FLOWCHART

A flowchart is a diagrammatic representation that illustrates the sequence of operations to be performed to get the solution of a problem. Flowcharts are generally drawn in the early stages of formulating computer solutions. Flowcharts facilitate communication between programmers and business people. These flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems. Once the flowchart is drawn, it becomes easy to write the program in any high level language. Often we see how flowcharts are helpful in explaining the program to others. Hence, it is correct to say that a flowchart is a must for the better documentation of a complex program.

GUIDELINES FOR DRAWING A FLOWCHART

Flowcharts are usually drawn using some standard symbols; however, some special symbols can also be developed when required. Some standard symbols, which are frequently required for flowcharting many computer programs are shown in Fig. 25.1

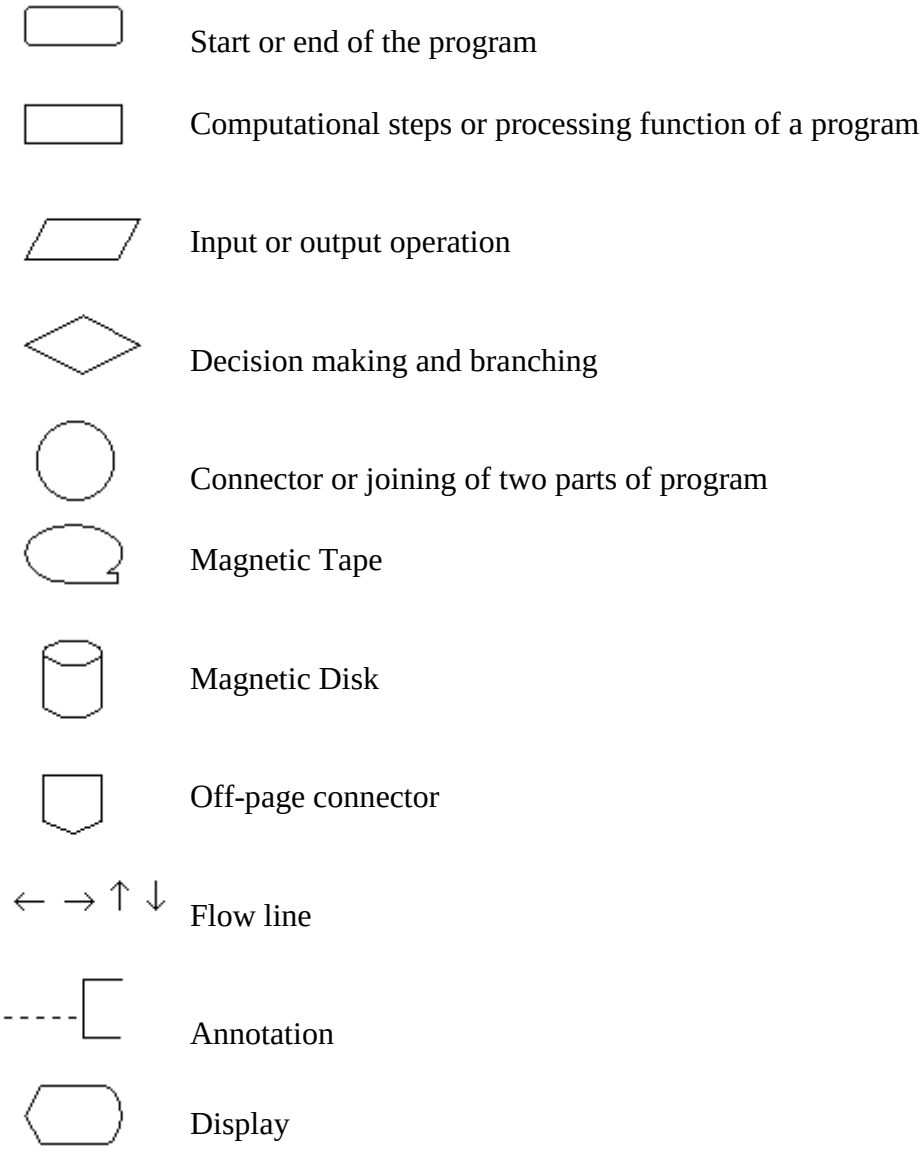
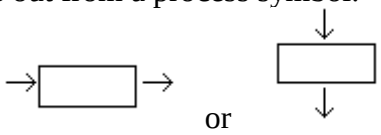


Fig. 25.1 Flowchart Symbols

The following are some guidelines in flowcharting:

a. In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
b. The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.

