INTRODUCTION TO ALGORITHMS

OBJECTIVES

After completing this chapter, you will be able to:

- Understand the basics and usefulness of an algorithm,
- Analyze various algorithms,
- Understand a flowchart and its advantages and limitations,
- Steps involved in designing a program.

INTRODUCTION

A computer is a useful tool for solving a great variety of problems. To make a computer do anything (i.e. solve a problem), you have to write a *computer program*. In a computer program you tell a computer, step by step, exactly what you want it to do. The computer then executes the program, following each step mechanically, to accomplish the end goal.

In mathematics, computer science, and related subjects, an *algorithm* is a finite sequence of steps expressed for solving a problem. An *algorithm* can be defined as "a process that performs some sequence of operations in order to solve a given problem". Algorithms are used for calculation, data processing, and many other fields.

In computing, algorithms are essential because they serve as the systematic procedures that computers require. A good algorithm is like using the right tool in the workshop. It does the job with the right amount of effort. Using the wrong algorithm or one that is not clearly defined is like trying to cut a piece of plywood with a pair of scissors: although the job may get done, you have to wonder how effective you were in completing it.

DEFINITION OF ALGORITHM

An algorithm is a sequence of instructions that tell how to solve a particular problem or a given set of problems. This description or procedure takes some value or set of values as input, and produces some value or set of values as output. A set of instructions is not an algorithm if it does not have a definite stopping place, or if the instructions are too vague to be followed clearly. The stopping place may be at variable points in the general procedure, but something in the procedure must determine precisely where the stopping place is for a particular case.

Let us follow an example to help us understand the concept of algorithm in a better way. Let's say that you have a friend arriving at the railway station, and your friend needs to get from the railway station to your house. Here are three different ways (algorithms) that you might give your friend for getting to your home.

The taxi algorithm:

- Go to the taxi stand.
- Get in a taxi.
- Give the driver my address.

The call-me algorithm:

- When your train arrives, call my mobile phone.
- Meet me outside the railway station.

The bus algorithm:

- Outside the railway station, catch bus number 321.
- Transfer to bus 308 near Kurla station.
- Get off near Kalina University.
- Walk two blocks west to my house.

All these three algorithms accomplish the same goal, but each algorithm does it in a different way. Each algorithm also has a different cost and a different travel time. Taking a taxi, for example, is the fastest way, but also the most expensive. Taking the bus is definitely less expensive, but a whole lot slower. You choose the algorithm based on the circumstances.

In computer programming, there are often many different algorithms to accomplish any given task. Each algorithm has advantages and disadvantages in different situations. Sorting is one place where a lot of research has been done, because computers spend a lot of time sorting lists.

PROPERTIES OF ALGORITHMS

A good algorithm should have the following properties:

- It must be explicit: it must be clear and obvious
- It must be precise: it must be specified exactly and accurately
- It must not be ambiguous: there should be no doubt about what to do next
- It must be effective: it must produce a result
- It must be finite: it must have a beginning and an end

REPRESENTATION OF ALGORITHMS

When an algorithm is written, a certain language that best describes the solution is used. The language used does not obey the rules of a particular programming language. One should feel free to write in a non-ambiguous way every instruction of the algorithm regardless of constrains imposed by the syntax rules of a particular programming language.
Some of the ways in which algorithms can be represented are, **pseudo code** and **flow charts**.

PSEUDO CODE

A pseudo code is an outline of a computer program, written in a mixture of a programming language and English. Writing pseudo code is one of the best ways to plan a computer program.
For example, here is a pseudo code of an algorithm that reads two numbers and says which is greater:

```
Start
Get A
Get B
Result = A + B
Print Result
End
End
```

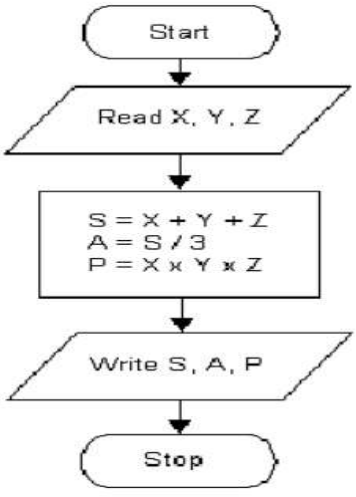
```
Start
largest = L0
for each item in the list (Length(L) ≥ 1), do
if the item ≥ largest, then
largest = the item
return largest
End
```

The advantage of pseudo code is that it allows the programmer to concentrate on how the program works while ignoring the details of the language.

