1   Programming

The chapter concerns the following;

- How to analyze a problem and develop an algorithm
- Control structures and their use
- Drawing flow charts, writing pseudo codes and conversions between them
- Finding alternate solutions to a problem
- Programming in Pascal
- Evolution of programming languages

1.1 Analyzing a problem

The raw materials that are used to solve a problem are known as the 'input'. The result obtained after solving a problem is known as the 'output'. Converting input to output is called the 'process'. A process takes place step by step and it is very important to understand the order of the process. When analyzing a problem, the input, process and output are identified separately.

Example -

Problem 1 :   Preparing a letter which can be posted.

Input :   A sheet of paper suitable to write the letter on and a pen
An envelope and a stamp
Glue

Process :   1. Writing the letter
2. Folding the letter and putting it into the envelope
3. Pasting the envelope
4. Writing the recipient's address on the envelope
5. Sticking the stamp

Output :   A letter ready to be posted.
Note: Steps No. 4 and 5 in this process can be interchanged. However, the other steps should be followed in the order indicated.

Problem 2 : Making a cup of tea

Input : Tea leaves, sugar, hot water

Process :
1. Putting tea leaves in the strainer
2. Pouring hot water to the cup through the strainer
3. Adding some sugar to the cup
4. Stirring it well with a spoon
5. Testing for taste, taking a small sip from the cup
6. If the taste is not satisfactory, go to step 3 and repeat step Nos. 4 and 5

Output : A cup of tea

Problem 3 : Dividing 40 page and 80 page books from a parcel of books between two siblings - Sanduni and Anupama.

Input : The parcel of books

Process :
1. Opening the book parcel
2. Taking a book out from the parcel
3. If it is a 40 page book, giving it to Sanduni
4. If it is a 80 page book, giving it to Anupama
5. Go to Step No. 2 till all the books are taken out of the parcel

Output : Sanduni getting 40 page books
Anupama getting 80 page books

Problem 4 : Adding two numbers

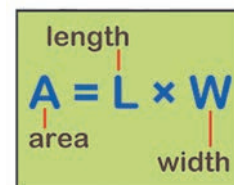**Input :** Two numbers

Process : Adding the two numbers

Output : Total

Problem 5 : Finding the area of a rectangle

Input : Length and width of the rectangle

Process : Area = Length x Width

Output : Area

Problem 6 : Finding the larger number between two numbers

Input :   Two numbers
Process :   Comparing the two numbers, finding the larger one
Output :   Larger number

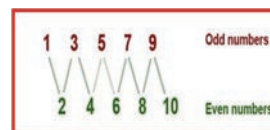**Problem 7 :** Finding whether a number is odd or even

Input :   Number

Process :   Dividing the number by 2

Deciding that the number is even if the remainder = 0

Deciding that the number is odd if the remainder = 1

Output :   Indicating whether the number is odd or even
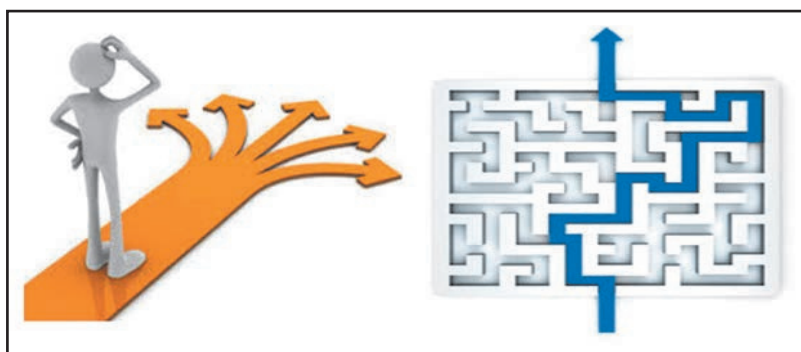
Activity

1. Identify the input, process and output related to dividing 100 toffees among 20 people.
2. Identify the input, process and output in making a kite.

What are alternative solutions?

If there is more than one solution to a given problem, such solutions are called alternative solutions. Such solutions depend on the nature of the problem.

Example

Imagine you come to school by school bus. If the bus breaks down on your way to school, you certainly will think of other alternative ways to reach school. Thus, you will think of **alternative solutions** to reach school.

1. Coming to school by another school bus which goes to your school
2. If you have money, reaching the school by CTB or private bus
3. Walking to school along the road
4. Walking to school using a short-cut
5. If you have a way to inform your parents, get their support to reach the school
6. Reaching the school by car or motor bike with the support of a trustworthy person

You may select a good solution out of these if it is mandatory to go to school that day.

Thus, if there are many solutions (set) to a particular problem, it is suitable to consider these and select an appropriate solution.

All the solutions pertaining to a problem are called **solution space**. In computer programming also, various solutions should be identified and an appropriate solution should be selected. Then, we can create a short, simple program.

Example 1

Let us examine the solution space to find the perimeter of a rectangle.

Let us analyze the input, process and output related to this problem.

Input   : Length and width of the rectangle

Process : Calculating the perimeter

Output : Indicating the perimeter

Let us examine the solution space to calculate the perimeter.

1st solution	Perimeter = length + width + length + width
2nd solution	Perimeter = length × 2 + width × 2
3rd solution	Perimeter = (length + width) × 2

Out of these solutions, a person who has knowledge only of addition, can select the 1st solution as the most appropriate. A person who has knowledge of multiplication and addition can select the 3rd solution out of the 2nd and 3rd solutions as the most appropriate. The reason for this is, it has the minimum number of additions and multiplications.

Example 2

Indicating that a student has failed if the score for the ICT Subject is less than 35, and indicating pass if the score is 35 or above.