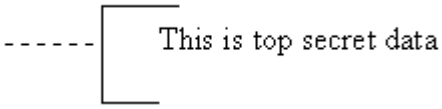
c. The usual direction of the flow of a procedure or system is from left to right or top to bottom.
d. Only one flow line should come out from a process symbol.



e. Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, should leave the decision symbol.

f. Only one flow line is used in conjunction with terminal symbol.

g. Write within standard symbols briefly. As necessary, you can use the annotation symbol to describe data or computational steps more clearly.



h. If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines. Avoid the intersection of flow lines if you want to make it more effective and better way of communication.
i. Ensure that the flowchart has a logical *start* and *finish.*
j. It is useful to test the validity of the flowchart by passing through it with a simple test data.

ADVANTAGES OF USING FLOWCHARTS

The benefits of flowcharts are as follows:

1. Communication: Flowcharts are better way of communicating the logic of a system to all concerned.
2. Effective analysis: With the help of flowchart, problem can be analysed in more effective way.
3. Proper documentation: Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. Efficient Coding: The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. Proper Debugging: The flowchart helps in debugging process.
6. Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part

LIMITATIONS OF USING FLOWCHARTS

1. Complex logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
2. Alterations and Modifications: If alterations are required the flowchart may require re-drawing completely.
3. Reproduction: As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
4. The essentials of what is done can easily be lost in the technical details of how it is done.

Example 1
Draw a flowchart to find the sum of first 50 natural numbers.
Answer: The required flowchart is given in Fig. 25.2.

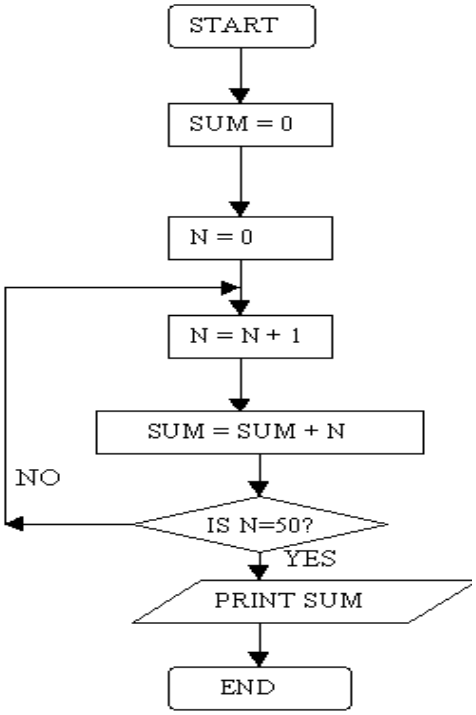


Fig. 25.2 Sum of first 50 natural numbers
Fig 2.2 Flowchart for computing the sum of first 50 natural numbers.

Example 2
Draw a flowchart to find the largest of three numbers A,B, and C.
Answer: The required flowchart is shown in Fig 25.3

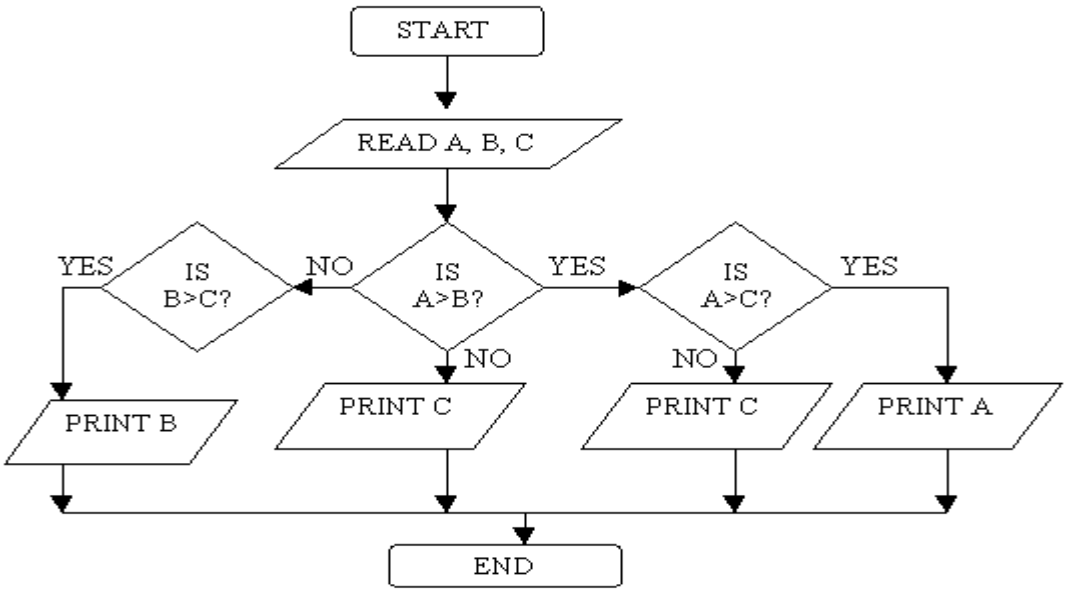


Fig 25.3 Flowchart for finding out the largest of three numbers

Example 3
Draw a flowchart for computing factorial N (N!)

