Algorithm and Flow Chart

Introduction

Intelligence is one of the key characteristics which differentiate a human being from other living creatures on the earth. Basic intelligence covers day to day problem solving and making strategies to handle different situations which keep arising in day to day life. One person goes Bank to withdraw money. After knowing the balance in his account, he/she decides to with draw the entire amount from his account but he/she has to leave minimum balance in his account. Here deciding about how much amount he/she may with draw from the account is one of the examples of the basic intelligence. During the process of solving any problem, one tries to find the necessary steps to be taken in a sequence. In this Unit you will develop your understanding about problem solving and approaches.

Problem Solving

Can you think of a day in your life which goes without problem solving? Answer to this question is of course, No. In our life we are bound to solve problems. In our day to day activity such as purchasing something from a general store and making payments, depositing fee in school, or withdrawing money from bank account. All these activities involve some kind of problem solving. It can be said that whatever activity a human being or machine do for achieving a specified objective comes under problem solving. To make it clearer, let us see some other examples.

Example1*:* If you are watching a news channel on your TV and you want to change it to a sports channel, you need to do something i.e. move to that channel by pressing that channel number on your remote. This is a kind of problem solving.

Example 2*:* One Monday morning, a student is ready to go to school but yet he/she has not picked up those books and copies which are required as per timetable. So here picking up books and copies as per timetable is a kind of problem solving.

Example 3*:* Some students in a class plan to go on picnic and decide to share the expenses among them. So calculating total expenses and the amount an individual have to give for picnic is also a kind of problem solving.

Now, broadly we can say that problem is a kind of barrier to achieve something and problem solving is a process to get that barrier removed by performing some sequence of activities.

Here it is necessary to mention that all the problems in the world can not be solved. There are some problems which have no solution and these problems are called *Open Problems*.

If you can solve a given problem then you can also write an algorithm for it. In next section we will learn what is an *algorithm.*

Algorithm

The term **algorithm** originally referred to any computation performed via a set of rules applied to numbers written in decimal form. The word is derived from the phonetic pronunciation of the last name of ***Abu Ja'far Mohammed ibn Musa al-Khowarizmi***, who was an Arabic mathematician who invented a set of rules for performing the four basic arithmetic operations (addition, subtraction, multiplication and division) on decimal numbers.

An algorithm is a representation of a solution to a problem. If a problem can be defined as a difference between a desired situation and the current situation in which one is, then a problem solution is a procedure, or method, for transforming the current situation to the desired one. We solve many such trivial problems every day without even thinking about it, for example making breakfast, travelling to the workplace etc. But the solution to such problems requires little intellectual effort and is relatively unimportant. However, the solution of a more interesting problem of more importance usually involves stating the problem in an understandable form and communicating the solution to others. In the case where a computer is part of the means of solving the problem, a procedure, explicitly stating the steps leading to the solution, must be transmitted to the computer. This concept of problem solution and communication makes the study of algorithms important to computer science.

Throughout history, man has thought of ever more elegant ways of reducing the amount of labour needed to do things. A computer has immense potential for saving time/energy, as most (computational) tasks that are repetitive or can be generalized can be done by a computer. For a computer to perform a desired task, a method for carrying out some sequence of events, resulting in accomplishing the task, must somehow be described to the computer. The algorithm can be described on many levels because the algorithm is just the procedure of steps to take and get the result. The language used to describe an algorithm to other people will be quite different from that which is used by the computer, however the actual algorithm will in essence be the same.

Algorithm can be defined as: ***"A sequence of activities to be processed for getting desired output from a given input."*** .

Webopedia defines an algorithm as: ***"A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point."***

There may be more than one way to solve a problem, so there may be more than one algorithm for a problem. Now, if we take definition of algorithm as: ***"A sequence of activities to be processed for getting desired output from a given input."*** Then we can say that:

1. Getting specified output is essential after algorithm is executed.

2. One will get output only if algorithm stops after finite time.

3. Activities in an algorithm to be clearly defined in other words for it to be unambiguous.

Alternatively, we can define an algorithm as a set or list of instructions for carrying out some process step by step. A recipe in a cookbook is an excellent example of an algorithm. The recipe includes the requirements for the cooking or ingredients and the method of cooking them until you end up with a nice cooked dish.

Example of a recipe to make a cake.

