is less than 5. This expression will have a value of TRUE if the variable x is less than 5; otherwise the value of the expression will be FALSE.

3) **Logical Operators:** By using a logical operator an expression evaluates to either true (non-zero value) or false (0). These include:

&&	Logical AND	If both the operands are non-zero, then condition becomes true.
||	Logical OR	If any of the two operands is non-zero, then condition becomes true.
!	Logical NOT	Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

For example, let assume variable A holds 1 and variable B holds 0, then:

(A && B) is false.

(A || B) is true.

!(A && B) is true.

4) **Bitwise Operators:** To manipulate the data at the bit level C provides special operators known as bitwise operator. Bitwise operators shift the bits from right to left. Float and double cannot use the bitwise operators.

Following **table** describe the bitwise operator's symbol with their meaning.

Table: Bitwise Operators

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

i) **Bitwise AND:** If the value of both the operand is one then it provides value one when executed with the AND else it results in zero.

Boolean Table: Bitwise AND

Bit 1	Bit 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

For example, consider the following statements:

x=5 00000101

y=9 00001001

x & y 00000001 =1

In the first two statements 5 assign to the x, and 9 assign to the y. In the last statement 'and (&)' operation is performed which gives the result 1.

ii) **Bitwise OR:** If the value of both operands is 0 then it gives 0 else 1. Following table shows the bitwise operation.

Boolean Table: Bitwise OR

Bit 1	Bit 2	Result
0	0	0
0	1	1
1	0	1
1	1	1

For example, consider the following statements

x=5	00000101
y=9	00001001
x | y	00001101 =13

In the first two statements 5 assign to the x, and 9 assign to the y. In the last statement 'OR (|)' operation is performed which gives the result 13.

iii) **Bitwise XOR:** If the value of both operands is different then it gives 1 else 0. Following **table** shows the bitwise operation.

Boolean Table: Bitwise XOR

Bit 1	Bit 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

For example

x=5	00000101
y=9	00001001
x ^ y	00001100 =12

In the first two statements 5 assign to the x, & 9 assign to the y. In the last statement 'XOR (^)' operation is performed which gives the result 12.

iv) **Bitwise Left Shift:** To shifting the bits left this operator is used. It uses the two operands.

For example, consider the following statements:

x=13	00001101
x<<6	01000000 =128

In the above statement 13 is assign to the x and the shifted left by 6.

v) **Bitwise Right Shift:** To shifting the bits right this operator is used. It also uses the two operands.

For example

x=213	11010101
x>>4	00001101=6

In the above statement 13 is assign to the x and the shifted left by 6.

5) **Assignment Operators:** Assignment operator takes the form as:

variable = expression

The basic assignment operator is =. This is used to assign the value of an expression to a variable, as below:

c = a + b

The assignment operator takes various forms. These include:

Operator	Expression	Example
=	Assigns the RHS to the LHS	C = A + B will assign value of A + B into C.
+=	Assigns the sum	C += A is equivalent to C = C + A
-=	Assigns the difference	C -= A is equivalent to C = C - A
/=	Assigns the quotient	C /= A is equivalent to C = C / A
*=	Assigns the product	C *= A is equivalent to C = C * A
>>=	Assigns the right shift	C >>= 2 is same as C = C >> 2
<<=	Assigns the left shift	C <<= 2 is same as C = C << 2
&=	Assigns the Bitwise AND	C &= 2 is same as C = C & 2

Ques 28) What is an expression? Discuss about the different types of expressions.

Or

Can we use expressions including both integer data type and character data type? Justify your answer.

Ans: Expression

When variables, operators, and constants are arranged as per syntax (grammar) of the language then it is known as expression.

Types of Expressions

There are various types of expressions that are available on the bases of the operand and set of operator used.

Statements are not expression but are considered as elements of statements. Following **figure** shows the various types of expression:

1) **Arithmetic Expressions:** Arithmetic operators are used to create an arithmetic expression. It gives the output contains the int, float, and double type value. A 'pure integer expression' is an expression which has only integral operands and an expression which has only real operands is known as pure real expression. When an expression uses the real as well as integral operands then this is known as mixed mode expression.

   To control the associativity parentheses are used. It also helps that which operators are calculated first. Expressions which are written inside the parenthesis is calculated first. If expression has the nested parenthesis then innermost (parentheses inside the parenthesis) executed first.