Input : Marks

Process : Comparing the mark scored with 35

    Solution 1.   If the mark is less than 35

                    Result = Fail

                    If not

                    Result = Pass

    Solution 2.   If the mark is 35 or more than 35

                    Result = Pass

                    If not

                    Result = Fail

Output : Fail or Pass

Example 3

Finding the larger number from between two numbers *(See problem 6 in page 3)*

Let us consider the two input numbers as *n1* and *n2*.

    Solution 1.   If *n1* is larger than *n2,* the larger number will be *n1*.

                    If *n2* is larger than *n1*, the larger number will be *n2*.

    Solution 2.   Subtract *n2* from *n1*.

                    If the result is more than *0, n1* is the larger number.

                    If the result is less than *0, n2* is the larger number.

Thus, it is important to select the appropriate solution out of the available alternative solutions.

1.2 Problem Solving using Algorithms

An algorithm is a method to show the steps in solving a problem. An algorithm is a step-by-step procedure for solving a problem. The need for this is to present a way to solve the problem with a plan.

Example 1 - Let us develop an algorithm to post a letter.

(1) Writing the letter

(2) Folding the letter

(3) Inserting the letter in an envelope

(4) Writing the address

(5) Sticking the stamp

(6) Posting the letter

Step (1), (2) and (3) in this algorithm should be followed in the given order. Step (4) and (5) can be interchanged. The reason for this is, you can either stick the stamp after writing the address or write the address after sticking the stamp.

Thus, there are steps in an algorithm which should be followed in a strict sequential order. Sometimes, if the order of some steps is changed, it does not affect the process and the output is same.

Example 2 - Let us consider steps in measuring 500g of sugar using a scale.

(1) Putting sugar into a bag

(2) Placing the bag on the scale and getting the reading on the scale

(3) If the weight of the sugar is less than 500g, add sugar till it weighs 500g

(4) If the weight of the sugar is more than 500g, remove sugar from the bag till it weighs 500g

(5) Remove the bag of sugar from the scale when the weight is 500g

The algorithm to measure 500g sugar is given above.

> ### Activity
>
> There are 183 students in a primary school. The principal has decided to hold an inter-house sports meet dividing them into three houses – Olu, Nelum and Manel. Develop an algorithm to the devide the students into the three houses.

1.2.1 Control Structures

Three types of control structures are used in an algorithms.

    i.   **Sequence**

    ii. **Selection**

    iii. **Repetition**

i. Sequence

If the steps from the beginning to the end of an algorithm are carried out in a strict order, it is called a sequence.

Example -

1. Climbing up or down step by step when going on a staircase

2. Students who were admitted to grade 1 of the school continue studies till grade 13
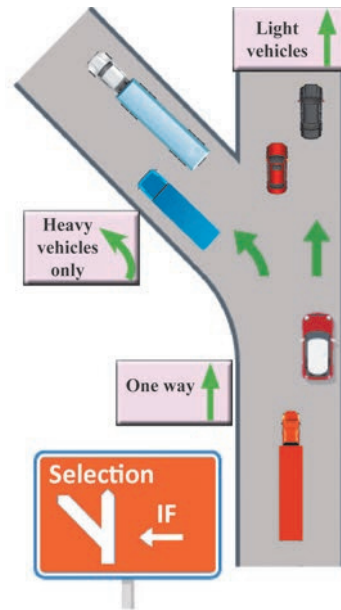
Activity

Write down two incidents which consist of sequences.

ii. Selection

Selection is a situation where step(s) are executed depending on whether a condition of an algorithm is satisfied or not. There are two choices; if the condition is satisfied, one is selected and if it is not satisfied, the other selection is selected.

Examples of selection

1. Admitting a child to Grade 1:

   If a child is below 5 years as at 31st of January that year

   > The child cannot be admitted to school

   If not

   > The child can be admitted to school

2. Passing a subject:

   If the mark is 35 or more

   > It is a Pass

   If not

   > It is a Fail

3. Buying a book:

   If you have money equal to or more than the price of the book

   > You can buy the book

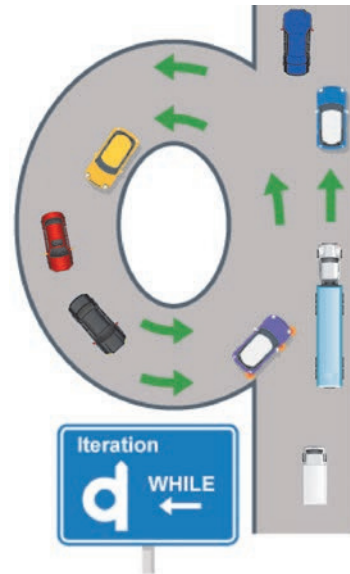   If not

   > You cannot buy the book

Activity

1. Write down three incidents which consist of selection.

2. If a Sri Lankan citizen gets the right to vote after completing the age of 18, select the most suitable word for the blanks given below.

   If the age is ……………… (more/less) than 18

   > The vote ……………… (can/cannot) be casted

   If not,

   > The vote …………… (can/cannot) be casted

---

iii. Repetition

If one or several steps of an algorithm are repeated until a condition is satisfied, it is called repetition.

Examples

1. Let us consider the process of a class teacher marking the attendance register.

     (1) Call the first name on the register

     (2) Mark 1 if the student is present

     (3) Mark 0 if the student is absent

     (4) Call the name of the next student

     (5) Repeat step (2) or (3) and (4) till the last name of the register is called

2. Let us consider the process of reading a paragraph and calculating the number of words you read.

     (1) Read the first word of the paragraph

     (2) Number of words = 1

     (3) Read the next word

     (4) Add 1 to the number of words

     (5) Repeat step (3) and (4) till the end of the paragraph

     (6) After reading the paragraph, indicate the number of words

Activity

1. Write down as steps, two scenario that comprise repetition.
2. Fill in the blanks below related to repetition that output 5 times from 5 to 60.