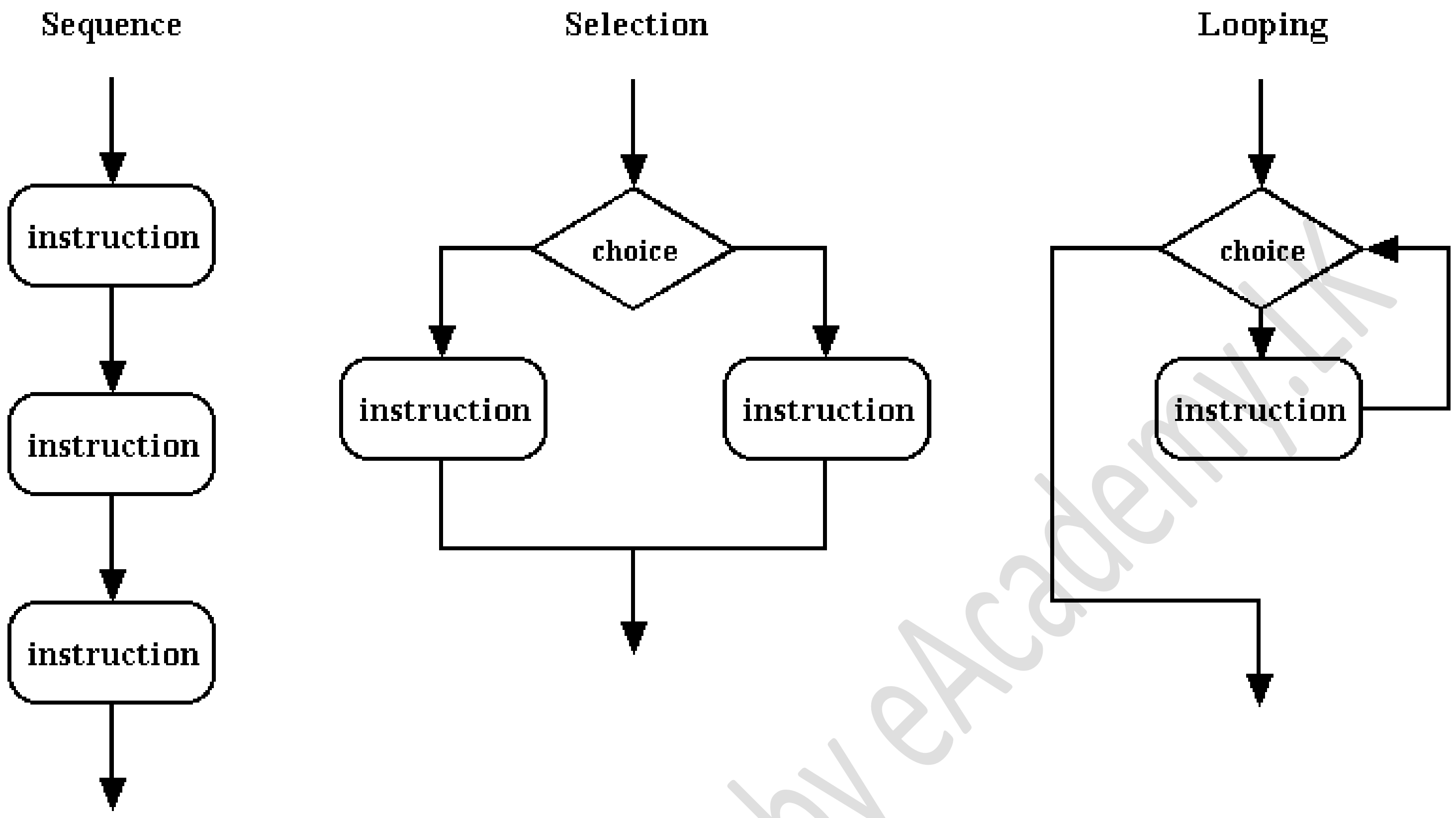
"4 extra large eggs, beaten
1&1/2 C. stock
1/2 teaspoon salt
1 scallion, minced
1 C. small shrimp or lobster flakes
1 t. soy sauce
1 Tablespoon oil

1. Mix all the ingredients, except the oil, in a deep bowl.
2. Put 1water in wide pot, then place deep bowl of batter inside.
3. Cover pot tightly and steam 15 min.
4. Heat oil very hot and pour over custard.
5. Steam 5 more min.

Type of Algorithms

The algorithm and flowchart, classification into the three types of control structures. They are:
1. Sequence
2. Branching (Selection)
3. Loop (Repetition)

These three control structures are sufficient for all purposes.



<u>Sequence</u>

The sequence is exemplified by sequence of statements place one after the other – the one above or before another gets executed first. In flowcharts, sequence of statements is usually contained in the rectangular process box.

1. Find the area of a Circle of radius r.

Inputs to the algorithm:
Radius r of the Circle.

Expected output:
Area of the Circle

Algorithm:
Step1: Read\input the Radius r of the Circle
Step2: Area = PI*r*r // calculation of area
Step3: Print Area

2. Write an algorithm to read two numbers and find their sum.

Inputs to the algorithm:
First num1.
Second num2.

Expected output:
Sum of the two numbers.

Algorithm:
```
Step1: Start
Step2: Read\input the first num1.
Step3: Read\input the second num2.
Step4: Sum num1+num2 // calculation of sum
Step5: Print Sum
Step6: End
```

3. Convert temperature Fahrenheit to Celsius

Inputs to the algorithm:
Temperature in Fahrenheit
Expected output:
Temperature in Celsius

Algorithm:
```
Step1: Start
Step 2: Read Temperature in Fahrenheit F
Step 3: C 5/9*(F32)
Step 4: Print Temperature in Celsius: C
Step5: End
```
<u>Selection</u>

The branch refers to a binary decision based on some condition. If the condition is true, one of the two branches is explored; if the condition is false, the other alternative is taken. This is usually represented by the "if-then" construct in pseudo-codes and programs. In flowcharts, this is represented by the diamond-shaped decision box. This structure is also known as the selection structure.

1. Write algorithm to find the greater number between two numbers.

```
Step1: Start
Step2: Read/input A and B
Step3: If A greater than B then C=A
Step4: if B greater than A then C=B
Step5: Print C
Step6: End
```

2. An algorithm to find the largest value of any three numbers.

```
Step1: Start
Step2: Read/input A,B and C
Step3: If (A>=B) and (A>=C) then Max=A
Step4: If (B>=A) and (B>=C) then Max=B
Step5:If (C>=A) and (C>=B) then Max=C
Step6: Print Max
Step7: End
```

Repetition

The loop allows a statement or a sequence of statements to be repeatedly executed based on some loop condition. It is represented by the 'while' and 'for' constructs in most programming languages, for unbounded loops and bounded loops respectively. (Unbounded loops refer to those whose number of iterations depends on the eventuality that the termination condition is satisfied; bounded loops refer to those whose number of iterations is known before-hand.) In the flowcharts, a back arrow hints the presence of a loop. A trip around the loop is known as iteration. You must ensure that the condition for the termination of the looping must be satisfied after some finite number of iterations, otherwise it ends up as an infinite loop, a common mistake made by inexperienced programmers. The loop is also known as the repetition structure.