2) **Relational Expressions:** For comparison of two operands relational expression is used. To decide whether a decision is taken or not relational operators is used. Two different types of data values cannot be compared in the relational operation. **For example,** an int value cannot compare with the char.

   In this zero value is equal to the false (Boolean) and non-zero value is equal to true (Boolean). **For example,**

x % 2 = 0	It test whether the operand x is an even number or not. The output of the expression is 0 if the value is even else 1.

3) **Logical Expressions:** To take decision in complex test condition logical expression is used. It also gives the true and false (zero and non-zero) value. **For example,**

(x > 2) && (x % 2 = 0)	This expression checks two things: 1) Is operand x is greater than 2. 2) x is even. This expression output is 1 only if both the situations are true.

4) **Conditional Expressions:** A conditional expression is written as:

exp1? exp2 : exp3

Where, **exp1**=condition,
exp2 and **exp3**=expressions.

The value of exp1 (condition) is responsible for the final value of this conditional expression.

If the value of condition exp1 is true then the exp2 is executed else exp3.

Expressions consisting of both Integer Data Type and Character Data Type

Yes, we can use an expression including both integer and character data type. Like in any other programming language, in C, there are number of arithmetic relational and logical operator. We can use them to write expressions that are made up of simpler basic types.

For example, consider the following code segment:

```
#include <stdio.h>
main() {
 int i = 17;
 char c = 'c'; /* ascii value is 99 */
 float sum;
 sum = i + c; /*expression including both integer and
 character data type*/
 printf("Value of sum : %f\n", sum ); }
```

Explanation: When the above code is compiled and executed, it produces the following result:
Value of sum: 116.000000

Here, first 'c' gets converted to integer, but as the final value is double, usual arithmetic conversion applies and the compiler converts i and c into 'float' and adds them yielding a 'float' result.

Ques 29) Explain special operators (comma, sizeof, ternary) with the help of suitable example.

Or

What is ternary operator? Explain with suitable program.

Ans: Special Operators

1) **Comma Operator:** The comma operator (represented by the token) is a binary operator that evaluates its first operand and discards the result, and then evaluates the second operand and returns this value (and type).

The comma operator can be used to link the related expressions together. **For example,** let consider the following statement:

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

First assigns the value 10 to **x**, then assigns 5 to **y**, and finally assigns 15 (i.e., 10 + 5) to **value**. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

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

For example,

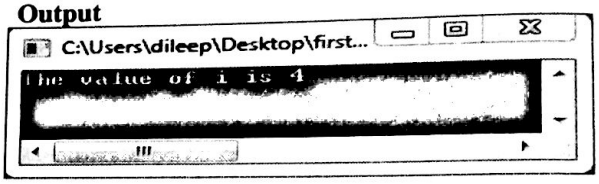
```
m = sizeof(sum);
n = sizeof(longint);
k = sizeof(235L);
```

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

Program: Illustrating sizeof Operator

```
#include<stdio.h>
#include<conio.h>
main()
{   int x;
    unsigned int i;
    i=sizeof(x);
    printf("The value of i is %u\n", i);
    getch();}
```

Output



3) **Ternary Operator:** The ternary operator allows one to execute different code depending on the value of a condition, and the result of the expression is the result of the executed code. The ternary operator is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison. It is often used as a way to assign variables based on the result of a comparison. When used correctly it can help increase the readability and reduce the amount of lines in code. They also called as?: operator.

Syntax:

expression1? expression2 : expression3

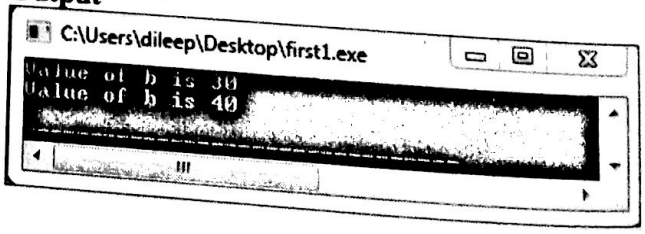
Where,

i) **expression1** is condition.
ii) **expression2** is statement followed if condition is True.
iii) **expression3** is statement followed if condition is False.