    I. n = 5

    II. Output the value of n.

    III. Add 5 to the value of n.

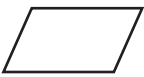   IV. Repeat step number ……… and ……….. till the value of n = 60.

1.3 Representation of an algorithm

Flow charts and pseudo codes are used as tools to present an algorithm to make the algorithm understand better.

1.3.1 Flowcharts

Flowcharts are used to present how the algorithm is built step by step in a dramatic manner. The symbols given in table are used to indicate different functions. (See Table 1.1)

Table 1.1

Symbol	Function
	Start or end
	Input or output
	Process
	Decision
	Flow direction
	Connector

Sequence

In sequence the steps from the beginning to the end are executed in order.

E.g. 1 - Finding the area of a circle.

```
        Start
          ↓
Input the radius of the circle
          ↓
Calculate the area of the circle
          ↓
Output the area of the circle
          ↓
         End
```

Selection

The selection indicates the flow of direction depending on a condition being satisfied or not.

E.g. - Finding whether a number is odd or even

```
                    ┌──────────┐
                    │  Start   │
                    └──────────┘
                         │
                    ╱──────────────╲
                   ╱ Input the number╲
                   ╲                  ╱
                    ╲────────────────╱
                         │
              ┌────────────────────────┐
              │ Find the remainder when the│
              │   number is divided by 2   │
              └────────────────────────┘
                         │
                    ╱────────────╲         ╱──────────────╲
                   ╱ Is the remainder 0?╲─No─ An odd number  ╱
                   ╲                     ╱    ╲──────────────╱
                    ╲───────────────────╱              │
                         │ Yes                         │
                   ╱────────────────╲                  │
                  ╱ An even number   ╱                 │
                  ╲──────────────────╱                 │
                                        ┌──────────┐
                                        │   End    │
                                        └──────────┘
```
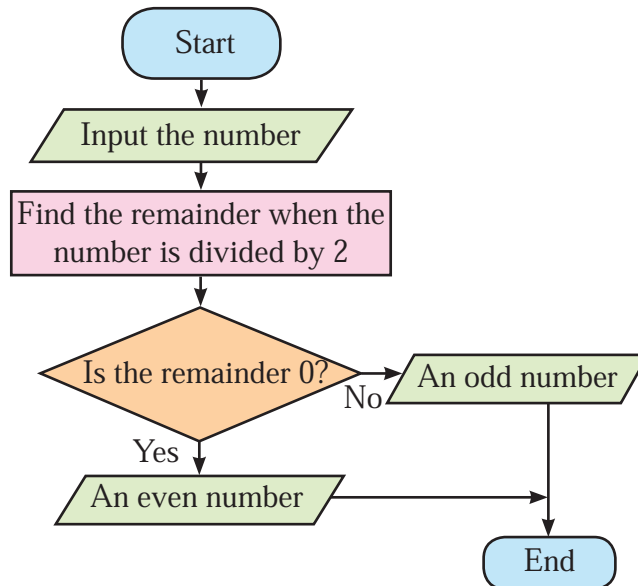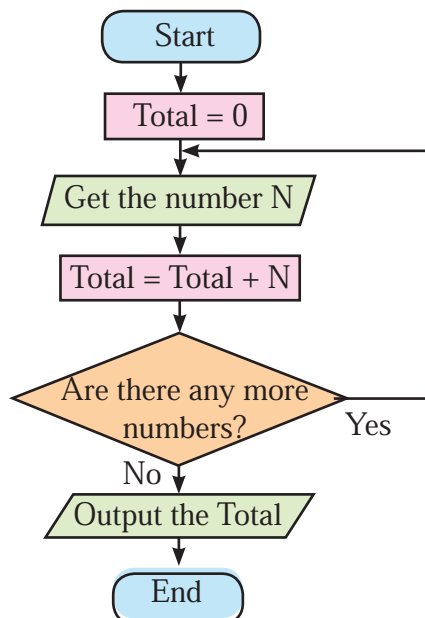
Repetition

Steps are repeated till a condition is satisfied or are continued till it is satisfied.

E.g. - Finding the total of some numbers

```
               ┌──────────┐
               │  Start   │
               └──────────┘
                    │
              ┌──────────┐
              │ Total = 0 │
              └──────────┘
                    │
           ╱───────────────╲
          ╱ Get the number N ╱
          ╲──────────────────╱
                    │
          ┌──────────────────┐
          │ Total = Total + N │
          └──────────────────┘
                    │
           ╱────────────────╲
          ╱ Are there any more ╲──Yes
          ╲    numbers?        ╱
           ╲──────────────────╱
                    │ No
          ╱──────────────────╲
         ╱ Output the Total    ╱
         ╲────────────────────╱
                    │
              ┌──────────┐
              │   End    │
              └──────────┘
```

Draw flow charts to solve the problems given below.

1. Find the perimeter and area of a rectangle.

2. It is decided to add Rs.5000 to the basic salary of the employees of a company. Calculate the new salary.

3. When posting a letter, postal fare is charged according to its weight. The standard fare should be paid for letters which are equal to or less than the standard weight. An additional fare should be charged if the weight is more than the standard weight.

4. Indicate the first 12 multiples of 7.

5. Draw flow charts for the examples in selection 1:2:1.

1.3.2 Pseudo codes

When an algorithm is presented in simple English terms it is called a pseudo code. Pseudo codes are independent of a computer language. Pseudo codes can be converted to any programming language instructions. Hence, pseudo codes make computer programming easier.