FLOW CHARTS

A flow chart is a diagram that uses symbols and words to describe an algorithm. Each step in the flowchart is followed by an arrow that indicates which step to do next. The following symbols are used in flow charting:

Elongated circle (terminator): indicates the beginning/end or start/stop point
Rectangle (process): indicates an operation or an action step
Rhombus or diamond (decision): indicates a question or branch in the process (conditional statement)
Arrow (flow line): indicates the direction of flow
Parallelogram (data I/O): indicates data inputs and outs to and from a process
Display: displays information on the screen
Document: prints a document or report Sub-routine
Example of a flow chart:



Advantages of Using Flowcharts:

The *benefits of flowcharts* are as follows:

- ***Communication*:** Flowcharts are better way of communicating the logic of a system to all concerned.
- ***Effective analysis*:** With the help of flowchart, problem can be analyzed in more effective way.
- ***Proper documentation*:** Program flowcharts serve as a good program documentation, which is needed for various purposes.
- ***Efficient Coding*:** The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
- ***Proper Debugging*:** The flowchart helps in debugging process.
- ***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:

Although a flowchart is a very useful tool, there are a few limitations in using flowcharts which are listed below:

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

When to Use a Flowchart:

- To communicate to others how a process is done.
- A flowchart is generally used when a new project begins in order to plan for the project.
- A flowchart helps to clarify how things are currently working and how they could be improved. It also assists in finding the key elements of a process, while drawing clear lines between where one process ends and the next one starts.
- Developing a flowchart stimulates communication among participants and establishes a common understanding about the process. Flowcharts also uncover steps that are redundant or misplaced.
- Flowcharts are used to help team members, to identify who provides inputs or resources to whom, to establish important areas for monitoring or data collection, to identify areas for improvement or increased efficiency, and to generate hypotheses about causes.

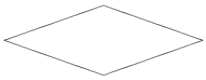
- ❖ It is recommended that flowcharts be created through group discussion, as individuals rarely know the entire process and the communication contributes to improvement.
- ❖ Flowcharts are very useful for documenting a process (simple or complex) as it eases the understanding of the process.
- ❖ Flowcharts are also very useful to communicate to others how a process is performed and enables understanding of the logic of a process

Flowchart Symbols & Guidelines

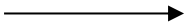
a. **Elongated circle:** indicate the beginning (start) or end (stop) of the algorithm

b. **Rectangle:** indicates instructions or actions

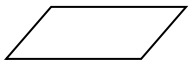
c. **Diamond:** indicates a decision that has to be made



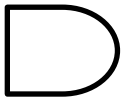
d. **Arrow:** indicates the direction of flow



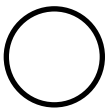
e. **Parallelogram:** indicates data input and output

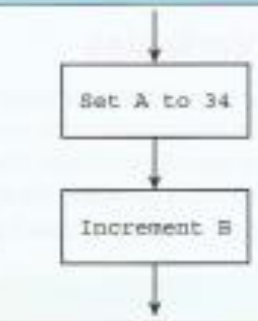
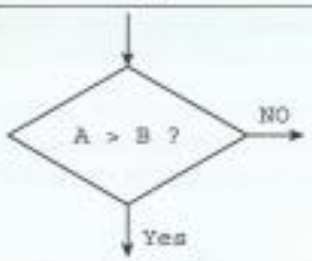
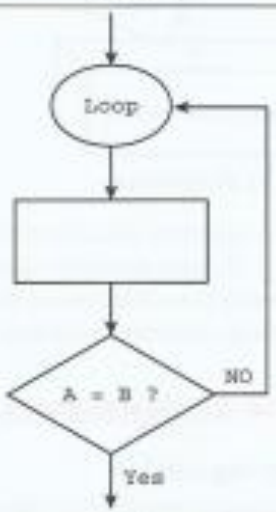
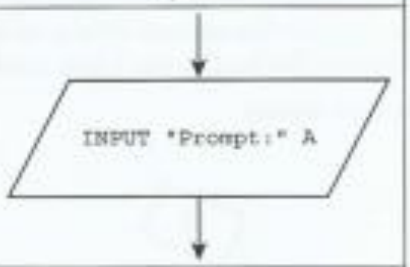
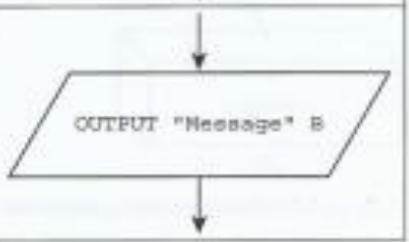