1. An algorithm to calculate even numbers between 0 and 99

```
Step1. Start
Step2. I = 0
Step3. Write I in standard output
Step4. I = I+2
Step5. If (I <=98) then go to line 3
Step6. End
```

2. Design an algorithm which gets a natural value, n as its input and calculates odd numbers equal or less than n. Then write them in the standard output:

```
Step1. Start
Step2. Read n
Step3. I ← 1
Step4. Write I
Step5. I ← I + 2
Step6. If (I <= n) then go to line 4
Step7. End
```

3. Design an algorithm which generates even numbers between 1000 and 2000 and then prints them in the standard output. It should also print total sum:

```
Step1. Start
Step2. I = 1000 and S = 0
Step3. Write I
Step4. S = S + I
Step5. I = I + 2
Step6. If (I <= 2000) then go to line 3
else go to line 7
Step7. Write S
Step8. End
```

4. Design an algorithm with a natural number, n, as its input which calculates the following formula and writes the result in the standard output:

$S = \frac{1}{2} + \frac{1}{4} + \ldots + 1/n$

```
Step1. Start
Step2. Read n
Step3. I ← 2 and S ← 0
Step4. S= S + 1/I
Step5. I ← I + 2
Step6. If (I <= n) then go to line 4
else write S in standard output
Step7. End
```

Combining the use of these control structures, for example, a loop within a loop (nested loops), a branch within another branch (nested if), a branch within a loop, a loop within a branch, and so forth, is not uncommon. Complex algorithms may have more complicated logic structure and deep level of nesting, in which case it is best to demarcate parts of the algorithm as separate smaller *modules*. Beginners must train themselves to be proficient in using and combining control structures appropriately, and go through the trouble of tracing through the algorithm before they convert it into code.

Algorithms executed by a computer can combine millions of elementary steps, such as additions and subtractions, into a complicated mathematical calculation.

Also by means of algorithms, a computer can control a manufacturing process or coordinate the reservations of an airline as they are received from the ticket offices all over the country. Algorithms for such large-scale processes are, of course, very complex, but they are built up from pieces.

One of the obstacles to overcome in using a computer to solve your problems is that of translating the idea of the algorithm to computer code (program). People cannot normally understand the actual machine code that the computer needs to run a program, so programs are written in a programming language such as C or Pascal, which is then converted into machine code for the computer to run.

Properties of algorithm

Donald Ervin Knuth has given a list of five properties for a algorithm, these properties are:

1) Finiteness: An algorithm must always terminate after a finite number of steps. It means after every step one reach closer to solution of the problem and after a finite number of steps algorithm reaches to an end point.

2) Definiteness: Each step of an algorithm must be precisely defined. It is done by well thought actions to be performed at each step of the algorithm. Also the actions are defined unambiguously for each activity in the algorithm.

3) Input: Any operation you perform need some beginning value/quantities associated with different activities in the operation. So the value/quantities are given to the algorithm before it begins.

4) Output: One always expects output/result (expected value/quantities) in terms of output from an algorithm. The result may be obtained at different stages of the algorithm. If some result is from the intermediate stage of the operation then it is known as intermediate result and result obtained at the end of algorithm is known as end result. The output is expected value/quantities always have a specified relation to the inputs.

5) Effectiveness: Algorithms to be developed/written using basic operations. Actually operations should be basic, so that even they can in principle be done exactly and in a finite amount of time by a person, by using paper and pencil only.

Advantages of algorithm

- It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
- An algorithm uses a definite procedure.
- It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
- Every step in an algorithm has its own logical sequence so it is easy to debug.

In the problem-solving phase of computer programming, you will be designing algorithms. This means that you will have to be conscious of the strategies you use to solve problems in order to apply them to programming problems. These algorithms can be designed though the use of **pseudocodes** and **flowcharts**.

Flowcharts

Flowcharting is a tool developed in the computer industry, for showing the steps involved in a process. A flowchart is a diagram made up of *boxes*, *diamonds* and other shapes, *connected by arrows* - each shape represents a step in the process, and the arrows show the order in which they occur. Flowcharting combines symbols and flow lines, to show figuratively the operation of an algorithm.

In computing, there are dozens of different symbols used in flowcharting (there are even national and international flowcharting symbol standards). In business process analysis, a couple of symbols are sufficient. A box with text inside indicates a step in the process, while a diamond with text represents a decision point. See the figure for an example.

If the flowchart is too messy to draw, try starting again, but leaving out all of the decision points and concentrating on the simplest possible course. Then the session can go back and add the decision points later. It may also be useful to start by drawing a high-level flowchart for the whole organization, with each box being a complete process that has to be filled out later.

From this common understanding can come a number of things - process improvement ideas will often arise spontaneously during a flowcharting session. And after the session, the facilitator can also draw up a written procedure - a flowcharting session is a good way of documenting a process.

Process improvement starts with an understanding of the process, and flowcharting is the first step towards process understanding.

Flowcharting Symbols

There are six basic symbols commonly used in flowcharting of assembly language programs: Terminal, Process and input/output, Decision, Connector and Predefined Process. This is not a complete list of all the possible flowcharting symbols, it is the ones used most often in the structure of Assembly language programming.

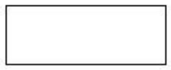
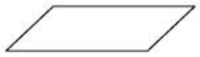
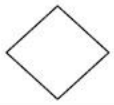
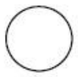
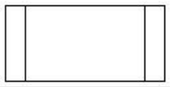
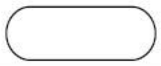
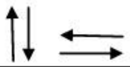
• A round-edged rectangle to represent starting and ending activities, which are sometimes referred to as terminal activities.

• A rectangle to represent an activity or step.

• Each step or activity within a process is indicated by a single rectangle, which is known as an activity or process symbol.

• A diamond to signify a decision point. The question to be answered or decision to be made is written inside the diamond, which is known as a decision symbol. The answer determines the path that will be taken as a next step.

• Flow lines show the progression or transition from one step to another.

Symbol	Name	Function
	**Process**	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

General Rules for flowcharting

1. All boxes of the flowchart are connected with Arrows. (Not lines)

2. Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.

3. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.

4. Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.