Let us see simple English terms used in an pseudo code.

BEGIN - To indicate a beginning

END - To indicate an end

INPUT , READ , GET - To indicate an input

OUTPUT, DISPLAY , SHOW - To show an output

PROCESS, CALCULATE - To indicate a process

IF … THEN .. .ELSE … ENDIF - Used to indicate a selection

FOR – DO

WHILE – ENDWHILE } Used to indicate a repetition

REPEAT - UNTIL

Writing pseudo codes

E.g. 1 - Finding the area of a circle

 BEGIN

        INPUT Radius

        CALCULATE Area $= 22/7 \times$ Radius $\times$ Radius

        DISPLAY Area

    END.

E.g. 2 - Finding whether a number is odd or even

```
BEGIN
        READ number as N
        CALCULATE  Remainder after number is divided by 2
        IF Remainder = 0 THEN
                DISPLAY "Even number"
        ELSE
                DISPLAY "Odd number"
        ENDIF
END.
```
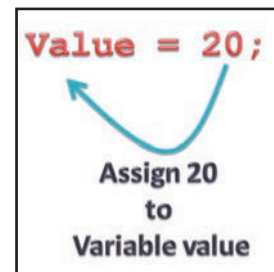
E.g. 3 - Finding the total of some numbers

```
BEGIN
        Total = 0
        REPEAT
                READ Number as N
                CALCULATE Total = Total + N
        UNTIL numbers are over
        DISPLAY Total
END.
```
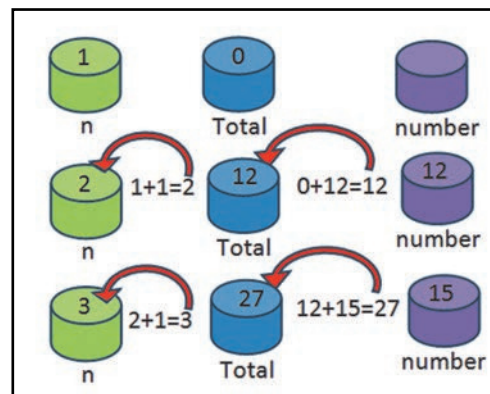
E.g. 4 - Finding the total and the average of 10 numbers

Value = 20;

Assign 20
to
Variable value

```
BEGIN
        Total = 0
        Average = 0
        n = 1
        WHILE n <= 10
                READ Number
                CALCULATE Total = Total + Number
                n = n + 1
        ENDWHILE
        CALCULATE Average = Total/(n - 1)
        DISPLAY Total, Average
END.
```

Following are some of the facts about the above pseudo code.

- Total, Average and Number are variables.
  - ° When values are assigned to number variable, the value of Total, Average and n variables change.
- n indicates the number of repetitions. (Number of times the loop is executed)
- The statements Total = 0 and Average = 0, makes starting values of these variables assigned as 0.
  - ° Hence, the initial value of Total and Average are 0s.
- The statement n = 1, makes the starting value of n is assigned to 1.
- n <= 10 is the condition that should be satisfied.
- WHILE n <= 10 indicates that the loop n should be repeated until value of n is 10.
  - ° Repetition occurs when the value of n is 10 or less than 10. This means, till the condition n <=10 is true, repetition occurs. When the value of n becomes 11, the repetition stops. Then the condition becomes false.
- READ denotes getting a value for Number variable.
- Total = Total + Number denotes the present value of Total is added to number and the resultant value is assigned on the new value of Total.

- n = n + 1 calculates the number of repetitions. 1 is added to the present value of n and the result is assigned to n.
- ENDWHILE indicates the limit to end repetition. Hence, only READ number, Total = Total + Number and n = n + 1 are repeated till the condition n <= 10 is fulfilled.
- When the repetition stops, the value of n is 11 and the condition is false.
- By Average = Total/(n - 1) , final value of Total is divided by (n − 1) and that value is assigned to Average variable.
- DISPLAY (Total, Average) produces the output of the total of 10 numbers and its average.
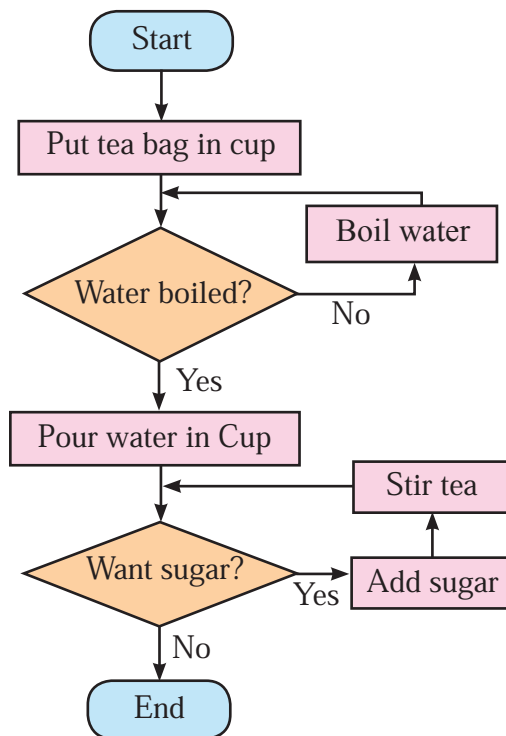
- When a value is assigned to a variable, the previous value is lost.
- When the statement Total = Total + Number is executed, the value assigned to the Number variable is added with the value assigned to Total variable and the result obtained is assigned to the Total variable.
- Total = Total + Number is not a mathematical formula.