f. **Delay**: used to indicate a delay or wait in the process for input from some other process.



g. Connector : connector symbol is used to connect different flow lines.

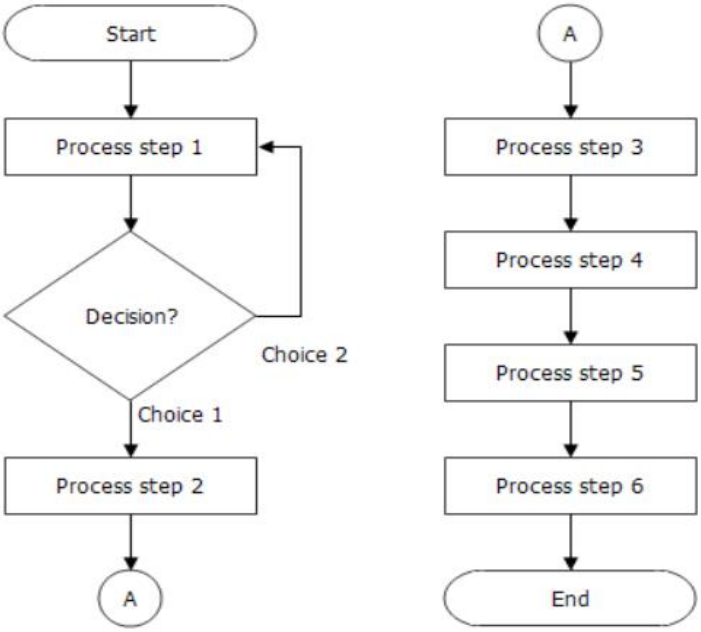


	Structured English	Pseudocode	Flowchart
Assignment and Sequence	SET A TO 34 INCREMENT B	A ← 34 B ← B + 1	
Selection	IF A IS GREATER THAN B THEN ... ELSE ...	IF A > B THEN ... ELSE ... ENDIF	
Repetition	REPEAT UNTIL A IS EQUAL TO B ...	REPEAT ... UNTIL A = B	
Input	INPUT A	INPUT "Prompt:" A	
Output	OUTPUT "Message" OUTPUT B	OUTPUT "Message" B	

These are the basic symbols used generally. Now, the ***basic guidelines*** for drawing a flowchart with the above symbols are that:

- In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
- The flowchart should be neat, clear and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
- The flowchart is to be read left to right or top to bottom.
- A process symbol can have only one flow line coming out of it.
- For a decision symbol, only one flow line can enter it, but multiple lines can leave it to denote possible answers.
- The terminal symbols can only have one flow line in conjunction with them

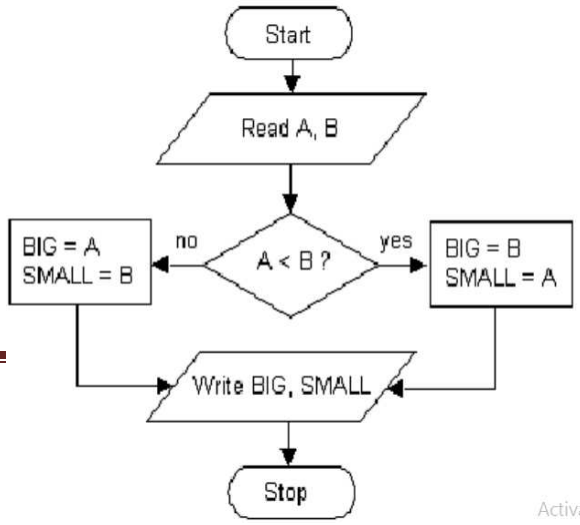
Example of flowchart is as shown below.