5. Connectors are used to connect breaks in the flowchart. Examples are:

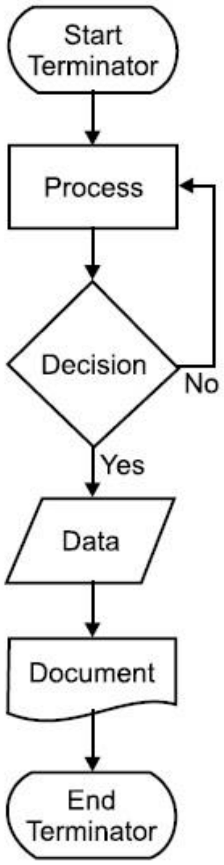
- From one page to another page.
- From the bottom of the page to the top of the same page.
- An upward flow of more than 3 symbols

6. Subroutines and Interrupt programs have their own and independent flowcharts.

7. All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.

8. All flowcharts end with a terminal or a contentious loop.

Flowcharting uses symbols that have been in use for a number of years to represent the type of operations and/or processes being performed. The standardized format provides a common method for people to visualize problems together in the same manner. The use of standardized symbols makes the flow charts easier to interpret. However, standardizing symbols is not as important as the sequence of activities that make up the process.



Flowcharting Tips

- Chart the process the way it is really occurring. Do not document the way a written process or a manager thinks the process happens.
- People typically modify existing processes to enable a more efficient process. If the desired or theoretical process is charted, problems with the existing process will not be recognized and no improvements can be made.
- Test the flow chart by trying to follow the chart to perform the process charted. If there is a problem performing the operation as charted, note any differences and modify the chart to correct. A better approach would be to have someone unfamiliar with the process try to follow the flow chart and note questions or problems found.
- Include mental steps in the process such as decisions. These steps are sometimes left out because of familiarity with the process, however, represent sources of problems due to a possible lack of information used to make the decision can be inadequate or incorrect if performed by a different person.

Now, we will discuss some examples on flowcharting. These examples will help in proper understanding of flowcharting technique. This will help you in program development process in next unit of this block.

Example 1. Design an algorithm and the corresponding flowchart for adding the test scores as given below:

26, 49, 98, 87, 62, 75

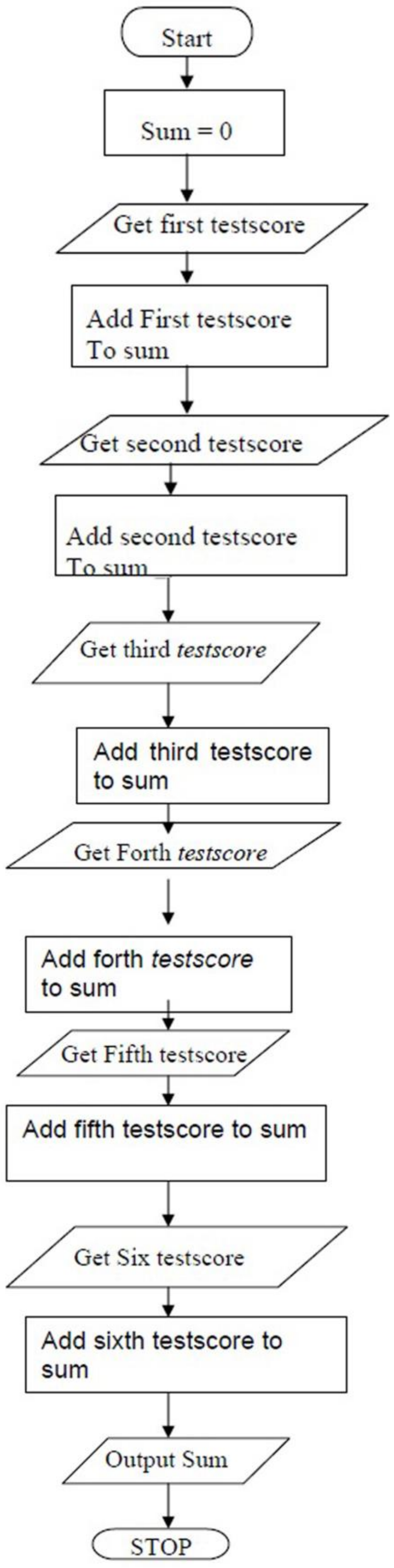
a) Algorithm

1. Start
2. Sum = 0
3. Get the first test score
4. Add first test score to sum
5. Get the second test score
6. Add to sum
7. Get the third test score
8. Add to sum
9. Get the Forth test score
10. Add to sum
11. Get the fifth test score
12. Add to sum
13. Get the sixth test score
14. Add to sum
15. Output the sum
16. Stop

b) The corresponding flowchart is as follows:

The algorithm and the flowchart here illustrate the steps for solving the problem of adding six test scores where one test score is added to sum at a time. Both the algorithm and flowchart should always have a **Start** step at the beginning of the algorithm or flowchart and at least one **stop** step at the end, or anywhere in the algorithm or flowchart.

Since we want the sum of six test score, then we should have a container for the resulting sum. In this example, the container is called **sum** and we make sure that sum should start with a zero value by step 2.

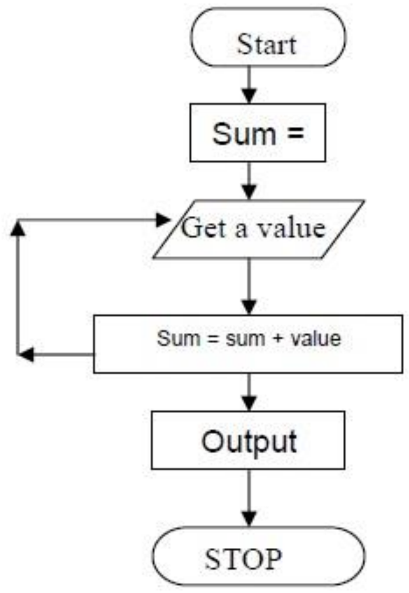

Start
Sum = 0
Get first testscore
Add First testscore To sum
Get second testscore
Add second testscore To sum
Get third testscore
Add third testscore to sum
Get Forth testscore
Add forth testscore to sum
Get Fifth testscore
Add fifth testscore to sum
Get Six testscore
Add sixth testscore to sum
Output Sum
STOP