1.3.3 Converting flow charts to pseudo codes

An algorithm can be presented in a flow chart as well as in a pseudo code. Hence, let us examine how a flow chart can be converted to a pseudo code.

BEGIN

    Put tea bag in cup

    WHILE (not water boiled)

        Boil water

    ENDWHILE

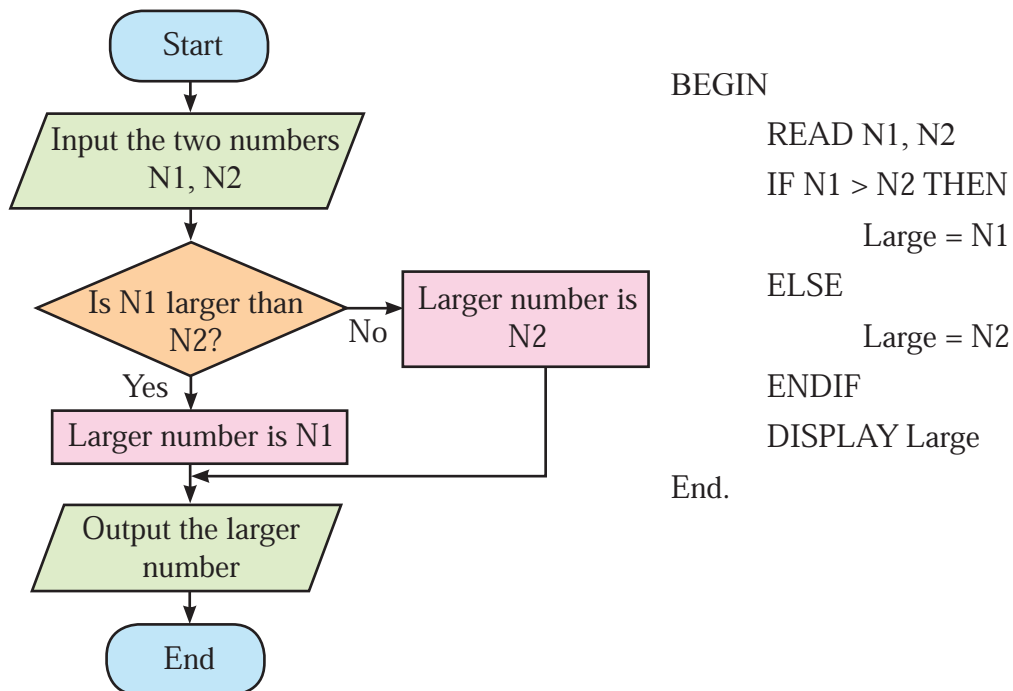   Pour water in cup

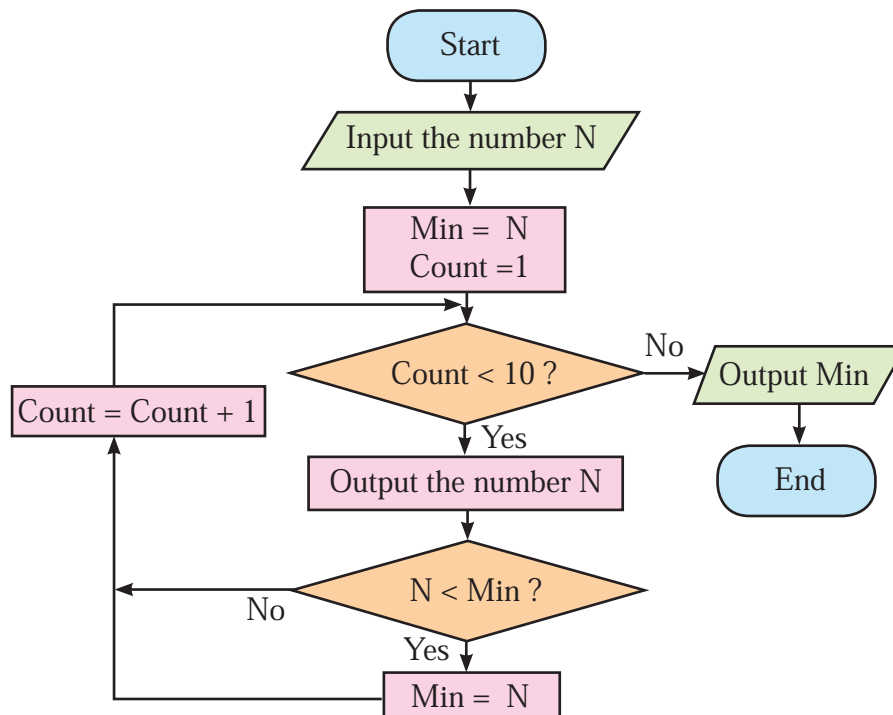   WHILE (sugar needed)

        Add sugar

        Stir tea

    ENDWHILE

END

E.g. 1 - Finding the larger number from two different numbers


```
BEGIN
        READ N1, N2
        IF N1 > N2 THEN
                Large = N1
        ELSE
                Large = N2
        ENDIF
        DISPLAY Large
End.
```

E.g. 2 - Finding the smallest number from 10 numbers

```
BEGIN
        INPUT Number as N
        Min = N
        Count = 1
        WHILE Count < 10
                OUTPUT Number as N
                IF N < Min  Then
                        Min = N
                ENDIF
                Count = Count + 1
        ENDWHILE
        PRINT  Min
END.
```
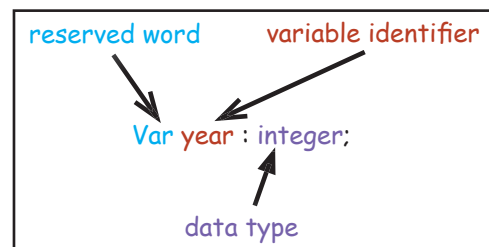
1.4 Pascal programming

1.4.1 Identifiers

An identifier is a term used to represent a variable, constant or a program. The following are some rules that should be followed in declaring an identifier.