Example:

Consider another problem of finding the largest number between A and B Algorithm for the above problem is as follows:

```
GET A, B
If A is less than B
BIG=B
SMALL = A
Else
BIG=A
SMALL = B
Display BIG, SMALL
```



ELEMENTARY DATA AND DATA TYPES

Variables, Constants and Literals

a. **Variable**: an object in a program whose value can be modified during the execution of the program. In the above flow chart, x, y and z are variables. it can also be defined as a storage location for a data value that has an identifier.
b. **Constant**: an object whose value cannot be modified in the course of the algorithm or program. A constant is given a value that remains the same all through the program.

Variables and constants are characterized by:

- an identifier: which is the name of the object
- a value: which is the content of the object
- a type: which defines the domain in which the object gets its value

c. Literal: anything (numbers or text) that is usually written within double quotes. For example, "Enter a number", "The result is".

Data Types

A data type defines the domain in which an object gets its value and the kind of operations that can be performed on the object. The basic data types are:

- Integer: used to store positive, negative whole number and zero
- Real: used to store real numbers both positive and negative
- Boolean: used to store logic values like true or false
- Character: used to store a single character or digit.
- String: used to store a word or phrase e.g. the word hamburger will be stored as a string

Basic Instructions

Three basic instructions used in an algorithm are input, output and assignment instructions:

- An input instruction allows information to be typed from the keyboard. Example: read (a, b), get (number)
- An output instruction allows:
  - ✓ display of information on the screen
  - ✓ printing of information on paper

  For example: write "a is greater than b" or print "b is greater than a"

- The assignment statement allows a value to be assigned to a variable. A variable can be assigned the content of another variable, a constant, a literal, an arithmetic or Boolean expression. The symbol used is ←.

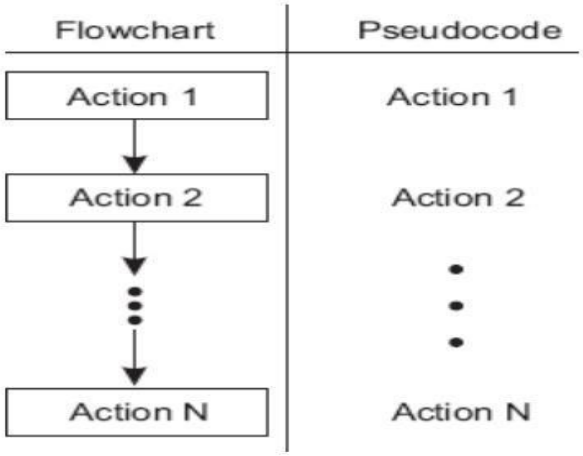
Examples: $z \leftarrow x/y$, $sum \leftarrow a + b$, $total \leftarrow total + 1$, $pi \leftarrow 3.14$

In an assignment statement, the value to the right is assigned to the variable in the left.
In the case of $sum \leftarrow a + b$, "$a + b$" is calculated and the result is assigned (kept) in the variable sum.
For $total \leftarrow total + 1$, "$total + 1$" is calculated and the result is assigned to the variable total, meaning that the value of total has been increased by 1.
E.g. Let $total = 3$
$total \leftarrow total + 1 \Rightarrow total = 3 + 1 = 4$.

BASIC ALGORITHMIC CONTROL STRUCTURES

Sequence Control Structure

A sequence control structure executes a set of instructions one after the other from the first to the last in the order they are given.