Example 2**:** The problem with this algorithm is that, some of the steps appear more than once, i.e. step 5 get second number, step 7, get third number, etc. One could shorten the algorithm or flowchart as follows:

a) Algorithm

1. Start
2. Sum = 0
3. Get a value
4. sum = sum + value
5. Go to step 3 to get next Value
6. Output the sum
7. Stop

b) The corresponding flowchart is as follows:



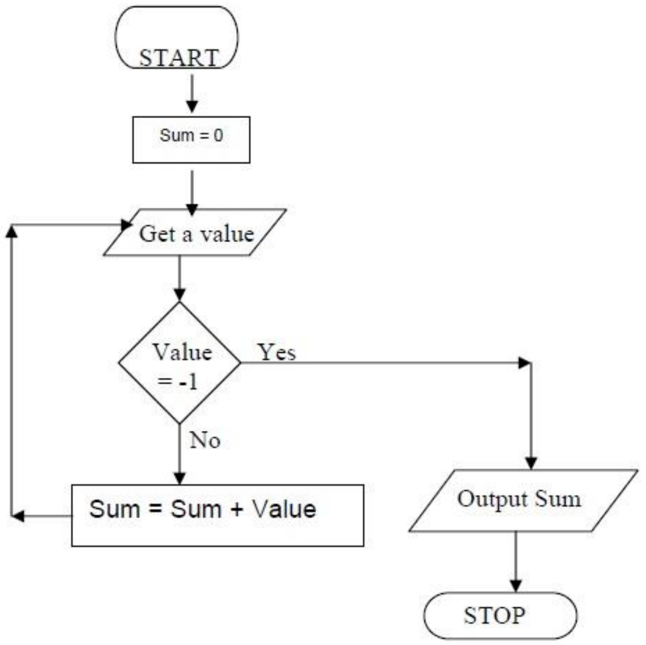
This algorithm and its corresponding flowchart are a bit shorter than the first one. In this algorithm, step 3 to 5 will be repeated, where a number is obtained and added to sum. Similarly the flowchart indicates a flowline being drawn back to the previous step indicating that the portion of the flowchart is being repeated. One problem indicates that these steps will be repeated endlessly, resulting in an **endless** algorithm or flowchart. The algorithm needs to be improved to eliminate this problem. In order to solve this problem, we need to add a last value to the list of numbers given. This value should be unique so that, each time we get a value, we test the value to see if we have reached the last value.

In this way our algorithm will be a finite algorithm which ends in a finite number of steps as shown below. There are many ways of making the algorithm finite.

The new list of numbers will be 26, 49, 498, 9387, 48962, 1, -1. The value –1 is a unique number since all other numbers are positive.

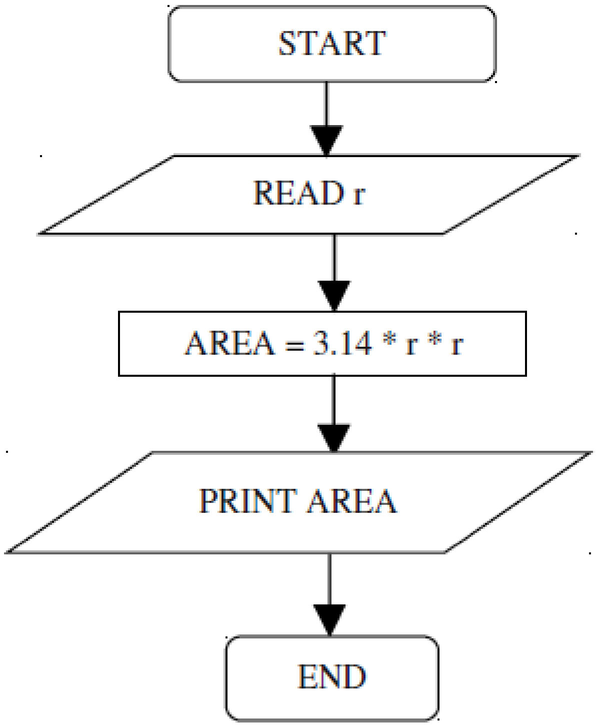
1. Start
2. Sum = 0
3. Get a value

4. If the value is equal to –1, go to step 7
5. Add to sum (sum = sum + value)
6. Go to step 3 to get next Value
7. Output the sum
8. Stop