- Reserved words in Pascal cannot be used as an identifier name. In any programming language reserved words cannot be used as identifiers.

  E.g. - BEGIN, END are not valid (**E.g.** - A-Z, a-z)

reserved word   variable identifier

Var year : integer;

data type

- Should start with an English letter.
- After the first letter of the identifier, letters (a-z, A-Z) or numbers (0-9) and underscore (_) can be used.

  E.g. - Student_name

- Not case sensitive (**E.g.** - Art, art, ART will be the same identifier)
- There should not be any space between words.

  E.g. - Student Name - Not valid

- The special characters such as the following should not be included in an identifier.

    ~ ! @ # $ % ^ & * ( ) - + = { } [ ] : ; ' " <> ? , . / | \

    but, only underscore (_) is valid.

- Use of meaningful terms for identifiers can make program easily understood.

    Examples of valid identifiers

    Sum, SUM, Total_Nos, Num1, FirstName, Last_Name

    Examples of identifiers that are not valid

    $75, Average Marks, 9A, Last-name

1.4.2 Reserved words

The reserved words in Pascal, are defined in Pascal language. Hence, reserved words are not used as identifiers.

Reserved words are different from language to language. The following are reserved words used in Pascal.

and	exports	mod	shr
asm	file	nil	string
array	for	not	then
begin	function	object	to
case	goto	of	type
const	if	or	unit
constructor	implementation	packed	until
destructor	in	procedure	uses
div	inherited	program	var
do	inline	record	while
downto	interface	repeat	with
else	label	set	xor
end	library	shl	ate
			to

1.4.3 Standard data types in Pascal

When a program is executed, the input and output should be stored in computer memory. The space needed for each is defined according to the data type. Hence, it is essential for a programmer to have knowledge of data types.

The following are data types and their ranges.

Integer - Plus or minus whole numbers

E.g. - 0, 46, -12

Real - Plus or minus decimal numbers

E.g. - 0.0, 25.68

Boolean

True or False

Char - Any character of the key board

E.g. - 'k', '#', '7'

VARIABLE identifier (NAME)	VALUE	TYPE
number	- 123	integer
sum	456	integer
character	'B'	char
book	'Mathematics'	string

String - Any sequence of characters

E.g. - 'ICT', 'programming', 'Sri Lanka'

Important

Values of Char and String are included inside single quotation ' '.
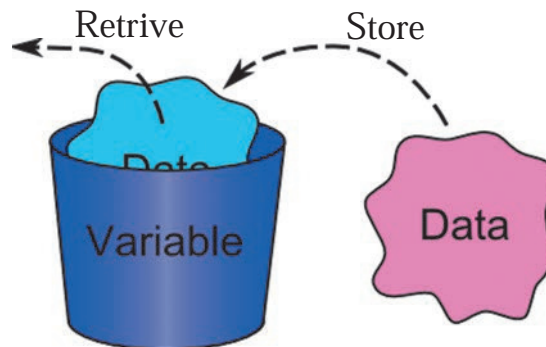
1.4.4 Variables and constants

Variable

A variable is an identifier which changes the values given to it when the program is being executed.

In Pascal, "var" is used to declare variables.

Retrive   Store

E.g. -   var count : integer;
         var a,b : real;
         var   n1, n2 : integer;
                 Avg : real ;
                 Pass : boolean;
                 Character : char;
                 Name,school : String;

Variable   Data

Important

A variable has a <u>name</u>, and it stores a <u>value</u> of the <u>declared</u> type.

Constants

The identifiers which do not change their values while the program is executed are called constants. In Pascal, "const" is used to declare a constant.

Example

Const   max = 100;
Const   pi = 22/7;

Observation

- When executing a program, a variable can take different values. However, the value of a constant remains unchanged.

1.5 Operators

Operators are required to perform calculations, comparisons, and to evaluate logical expressions. Hence, operators are essential in programming.

Basic types of operators

1. Algebra operators

Operator	Usage	Example expression	Result
+	Addition	6 + 3	9
-	Subtraction	7 - 5	2
*	Multiplication	2 * 5	10
/	Division	10/4	2.50
DIV	Division of round numbers	20 DIV 6	3
MOD	Remainder after division	20 MOD 6	2

```
        3 ←────── DIV
   6 | 20
     | 18
        2 ←────── MOD
```

2. Comparison operator

Comparison operators are used to compare values or expressions. The final result of an expression which consists of a comparison always takes a Boolean value. Hence expression will be True or False.

Function	Usage	Example expression	Result
>	Greater than	7 > 3	True
>=	Greater than or equal	8 >= 8	True
<	Less than	3 < 2	False
<=	Less than or equal	4 <= 6	True
=	Equal	3 = 1	False
< >	Not equal	2 < > 5	True

3. Logical operators

Logical operators are used to combine two or more expressions. For further study on this refer basic logic gates that you learnt in Grade 10.

i) AND operator

AND operator takes the form "(First Expression) AND (Second Expression)". Depending on the first expression or second expression being True or False, the result becomes True or False. The following table shows the function of AND.

First Expression	Second Expression	(First Expression) AND (Second Expression)
False	False	False
False	True	False
True	False	False
True	True	True

Example

1. (Rain fall > 56) AND (Temperature < 30)
2. (Height > 60) AND (Age < 15)
3. Let us consider, (3 >= 2) AND(3 < > 3)
   3 >= 2 is True. 3 < > 3 is False. Hence, the result of the expression is False.