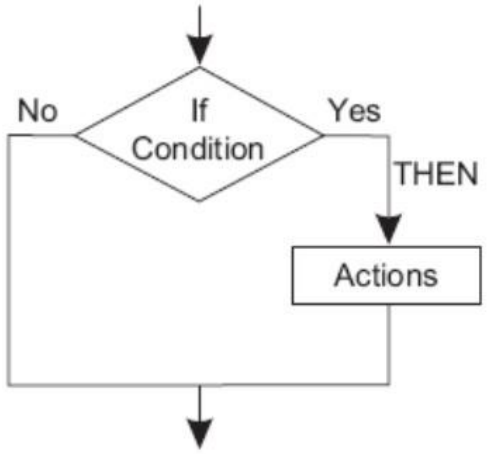


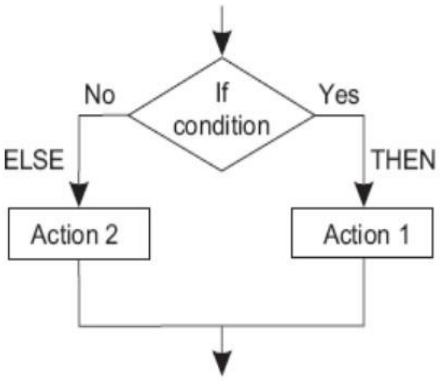
Selection Control Structure

A selection control structure (condition control structure) chooses the instruction or set of instructions to be executed based on the validity of a certain condition. Examples: If …then else and case … of.

a If… then … else

Syntax: IF condition THEN
Instruction 1
ELSE
Instruction 2

Flowchart	Pseudocode
	• • • IF condition THEN List of actions ENDIF • • •



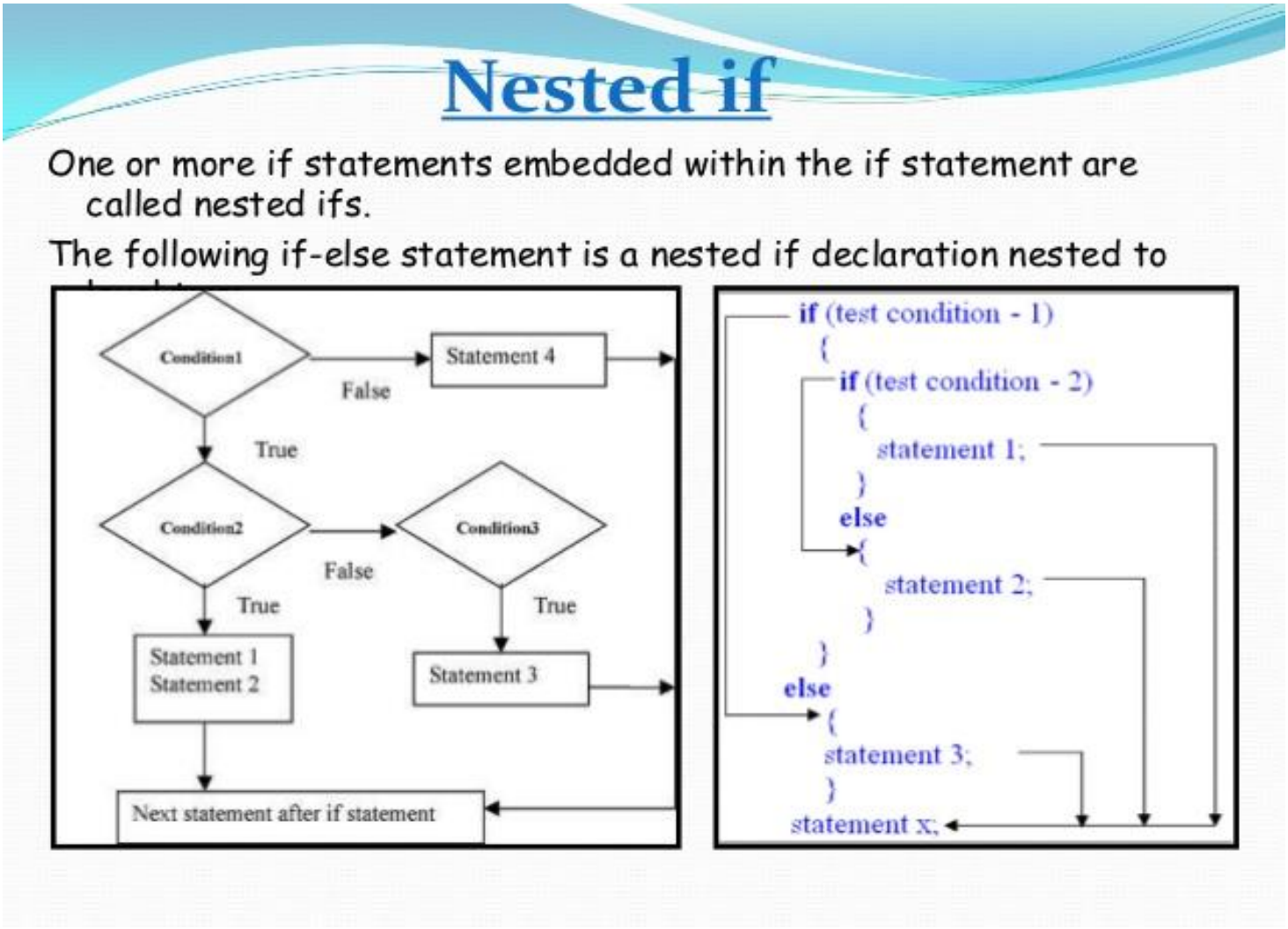
Condition is a Boolean expression meaning that it can take only one of two values true or false. The condition is evaluated, if it is true, instruction 1 is executed. If it is false, instruction 2 is executed.