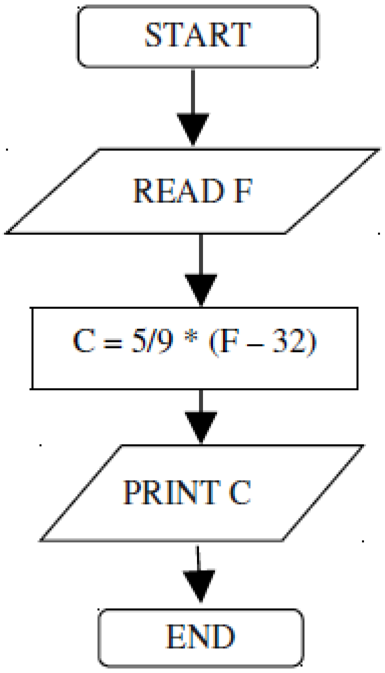


More examples for Flow Charts

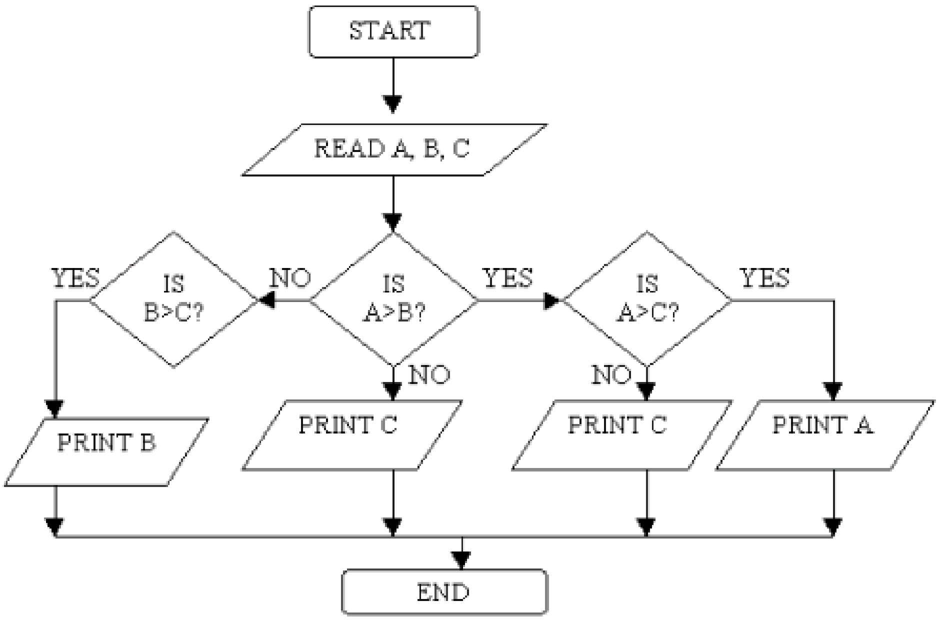
1: Find the area of a circle of radius r.



2: Convert temperature Fahrenheit to Celsius.



3. Find the largest of three numbers A, B, and C.



Advantages of using Flowcharts

As we discussed flow chart is used for representing algorithm in pictorial form. This pictorial representation of a solution/system is having many advantages. These advantages are as follows:

- A Flowchart can be used as a better way of communication of the logic of a system and steps involve in the solution, to all concerned particularly to the client of system.
- A flowchart of a problem can be used for effective analysis of the problem.
- Program flowcharts are a vital part of a good program documentation. Program document is used for various purposes like knowing the components in the program, complexity of the program etc.
- Once a program is developed and becomes operational it needs time to time maintenance. With help of flowchart maintenance become easier.
- Any design of solution of a problem is finally converted into computer program. Writing code referring the flowchart of the solution become easy.

Pseudocode

Pseudocode is one of the tools that can be used to write a preliminary plan that can be developed into a computer program. **Pseudocode** is a generic way of describing an algorithm without use of any specific programming language syntax. It is, as the name suggests, *pseudo* code —it cannot be executed on a real computer, but it models and resembles real programming code, and is written at roughly the same level of detail.

Pseudocode, by nature, exists in various forms, although most borrow syntax from popular programming languages (like **C**, **Lisp**, or **FORTRAN**). Natural language is used whenever details are unimportant or distracting.

Computer science textbooks often use pseudocode in their examples so that all programmers can understand them, even if they do not all know the same programming languages. Since pseudocode style varies from author to author, there is usually an accompanying introduction explaining the syntax used.

In the algorithm design, the steps of the algorithm are written in free English text and, although brevity is desired, they may be as long as needed to describe the particular operation. The steps of an algorithm are said to be written in pseudocode. Many languages, such as Pascal, have a syntax that is almost identical to pseudocode and hence make the transition from design to coding extremely easy.

The following section deal with the control structures (control constructs) Sequence, Selection and Iteration or Repetition.

Control Structures or Logical Structures

The key to better algorithm design and thus to programming lies in limiting the control structure to only three constructs. These are illustrated below:

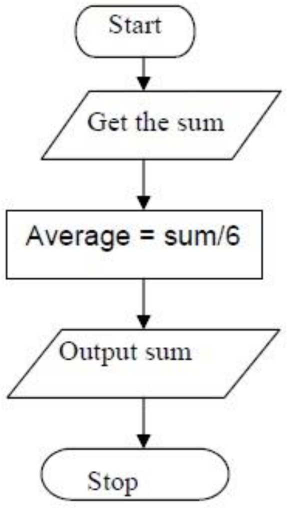
<u>The sequence structure</u>

The first type of control structures is called the sequence structure. This structure is the most elementary structure. The sequence structure is a case where the steps in an algorithm are constructed in such a way that, no condition step is required. The sequence structure is the logical equivalent of a straight line.

For example, suppose you are required to design an algorithm for finding the average of six numbers, and the sum of the numbers is given. The pseudocode will be as follows.

```
Start
Get the sum
Average = sum / 6
Output the average
Stop
```

The corresponding flowchart will appear as follows.



Example 1: This is the pseudo-code required to input three numbers from the keyboard and output the result.

```
Use variables: sum, number1, number2, number3 of type integer
Accept number1, number2, number3
Sum = number1 + number2 + number3
Print sum
End program
```

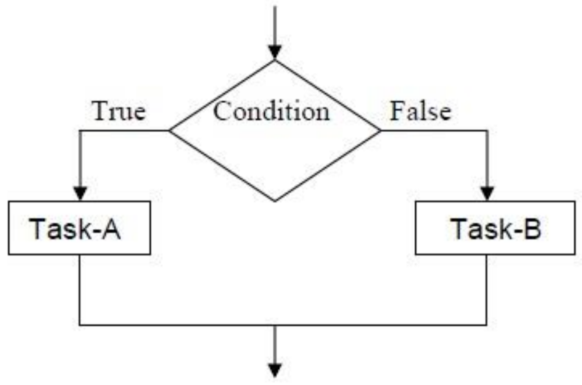
Example 2: The following pseudo-code describes an algorithm which will accept two numbers from the keyboard and calculate the sum and product displaying the answer on the monitor screen.

```
Use variables sum, product, number1, number2 of type real
display "Input two numbers"
accept number1, number2
sum = number1 + number2
print "The sum is ", sum
product = number1 * number2
print "The Product is ", product
end program
```

Decision Structure or Selection Structure

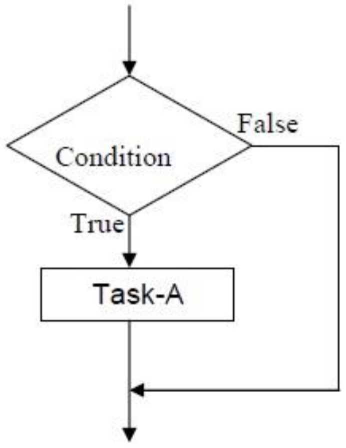
The decision structure or mostly commonly known as a selection structure, is case where in the algorithm, one has to make a choice of two alternatives by making decision depending on a given condition.