Where N! = 1 ´ 2 ´ 3 ´ …… N .
The required flowchart has been shown in fig 25.4
Answer:

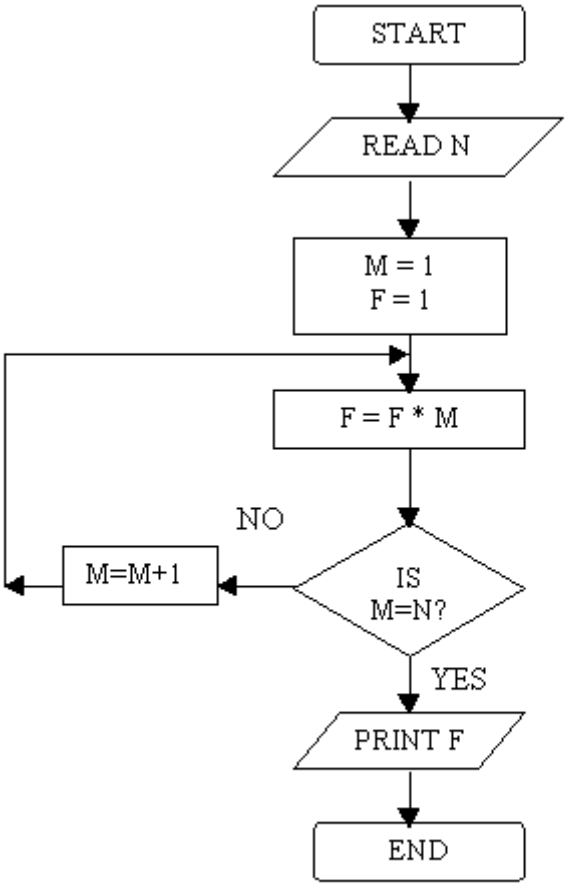
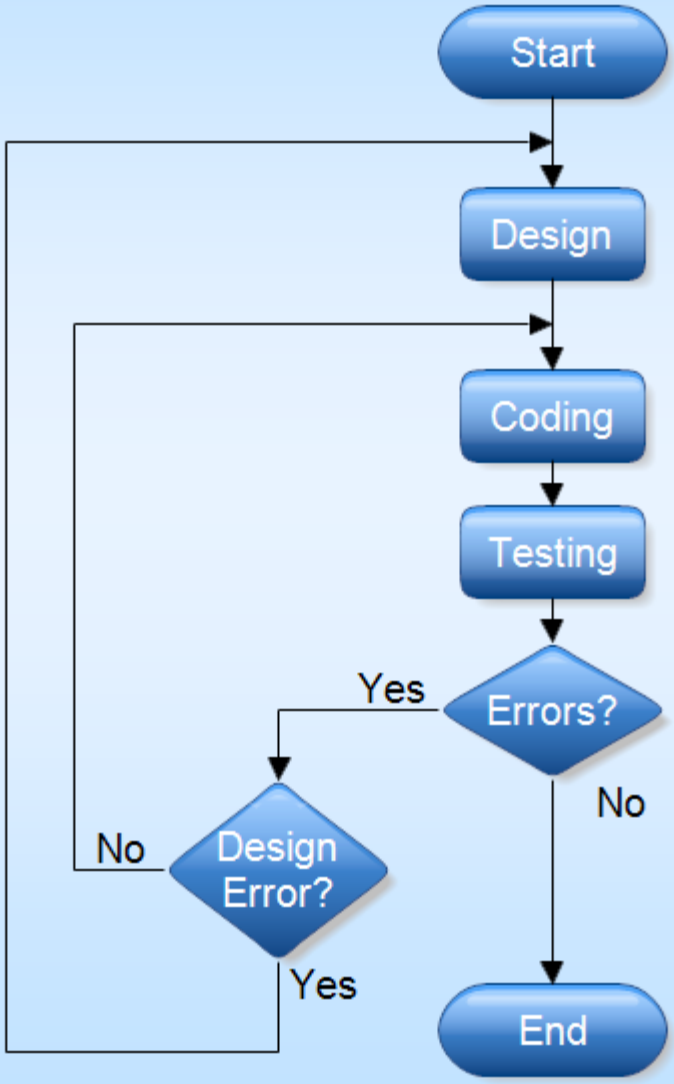