Program: Illustrating Ternary Operator

```
#include <stdio.h>
main()
{   int a , b;
    a = 10;
    printf( "Value of b is %d\n", (a == 1) ? 40: 30 );
    printf( "Value of b is %d\n", (a == 10) ? 40: 30 );
}
```

Output



Ques 30) Give an example to differentiate between post-fix and pre-fix decrement operator.

Ans: Difference between Post-Fix and Pre-Fix Decrement Operator

1) **Pre-fix Decrement Operator:** Pre-decrement operator is used to decrement the value of variable before using in the expression. In the Pre-decrement value is first decremented and then used inside the expression.

Syntax:

```
--var;
```

2) **Post-fix Decrement Operator:** Post-decrement operator is used to decrement the value of variable immediatly after executing expression completely in which post decrement is used. In the Post-decrement old value is first used in a expression and then old value will be decrement by 1.

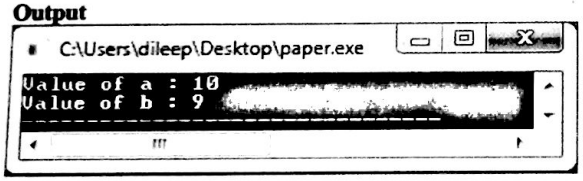
Syntax:

```
var--;
```

Program: Illustrating Difference between Post-fix and Pre-fix Decrement Operator

```
#include<stdio.h>
void main()
{       int a, b,x=10,y=10;
   a = x--;
   /* 10 is displayed then, value of a is decreased to 9 */
   b = --y;
   /* 10 is decreased then, value of b becomes 9 */
   printf("Value of a : %d",a);
   printf("\nValue of b : %d",b);}
```

Output



Ques 31) What is precedence and associativity of operators? Explain them with suitable example?

Ans: Precedence and Associativity in C

Precedence is used to determine the order in which different operators are evaluated in an expression.

Associativity is used to determine the order in which different operators with same precedence are evaluated in an expression. Associativity is applied when more than one operator of same precedence is used in an expression.

Associativity can be left-to-right or right-to-left. Left-to-right associativity evaluates the expression by starting on the left and moving to the right whereas the right-to-left associativity evaluates the expression by starting on the right and moving to the left.

Precedence is applied before associativity to determine the order in which expressions are evaluated. Associativity is applied later, if necessary.

The **tables below** show the order in which operators are evaluated:

Table: Precedence and Associativity Table

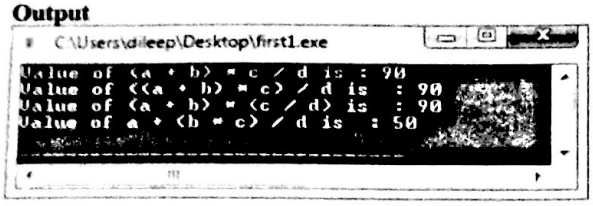
Description	Operator	Associativity
Function expression	( )	Left to Right
Array Expression	[ ]	Left to Right
Structure operator	->	Left to Right
Structure operator	.	Left to Right
Unary minus	-	Right to Left
Increment/Decrement	++ —	Right to Left
One's compliment	~	Right to Left
Negation	!	Right to Left
Address of	&	Right to Left
Value of address	*	Right to Left
Type cast	(type)	Right to Left
Size in bytes	sizeof	Right to Left
Multiplication	*	Left to Right
Division	/	Left to Right
Modulus	%	Left to Right
Addition	+	Left to Right
Subtraction	-	Left to Right
Left shift	<<	Left to Right
Right shift	>>	Left to Right
Less than	<	Left to Right
Less than or equal to	<=	Left to Right
Greater than	>	Left to Right
Greater than or equal to	>=	Left to Right
Equal to	==	Left to Right
Not equal to	!=	Left to Right
Bitwise AND	&	Left to Right
Bitwise exclusive OR	^	Left to Right
Bitwise inclusive OR	|	Left to Right
Logical AND	&&	Left to Right
Logical OR	||	Left to Right
Conditional	? :	Right to Left
Assignment	=	Right to Left
	*= /= %=	Right to Left
	+= -= &=	Right to Left
	^= |=	Right to Left
	<<= >>=	Right to Left
Comma	,	Right to Left