Selection structures are also called **case** selection structures when there are two or more alternatives to choose from. This structure can be illustrated in a flowchart as follows:



If condition is true
Then do task A
else
Do Task-B

In this example, the condition is evaluated, if the condition is true Task-A is evaluated and if it is false, then Task-B is executed. A variation of the construct of the above figure is shown below.



If condition is true then
Do Task-A

In this case, if condition is false, nothing happens. Otherwise Task-A is executed.

The selection requires the following.

- Choose alternative actions as a result of testing a logical condition
- Produce code to test a sequence of logical tests

Making Choices

There are many occasions where a program is required to take alternative actions. For example, there are occasions where we need to take action according to the user choice.

All computer languages provide a means of selection. Usually it is in the form of **If** statement and our pseudo-code is no exception to this.

We will use the if statement together with logical operators to test for true or false as shown below.

```
If a = b
print "a = b"
```

The action is only taken when the test is true.
The logical operators used in our pseudo-code are.

= is equal to
> is greater than
< is less than
>= is greater than or equal
<= is less than or equal
<> is not eaqual to

Example 1: The following shows how the selection control structure is used in a program where a user chooses the options for multiplying the numbers or adding them or subtracting.

```
Use variables: choice, of the type character
ans, number1, number2, of type integer
display "choose one of the following"
display "m for multiply"
display "a for add"
display "s for subtract"
accept choice
display "input two numbers you want to use"
accept number1, number2
if choice = m then ans = number1 * number2
if choice = a then ans = number1 + number2
if choice = s then ans = number1 - number2
display ans
```

Compound Logical Operators

There are many occasions when we need to extend the conditions that are to be tested. Often there are conditions to be linked.

In everyday language we say things like If I had the time and the money I would go on holiday. The and means that both conditions must be true before we take an action. We might also say I am happy to go to the theatre or the cinema. The logical link this time is or . Conditions in if statements are linked in the same way. Conditions linked with and only result in an action when all conditions are true. For example, if a >b and a > c then display "a is the largest". Conditions linked with an or lead to an action when either or both are true.

Example 1: The program is to input a examination mark and test it for the award of a grade. The mark is a whole number between 1 and 100. Grades are awarded according to the following criteria:

>= 80 Distinction
>= 60 Merit
>= 40 Pass
< 40 fail

The pseudo-code is

```
Use variables: mark of type integer
If mark >= 80 display "distinction"
If mark >= 60 and mark < 80 display "merit"
If mark >= 40 and mark < 60 display "pass"
If mark < 40 display "fail"
```

An **if** statement on its own is often not the best way of solving problems. A more elegant set of conditions can be created by adding an **else** statement to the if statement. The **else** statement is used to deal with situations as shown in the following examples.

Example 7: A person is paid at top for category 1 work otherwise pay is at normal rate.

```
If the work is category 1
pay-rate is top
Else
pay-rate is normal
```

The else statement provides a neat way of dealing with alternative condition. In pseudocode we write

```
If work = cat1 then p-rate: = top
Else p-rate = normal
```

Or

```
If work = cat1 then
p-rate: = top
Else
p-rate = normal
```

The following example illustrate the use of if … else statements in implementing double alternative conditions.

```
If salary < 50000 then
Tax = 0
Else
If salary > 50000 AND salary < 100000 then
Tax = 50000 * 0.05
Else
Tax = 100000 * 0.30
```

The case statement

Repeating the if … else statements a number of times can be somewhat confusing. An alternative method provided in a number of languages is to use a selector determined by the alternative conditions that are needed. In our pseudo-code, this will called a **case statement**.

Example 8: The following program segment outputs a message to the monitor screen describing the insurance available according to a category input by the user.

```
Use variables: category of type character
Display “input category”
Accept category
If category = U
Display “insurance is not available”
Else
If category = A then
Display “insurance is double”
Else
If category = B then
Display “insurance is normal”
Else
If category = M then
Display “insurance is medically dependent”
Else
Display “entry invalid”
```

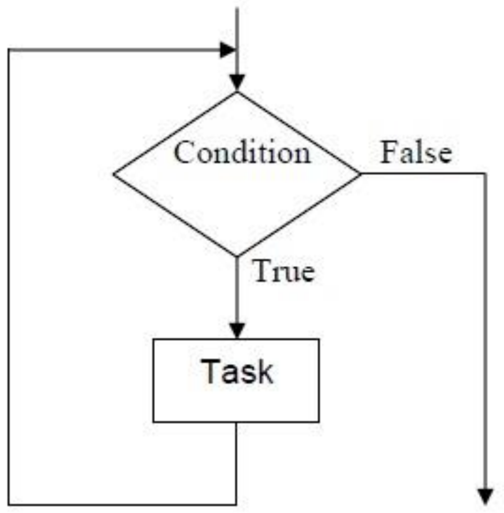
This can be expressed in a case statement as follows:

```
Use variables: category of type character
Display “input category”
Accept category
DO case of category
CASE category = U
Display “insurance not available”
CASE category = A
Display “insurance is double”
CASE category = B
Display “insurance is normal”
CASE category = M
Display “insurance is medically dependent”
OTHERWISE
Display “entry is invalid”
ENDCASE
```

Instead of using the word *otherwise*, one can use *else*.

Repetition or Iteration Structure

A third structure causes the certain steps to be repeated.



The Repetition structure can be implemented using

- Repeat Until Loop
- The While Loop
- The For Loop

Any program instruction that repeats some statement or sequence of statements a number of times is called an **iteration** or a **loop**. The commands used to create iterations or loops are all based on logical tests. There three constructs for iterations or loops in our pseudocode.