Fig 25.4 Flowchart for computing factorial N

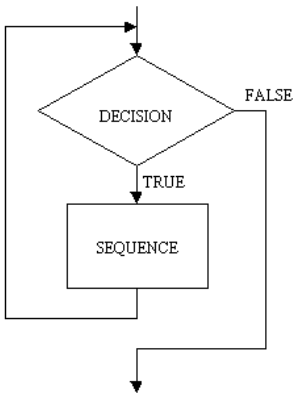
TERMINAL QUESTIONS

1. Draw a flowchart to read a number N and print all its divisors.
2. Draw a flowchart for computing the sum of the digits of any given number
3. Draw a flowchart to find the sum of given N numbers.
4. Draw a flowchart to computer the sum of squares of integers from 1 to 50
5. Draw a flowchart to arrange the given data in an ascending order.

Software Development

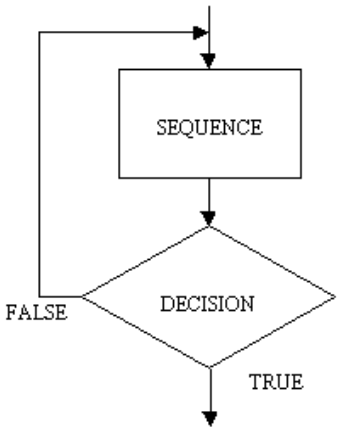


WHILE LOOP



While loop. The while loop is basically the reverse of the repeat loop, the decision comes first, followed by the process. The while loop is usually written so that it iterates *while* the condition is true, the repeat iterates *until* the condition becomes true

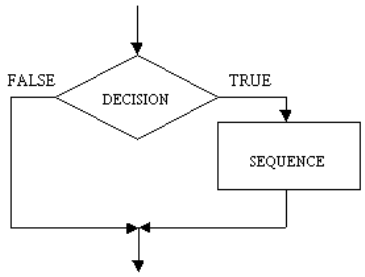
REPEAT



Repeat loop. Note that the repeat loop has the process preceding the decision. This means that a repeat loop will always execute the process part at least once
E.G

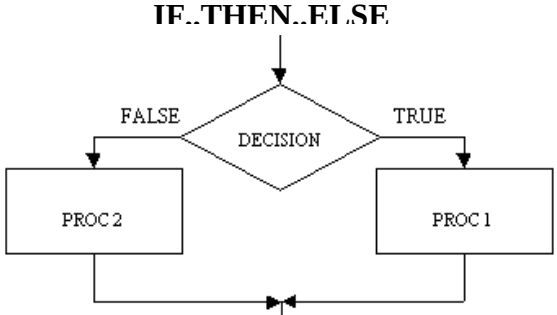
REPEAT
  LAUNCH MISSILE
UNTIL ENEMY STOPS

IF..THEN



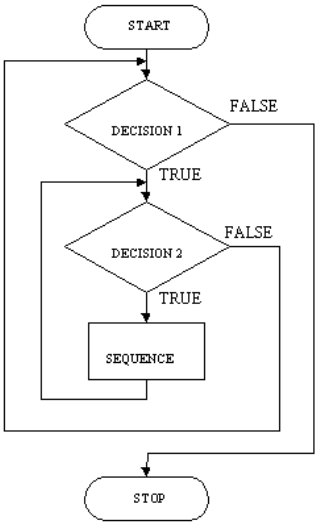
The **IF ... THEN** construct is shown here and is also known as the NULL ELSE, meaning that there is no ELSE part. I have use lines with arrow-heads (connectors) to indicate the flow of sequence. Although this is important in flow charts once you have gained some skill in using them and if you draw them carefully you will find that determining the sequence is straight forward. A typical rule is to use arrow-heads on connectors where flow direction may not be obvious.

IF..THEN..ELSE



The **IF ... THEN ... ELSE ...** construct has a process at each branch of the decision symbol. The only difference here is that each value of the decision (TRUE/FALSE) has a process associated with it.

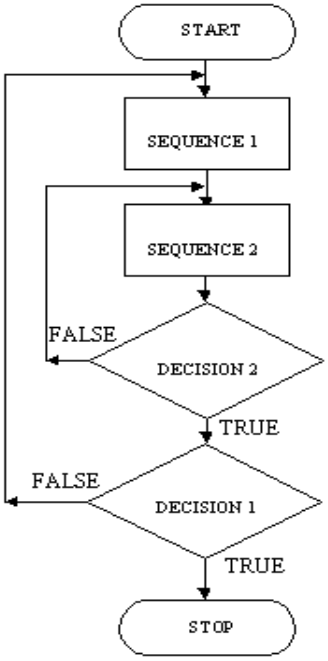
NESTED LOOP (USING WHILE)



The **nested while loop** is shown here. This example is much simplified, it doesn't show any initialisation of either of the loops, the outer loop doesn't do any processing apart from the processing the inner loop, neither loop shows any statements which will lead to the termination of the loops.