Program: Illustrating Precedence and Associativity of Operators

```
#include <stdio.h>
main()
{ int a = 20;
  int b = 10;
  int c = 15;
  int d = 5;
  int e;
  e = (a + b) * c / d;      // ( 30 * 15 ) / 5
  printf("Value of (a + b) * c / d is : %d\n",  e );
```

```
e = ((a + b) * c) / d;   // (30 * 15 ) / 5
printf("Value of ((a + b) * c) / d is : %d\n" , e );
e = (a + b) * (c / d);  // (30) * (15/5)
printf("Value of (a + b) * (c / d) is : %d\n", e );
e = a + (b * c) / d;    // 20 + (150/5)
printf("Value of a + (b * c) / d is : %d\n" , e );
return 0; }
```

Output



Ques 32) What is the type conversion? Explain with some example in C.

Or

Explain typecasting with the help of example.

Or

Write the difference between type conversion and type casting.

Or

Write difference between implicit and explicit type casting.

Ans: Type Conversion

Data type of a variable can be converted into other types. Their conversion generally depends upon the operator and type of operand. There are two types of conversion which are:

1) **Implicit Type Conversions:** In implicit type of conversions, the mixed data types are permitted in an expression and type conversions happen automatically, i.e., operations give data types of results themselves.

   i) **Integral Promotions:** When conversion happens from the lower rank to higher rank, then this conversion is known as type promotions and its reverse conversion is known as type demotions.

   ii) **Conversions between Integral Types:** When the integers of different data types are mixed in expressions, then there may be different types of conversions. If there is not sufficient space to hold the value then conversion from an integer to a shorter signed type will feel trouble. In such situation, the outcome is implementation defined.

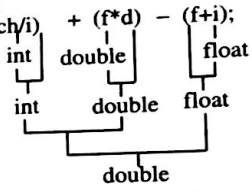
   There will be no problem if positive signed and unsigned numbers are mixed. But in case the unsigned number is positive and signed number is negative then resultant value cannot be represented in an unsigned variable because there will be loss of precision. According to C language standards, in order to convert a negative number to unsigned number, largest possible number of unsigned number plus one is added to outcome i.e., negative number.

   iii) **Conversions between Floating Types:** When the different float types of operands are mixed in an expression, then there will be a conversion which provides a common resulting type. **For example,** if we have two operands and first one is long double then the second operand is changed into long double and data type of result will also be long double. If we have two operands and first one is double then next is automatically converted into double and the type of result also be double.

   iv) **Conversions between Floating and Integral Types:** When the conversion from a float to integer type happens then this will removes the fractional part. The behaviour of conversion will be undefined in case the integer type does not hold thrown value.

For example,

```
char ch;
int i;
float f;
double d;
result = (ch/i) + (f*d) - (f+i);
```



Implicit Conversion

In the **above example**, the first line shows that character type variable ch is converted into an integer. Then the result obtained from the operation ch/i is converted into a double as the data type of result comes from f*d is double. The result comes from the operation f+i is float as f is float. The data type of final result will be double.

2) **Explicit Type Conversions (or Type Casting):** Type casting is a way to convert a variable from one data type to another data type. **For example,** if anyone wants to store a long value into a simple integer then he/she can type cast long to int.

The type casting is also known as **explicit type conversion.**

Syntax:

(data_type) expression

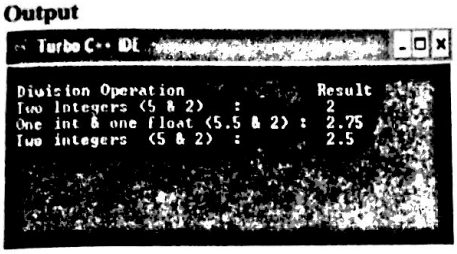
Where, **data_type** is any valid c data type, and expression may be constant, variable or expression.

Program: Illustrating Type Casting

```
# include <stdio.h>
# include <conio.h>
void main()
{
    clrscr();
    printf ("\n Division Operation          Result" );
    printf ("\n Two Integers (5 & 2)  :      %d",5/2);
```

```
    printf ("\n One int & one float (5.5 & 2) :
    %g",5.5/2);
    printf ("\n Two integers (5 & 2) :
    %g",(float)5/2);
    getch();
}
```

Output



Explanation: In the first division, operation data types are chosen as integer. Hence, the result turns out to be an integer. Actually the result should be a float value but the compiler returns an integer value. In the second division operation a float value is divided by an integer.