Note that instructions 1 and 2 could be compound instructions.

It is possible to nest many selection structures.

Syntax: **If** condition1 **then**
If condition2 **then**
Instruction 1
Else
Instruction 2
Else
Instruction 3

If condition1 is true, we move to condition2. If condition2 is true, then instruction 1 is executed otherwise, instruction 2 is executed. If condition 1 is false, instruction 3 is executed. Instruction 1 or instruction 2 will be executed if and only if condition 1 is true.

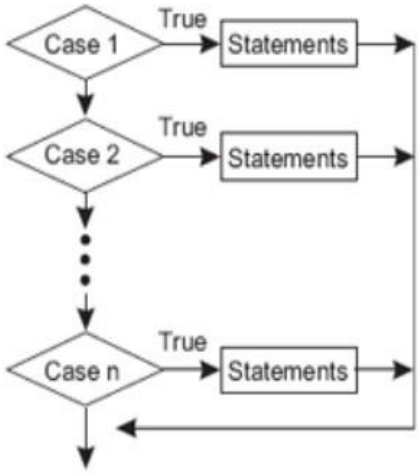


b Case … of

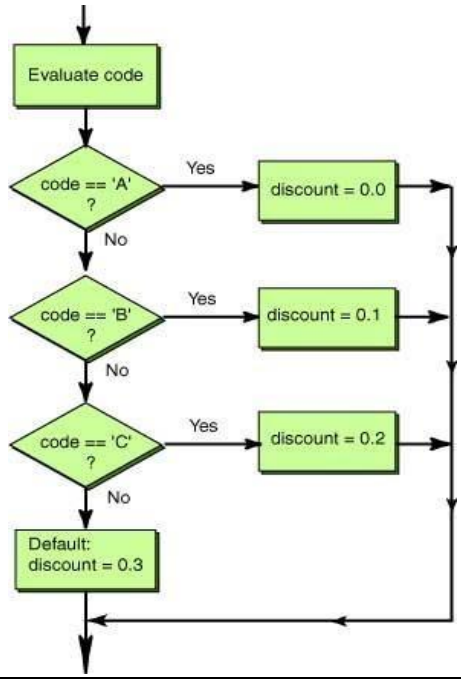
Syntax: **Case** variable **of**

Case 1: Instruction 1
Case 2: Instruction 2
…
Case n: Instruction n
End

The value of variable is evaluated, if it matches with case 1, instruction 1 is executed. If it matches with case 2, instruction 2 is executed and so on. Case…of is a multiple selection structure. It is used when an important number of choices are to be considered depending on the value of a variable.

Flowchart	Pseudocode
	CASE expression OF Condition 1: Sequence 1 Condition 2: Sequence 2 • • Condition n: Sequence n OTHERS : default sequence ENCASE

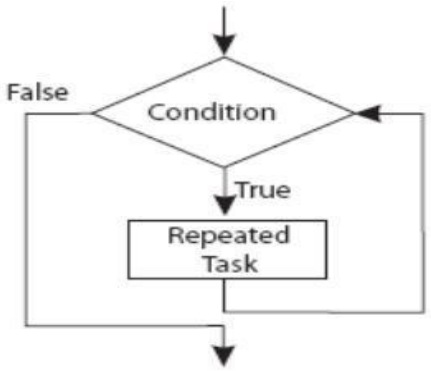
Example: ***To assign discount according to the code***