<u>The Repeat Until loop</u>

The syntax is;

```
REPEAT
A statement or block of statements
UNTIL a true condition
```

Example 1: A program segment repeatedly asks for entry of a number in the range 1 to 100 until a valid number is entered.

```
REPEAT
DISPLAY "Enter a number between 1 and 100"
ACCEPT number
UNTIL number < 1 OR number > 100
```

Example 2: A survey has been carried out to discover the most popular sport. The results will be typed into the computer for analysis. Write a program to accomplish this.

```
REPEAT
DISPLAY "Type in the letter chosen or Q to finish"
DISPLAY "A: Athletics"
DISPLAY "S: Swimming"
DISPLAY "F: Football"
DISPLAY "B: Badminton"
DISPLAY "Enter data"
ACCEPT letter
If letter = 'A' then
Athletics = athletics + 1
If letter = 'S' then
Swimming = Swimming + 1
If letter = 'F' then
Football = Football + 1
If letter = 'B' then
Badminton = Badminton + 1
UNTIL letter = 'Q'
DISLAY "Athletics scored", athletics, "votes"
DISLAY "Swimming scored", swimming, "votes"
DISLAY "Football scored", football, "votes"
DISLAY "Badminton scored", Badminton, "votes"
```

The WHILE loop

The second type of iteration we will look at is the while iteration. This type of conditional loop tests for terminating condition at the beginning of the loop. In this case no action is performed at all if the first test causes the terminating condition to evaluate as false.

The syntax is;

```
WHILE (a condition is true)
A statement or block of statements
ENDWHILE
```

Example 1: A program segment to print out each character typed at a keyboard until the character 'q' is entered.

```
WHILE letter <> 'q'
ACCEPT letter
DISPLAY "The character you typed is", letter
ENDWHILE
```

Example 2: Write a program that will output the square root of any number input until the number input is zero. In some cases, a variable has to be initialized before execution of the loop as shown in the following example.

```
Use variable: number of type real
DISPLAY "Type in a number or zero to stop"
ACCEPT number
WHILE number <> 0
Square = number * number
DISPLAY "The square of the number is", square
DISPLAY "Type in a number or zero to stop"
ACCEPT number
ENDWHILE
```

The FOR Loop

The third type of iteration, which we shall use when the number of iterations is known in advance, is a **for loop**. This, in its simplest form, uses an initialization of the variable as a starting point, a stop condition depending on the value of the variable. The variable is incremented on each iteration until it reaches the required value.

The syntax is;

```
FOR (starting state, stopping condition, increment)
Statements
ENDFOR
```

Example 1:

```
FOR (n = 1, n <= 4, n + 1)
DISPLAY "loop", n
ENDFOR
```

The fragment of code will produce the output

```
Loop 1
Loop 2
Loop 3
Loop 4
```

In the example, n is usually referred as the loop variable, or counting variable, or index of the loop. The loop variable can be used in any statement of the loop. The variable should not be assigned a new value within the loop, which may change the behavior of the loop.

Example 2: Write a program to calculate the sum and average of a series of numbers.

The pseudo-code solution is;

```
Use variables: n, count of the type integer
Sum, number, average of the type real
DISPLAY "How many numbers do you want to input"
ACCEPT count
SUM = 0
FOR (n = 1, n <= count, n + 1)
DISPLAY "Input the number from your list"
ACCEPT number
SUM = sum + number
ENDFOR
Average = sum / count
DISPLAY "The sum of the numbers is ", sum
DISPLAY "Average of the numbers is ", average
```

Flowcharts have been used in this section to illustrate the nature of the three control structures. These three are the basic control structures out of which all programs are built. Beyond this, flowcharts serve the programmer in two distinct ways: as problem solving tools and as tools for documenting a program.

Example 3:

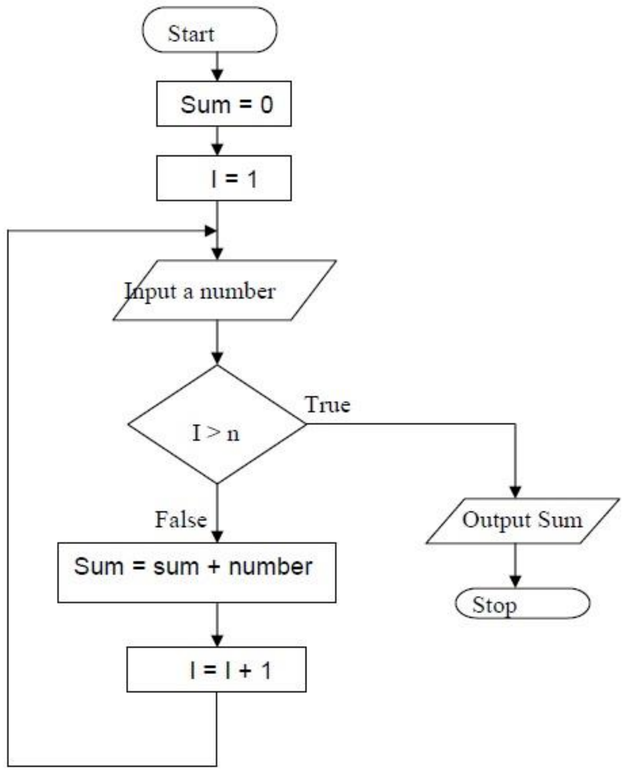
Design an algorithm and the corresponding flowchart for finding the sum of n numbers.

Pseudocode Program

```
Start
Sum = 0
Display “Input value n”
Input n
For(I = 1, n, 5)
Input a value
Sum = sum + value
ENDFOR
Output sum
Stop
```

In this example, we have used I to allow us to count the numbers, which we get for the addition. We compare I with n to check whether we have exhausted the numbers or not in order to stop the computation of the sum (or to stop the iteration structure). In such a case, I is referred to as a **counter**.

The corresponding flowchart will be as follows:


Start
Sum = 0
I = 1
Input a number
I > n
True
False
Output Sum
Stop
Sum = sum + number
I = I + 1