Important

* If at least one operator out of the two operators is false, the AND evaluates to false.

* When only both operators are true, the expression of AND evaluates to true.

ii) OR operator

OR operator takes the form "(First Expression) OR (Second Expression)". Depending on the first expression and second expression being True or False, the result of OR is True or False. The following table shows the function of the OR operator.

First expression	Second expression	(First expression) OR (Second expression)
False	False	False
False	True	True
True	False	True
True	True	True

Example

1. (Temperature > 30) OR (Rainfall < 55)
2. Let us consider, (3 >= 2) OR (3 <> 3)
   3 >= 2 is True. 3 <> 3 is false. Hence, the result of the expression is true.

Important

* When at least one operator out of the two operators is true, the OR operator evaluates to true.

* When only both operators are false, the expression of OR operator evaluates to false.

iii) NOT operator

A True expression is always evaluated as false with a NOT operator while a false expression is always evaluated as true.

Expression	NOT (Expression)
False	True
True	False

Example

1. NOT (Temperature > 30)
2. NOT(5 = 5) is indicated as false Statement.
   Therefore 5 = 5 is a true expression. Hence, the statement NOT (5 = 5) is indicated as false

Operator precedence

When Pascal expressions are evaluated, the order of precedence is given below.

Priority order	Operator						high
1	NOT						
2	*	/	DIV	MOD	AND		
3	+	-	OR				
4	=	<>	<	<=	>	>=	less

Evaluating expressions

E.g. 1
5 + 14 MOD 4
5 + 2
7

E.g. 2
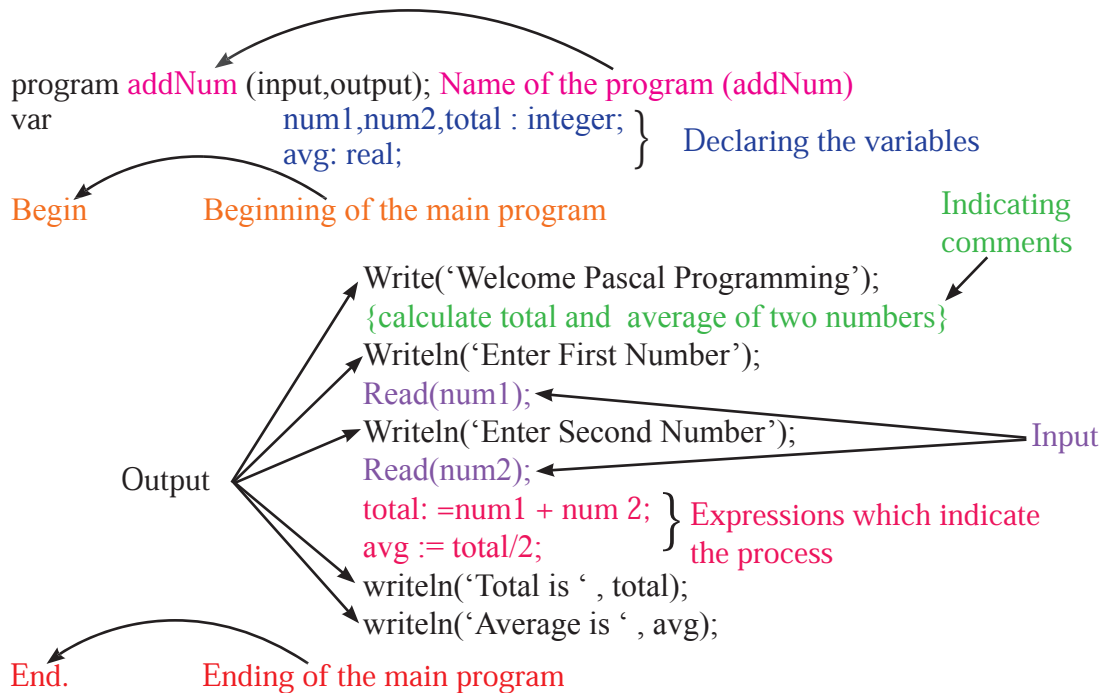3 + 7 DIV 2
3 + 3
6

E.g. 3
16 / 4 * 2
4 * 2
8

E.g. 4
NOT (8 MOD 2 > 5)
NOT (0 > 5)
NOT(False)
true

E.g. 5
4 >= 4 AND NOT(7 > 9)
True AND NOT(False)
True AND True
true

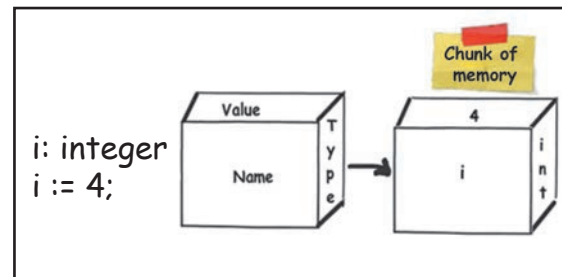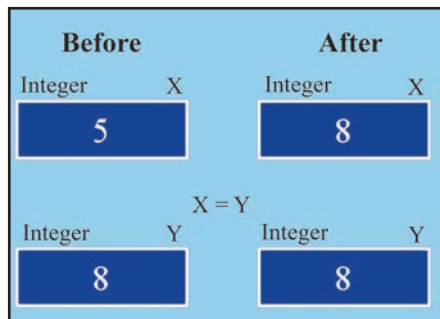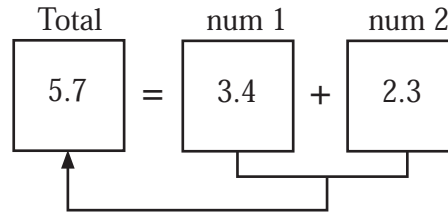Let us identify the basic components of a normal Pascal program.