The result of division in this case yields a float. In third division both the values are of integer type. The result obtained is converted into float. The program uses type casting which is putting data type in a pair parenthesis before operation.

Example: /*Program to Change Data Type of Results*/

```
# include <stdio.h>
# include <conio.h>
void main()
{
    clrscr();
    printf ("\n Division Operation Result" );
    printf ("\n Two Integers (5 & 2) : %d",5/2);
    printf ("\n One int & one float (5.5 & 2) : %g",5.5/2);
    printf ("\n Two integers (5 & 2) : %g",(float)5/2);
    getch();
}
```

Explanation: In the first case, i.e., when 5 is divided by 2 then the operation data type is selected as integer and the final result will also have an integer data type. Although the result is float type but the compiler does not returns the decimal parts.

In the second case, i.e., when 5.5(float) is divided by 2(integer) then result yields the float data types.

In the third case, i.e., when 5(integer) is divided by 2(integer) the result will be converted into float. Here the type casting (float) is used before operation.

Conversion Ranks

Figure 1.2 shows the conversion process which takes into the consideration some particular sequence given by conversion ranks.

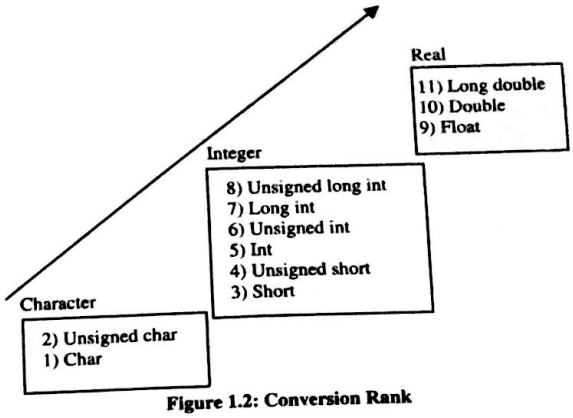


Figure 1.2: Conversion Rank

Difference between Type Conversion and Type Casting

Type Conversion is that which converts to data type into another. **For example,** converting a 'int' into 'float', converting a 'float' into 'double'. The Type Conversion is that which automatically converts the one data type into another but we cannot store a large data type into the other smaller one, **for example,** we can't store a 'float' into 'int' because a 'float' is greater than 'int'.

When a user can convert the one data type into then it is called as the type casting. Type Conversion is performed by the compiler but a casting is done by the user.

For example, converting a 'float' into 'int'. When we use the Type Conversion then it is called the promotion. When we use the type casting means then it is called as the demotion. When we use the type casting then we can loss some data.

Ques 33) Solve an expression involving three variables a, b, c:

a&&b || c&&(!b)

Given a = 3, b= 4, c= 5

Ans: a && || c && (!b)

Here a = 3, b = 4, c = 5

= 3 && || 5 && (!4)

= 0 || 5 && (0)

= 0 || 0

= 0 (false)

Ques 34) If a = 5; and b = 7; then give the value of expressions a && b and a & b in C.

Ans: Here, a = 5, b = 7

Then, a && b = 5 && 7= 1 (true)

And

a & b = 5 & 7

5 = 0101(binary form)

7 = 0111(binary form)

Bit wise AND (&) operator

```
   0101
 & 0111
 ------
   0101   = 5 (Decimal form)
```

Hence, 5 & 7 = 5

Ques 35) If a = 5; b = 6, then find the value of expression: ++a & b++.

Ans: Here, a = 5, b = 6
+ + a means it will increase before operate and b + + means, it will increase after operate.
a = 5 => + + a = 6
b = 6

```
=      0110      (Binary form)
= &    0110      (Binary form)
-------------
       0110      = 6 (Decimal form)
```

Hence, ++a & b + + = 6

Ques 1) If a = 10 and b = 15, find the value of following statement:
a < b ? a + 5: b + 6

Ans: This conditional operator expression works as follows:
Here, condition is a < b i.e., 10 < 15, which is evaluated true.

Hence, first expression of this is evaluated first.
i.e., a + 5 = 10 + 5 = 15
Hence, the output is 15.

Ques 36) What are the standard I/O functions used in C language?

Or

Write about the formatted and unformatted Input/output functions in 'C'.