Flowchart	Pseudocode
	START READ code CASE Grade OF A : discount = 0.0 B : discount = 0.1 C : discount = 0.2 OTHERS : discount = 0.3 ENDCASE DISPLAY discount STOP

Repetition Control Structure

The repetition (iteration) control structure executes an instruction or set of instructions many times until a certain condition is reached or while a condition is true. Repetition structures define the order of operations and the number of repetitions. They are also known as loops. Examples are, while…do, repeat…until, for…to…do.

a. While Loop

Flowchart	Pseudocode
	WHILE condition is True statements ENDWHILE

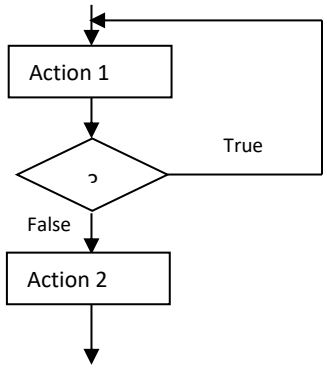
The condition is evaluated, if it is true instruction(s) is/are executed. Instruction(s) is/are executed as long as condition remains true. When the condition becomes false, the loop stops.
The condition for the loop to stop comprises of a variable called control or iteration variable whose value must change at the end of each execution of the loop. In the example above, the control variable is "i".

b. Repeat Loop

Syntax: **Repeat**
Instruction(s)
Until condition

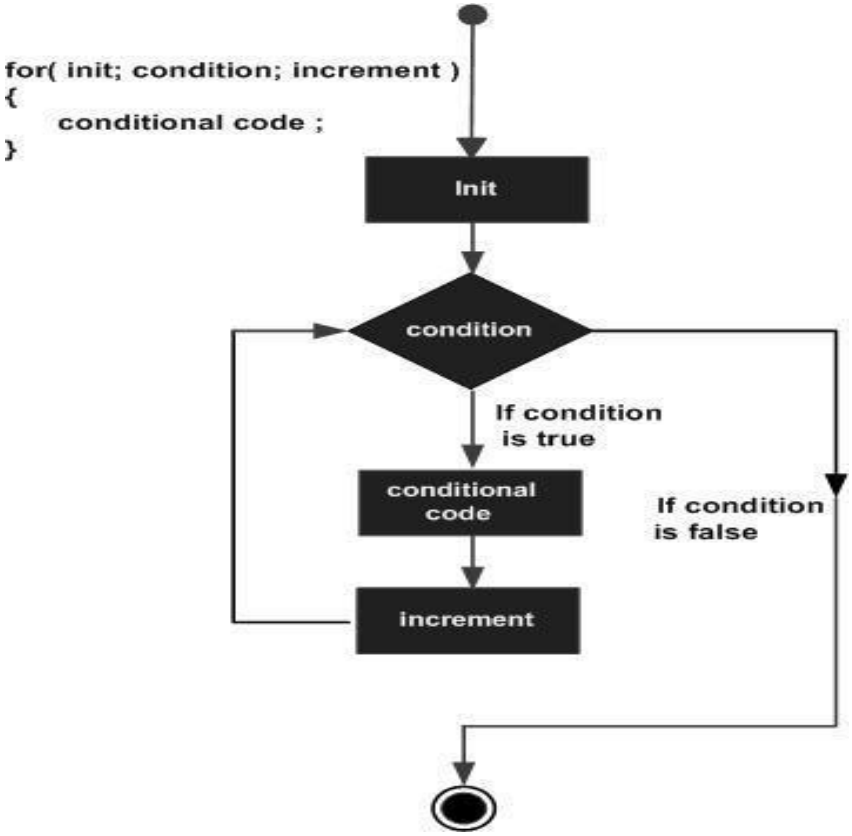
E x.

```
Get n
i ← 1
Repeat
   Print "this is a repeat loop"
   i ← i + 1
  Until i <= n
```



The instruction or set of instructions is executed and the condition is evaluated.
If it evaluates to false, the instruction or set of instructions is executed again. If condition evaluates to true, the program exits the loop.
Remark! The Repeat until loop must be executed at least once as the condition is evaluated only at the end of the loop.

c. For Loop


for(init; condition; increment)
{
conditional code ;
}
Init
condition
If condition
is true
conditional
code
If condition
is false
increment