Each single step through the outer loop will lead to the complete iteration of the innner loop. Assume that the outre loop counts through 10 steps and the inner loop through 100 steps. The sequence in the inner loop will be executed 10 * 100 times. Nested loops will do a lot of work.

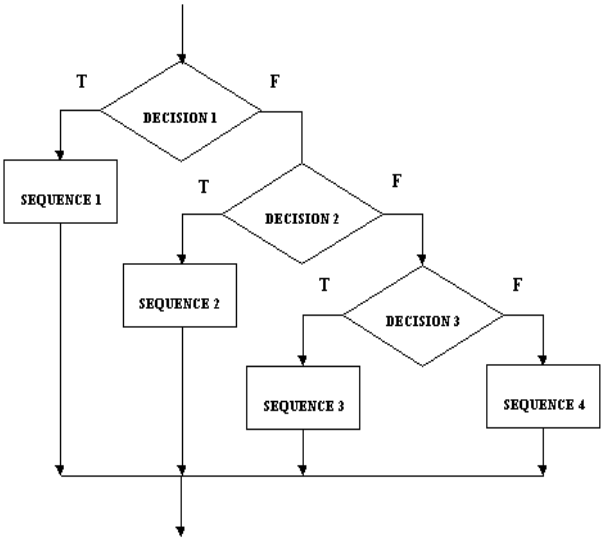
NESTED LOOP (USING REPEAT)



The **repeat loop** shown here, like the while loop example, is much simplified. It does show two processes, sequence 1 and sequence 2, one process in the outer loop and one process in the innner loop.

Like the while loop the nested repeat loop will see a great deal of work done. If the outer loop does a thousand iterations and the inner loops does a thousand iterations then sequence 2 will be executed 1000 * 1000 times.

Using multiway selection in flow charts

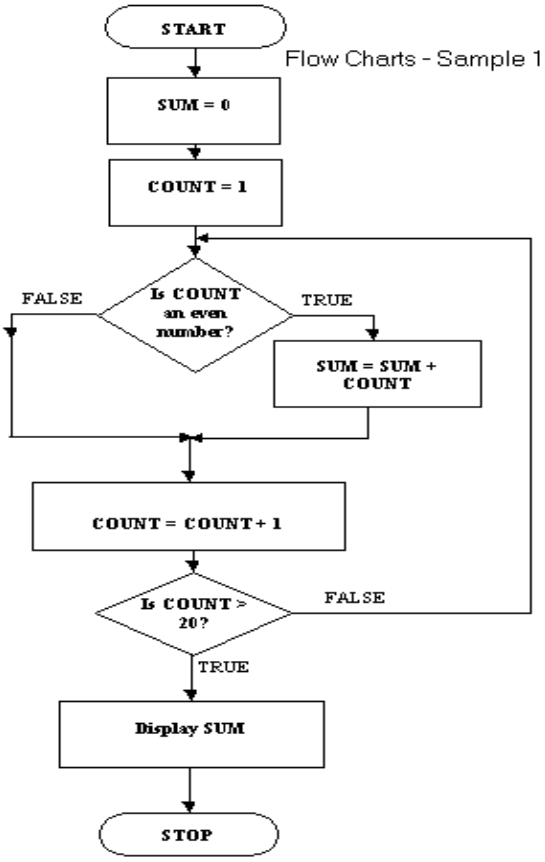


The flow chart form of **multiway selection** is shown here. You can see it how it shows quite clearly the notion of decisions nested within decisions.

If decision 1 is true then sequence 1 is executed and the multiway selection is finished. If decision 1 is false then decision 2 is tested, if this is true then sequence 2 is done and the multiway selection is finished. If decision 2 is false, you get the picture.

EXAMPLE

Sums all the even numbers between 1 and 20 inclusive and then displays the sum



The algorithm sums all the even numbers between 1 and 20 inclusive and then displays the sum. It uses a repeat loop and contains a null else within the repeat loop.

The equivalent pseudocode is:

```
sum = 0
count = 1
REPEAT
 IF count is even THEN sum = sum + count
 count = count + 1
UNTIL count > 20
DISPLAY sum
```