Or

Write short note on getch(), getchar() and putchar(), gets(), puts() functions.

Ans: Standard I/O Functions
To read and feed the data into the program C provides the various built-in functions. C provides several functions that give different levels of input and output capability. These functions are, in most cases, implemented as routines that call lower-level input/output functions.

The input and output functions in C are built around the concept of a set of standard data streams being connected from each executing program to the basic input/output devices.

These standard data streams or files are opened by the operating system and are available to every C and assembler program to use without having to open or close the files. These standard files or streams are called:

stdin	Connected to the keyboard
stdout	Connected to the screen
stderr	Connected to the screen

The following two data streams are also available on MS DOS-based computers, but not on UNIX or other multi-user-based operating systems:

stdaux	Connected to the first serial communication port
stdprn	Connected to the first parallel printer port

A number of functions and macros exist to provide support for streams of various kinds. The <stdio.h> header file contains the various declarations necessary for the functions, together with the macros and type declarations needed for the input and output functions.

Displaying data on screen, printer or in any file is known as 'Output'. In C programming there are set of built-in output functions are available. So using these functions programmer can send the data on the computer screen.

C has the various I/O functions which provide the different level of input and output capabilities. When implemented these functions (routine) call the lower-level I/O functions.

Following are the categories of the console I/O:

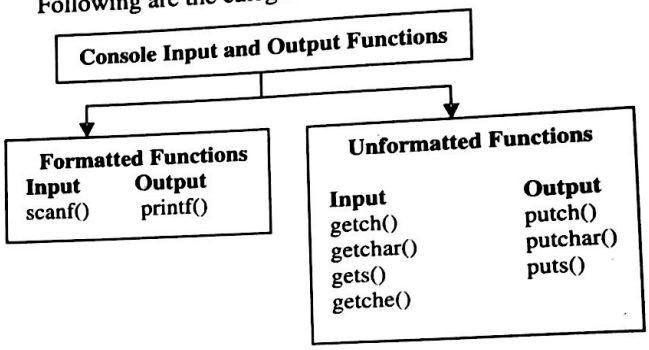


Figure 1.3: Console Input/Output Functions

1) **Formatted Input and Output Functions:** All kind of data values are read and write using formatted I/O functions. To recognise the data type they require conversion symbols. Therefore for both reading and writing of all data values these are used. After execution formatted functions returns the values. The number of variables successfully read and write are equal to the return values.

   Errors occurred during the reading and writing process can also discover with the help of these values.

   Formatted Input: scanf()
   To give the input into the computer system scanf() function is used. It is a standard library function of the C library.

   All the built-in data types can be read using scanf(). It is a general purpose console input unit. It also changes the format of the data according to the internal format.

   Data items having assigned value will be returned by the scanf() function. scanf() returns the EOF when errors are found.

   Syntax:

   scanf(control string, arg1, arg2,....., arg n)

   Where,

   control string: It determines how values are read into the variables pointed to in the argument list.

Following are the classifications of characters which are included by control string:

i) **Format Specifier:** % sign is used before the input format specifiers. It tells to the scanf() function about the data types that is to be read next.

ii) **White Space Characters:** A white space character is either:
   a) A space,
   b) A tab,
   c) A vertical tab, or
   d) A new-line.

   When used it skip one or more leading whitespace characters in the input stream.

iii) **Non-White-Space Characters in the Control String:** It reads and discards the matching characters in the input stream. Consider a situation where programmer write "%d,%d" in the scanf() function then scanf reads the integer, discard the comma and again read another integer. It terminates if the definite character is not found.

Formatted Output: printf()

printf() function is a standard library function and used to print the output data through computer onto a standard output device.

Using this function one can send the output of any combination value of numerical values, single character, and strings. This function sends the data to the standard output devices and takes it from the computer memory.

Syntax:

printf(control string, arg1, arg2,....., arg n)

Where,

i) **control string:** A string, contains formatting information, and

ii) **arg1, arg2... arg n:** These are arguments that represent the individual output data items.

2) **Unformatted Functions:** Only character data types are considered by the unformatted functions. No conversion symbol is required for data type's identification. Data conversion also not required. Any other type of data when passed treated as character data.

This function returns the same value.

Following are some unformatted function:

i) **getch() Function:** The getch() function is the standard library can be used to input a single character from the keyboard. The usage of getch is as follows:

```
c = getch();
```

When this statement is executed, the computer waits until a key is pressed. The ASCII value of the key pressed is then assigned to the character variable c.

One can use getch without a variable to affect a program pause, as in the following statements:

```
printf("Press any key to continue.");
getch();
```

The computer will output the prompt and then wait until a key is pressed. This can be any key, spacebar, return, and so on. This feature is useful to prevent the output from scrolling off the screen when the program outputs a large amount of data.

ii) **putch() Function:** Used to print a character on the screen and helps the compiler in single character output.

iii) **getchar() Function:** Used to return the character that is recently typed. Press the enter key when type the suitable character.

iv) **putchar() Function:** When used this function outputs a character variable and constant to the standard output device.

Program 3: Illustrating the getchar() and putchar() Functions

```
#include <stdio.h>
int main( ) {
  int c;
  printf( "Enter a value :");
  c = getchar( );
  printf( "\nYou entered: ");
  putchar( c );
  return 0;
}
```

Explanation: When compiled it waits for some text. After entering a text just press enter. Program will read the character and shows as follows:

```
$./a.out
Enter a value: this is test
You entered: t
```

v) **gets() Function:** From the standard input device it accepts the string and by declaring the string variable its length are controlled and limited. Use enter key when typing is complete because a string may contain multiple words.

vi) **puts() Function:** To a standard output device it gives a string constant or variable.

Program 4: Illustrating the gets() and puts() Functions

```
#include <stdio.h>
int main( ) {
  char str[100];
  printf( "Enter a value :");
  gets( str );
  printf( "\nYou entered: ");
  puts( str );
  return 0;
}
```

Explanation: When the program is run it waits input text. After providing the input when one press enter button then the program will read the whole line till the end and it shows as:

```
$./a.out
Enter a value: this is test
You entered: This is test
```

vii) **getche() Function:** The recently typed character is returned by this function. After typing the characters programmer does not need to press enter key. It immediate accepts the typed character.

Ques 37) What are the escape sequences characters?

Ans: Escape Sequence Characters
Sometimes programmer needs that when a key is press then result does not have any specific character or he/she refer to a particular character i.e., in c it has different meaning.

In these cases, programmer required an escape sequence.

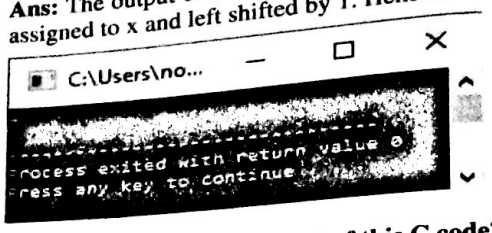
Table below shows the escape sequences that used in the C programming:

Escape Sequence	Character
\a	Bell (speaker beeps)
\b	Backspace (non-erase)
\f	Form feed/clear screen
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\?	Question mark
\'	Single quote
\"	Double quote
\xnn	Hexadecimal character code nn
\nn	Octal character code nn

Ques 38) What is the output of this C code?

```
#include <stdio.h>
int main()
{
    int x=2;
    x=x<<1;
    printf("%d\n", x);
    return 0;
}
```

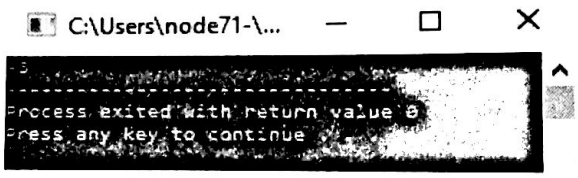
Ans: The output of the above code is as follows. Here 2 is assigned to x and left shifted by 1. Hence x = 4 now.



Ques 39) What is the output of this C code?

```
#include <stdio.h>
int main()
{
    int a=-5;
    int k = (a++, ++a);
    printf("%d",k);
    return 0;
}
```

Ans: The output of the above code is as follows. Here, – 5 is assigned to a and then first a is post incremented by 1 and then pre-incremented by 1 and assigned to k. Hence, k is – 3 now.



Ques 40) What will be the output of following code :

```
main( )
{
    float me = 1.1;
    double you = 1.1;
    if(me == you)
    printf("I LIKE C");
    else
    printf("I LIKE JAVA");
}
```

Ans: The output of the above code is as follows. Since double is not equal to float. Hence else part will be print on console.