program addNum (input,output); Name of the program (addNum)
var   num1,num2,total : integer; ⎫ Declaring the variables
                     avg: real;   ⎭

Begin   Beginning of the main program   Indicating
                                                                 comments

                    Write('Welcome Pascal Programming');
                    {calculate total and average of two numbers}
                    Writeln('Enter First Number');
                    Read(num1);
                    Writeln('Enter Second Number');   Input
      Output   Read(num2);
                    total: =num1 + num 2; ⎫ Expressions which indicate
                    avg := total/2;   ⎭ the process
                    writeln('Total is ' , total);
                    writeln('Average is ' , avg);

End.   Ending of the main program

Note : To write comments (*..............*) can also be used.

- "program", "input" and "output" are reserved words.

- "addNum" is an identifier. This is the name of the program. It is not essential to indicate input, output within brackets with the program name.

- read( ) and readln( ) functions words are used for input.

  ➢ Data is input to num1 variable through Read(num1);

  ➢ Data is input from readln() from a new row.

- write( ) and writeln( ) functions are used for output.
  - ➢ From the funtion Write('Welcome Pascal Programming') outputs 'Welcome Pascal Programming'.
  - ➢ The function writeln('Average is ' , avg); prints the value of the Avg variable in a new line.
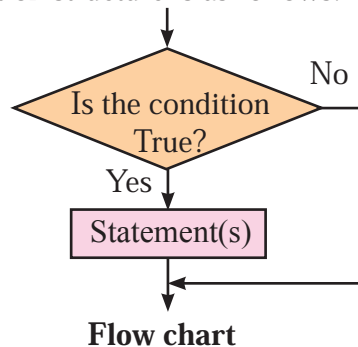
When writing Pascal statements;

- Semi-colon (;) is used at the end of an statement. Semi-colon indicates the end of an statement.

- What happens with the expression total := num1 + num2 is that it adds the variables num1 and num2 and assign result to the variable 'total'

- ":=" is the assignment operator.

Total		num 1		num 2
5.7	=	3.4	+	2.3

Before		After	
Integer	X	Integer	X
5		8	
	X = Y		
Integer	Y	Integer	Y
8		8	

```
i: integer
i := 4;
```

Value
Name
Type
Chunk of memory
4
i
int

1.6 Selection

IF statement

If condition structure is as follows.

Is the condition True?
No
Yes
Statement(s)

Flow chart

IF Condition THEN
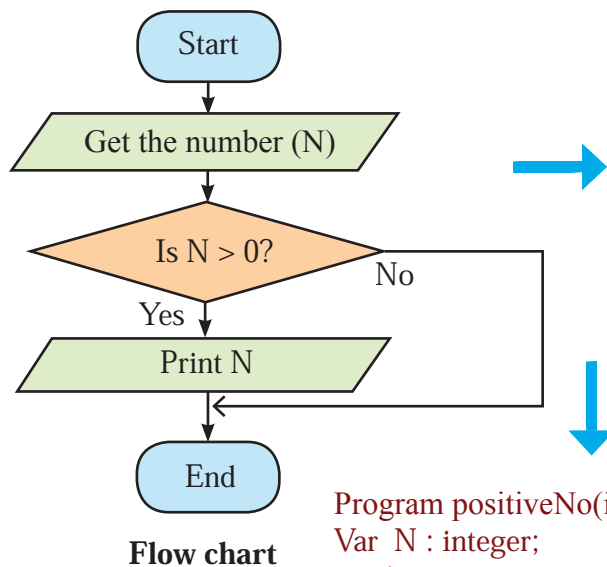   Statement(s)
ENDIF

Pseudo Code

There are two types of IF statements.

i) IF... THEN.... ENDIF

Here, the statement will be executed if only the condition is satisfied.

E.g. 1 - If the input number is only positive, print the number.

Begin
   Input N
    IF N > 0
  THEN
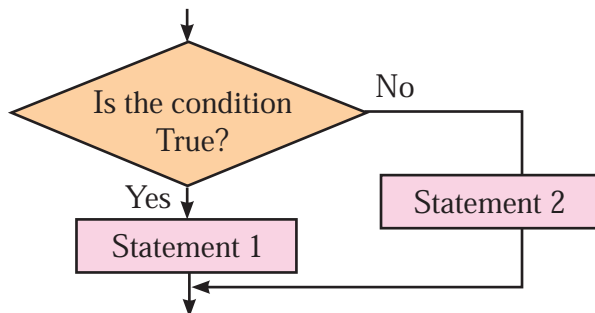   Print N
   ENDIF
End.

Pseudo Code

Flow chart

Program positiveNo(input, output);
Var N : integer;
Begin
   Writeln('Enter Number');
   Read(N);
      If N > 0 then
   Writeln('Positive Number');
End.

Pascal Code

ii) IF... THEN.... ELSE .... ENDIF

If the condition is satisfied Statement 1 is executed if not Statement 2 is executed. The If condition structure is as follows.

IF Condition THEN

     Statement1

ELSE

     Statement2

ENDIF

Pseudo Code

Flow chart

For free distribution

E.g. 2 - Finding the larger number from two unequal numbers.

```
program LargeNo(input,output);
Var  N1,N2,Large: integer;
Begin
        Writeln('Enter Two Numbers');
        Read(N1,N2);
        If N1 > N2 then
                Large := N1
        Else
                Large := N2;
        Writeln('Large Number is ' , Large);
End.
```
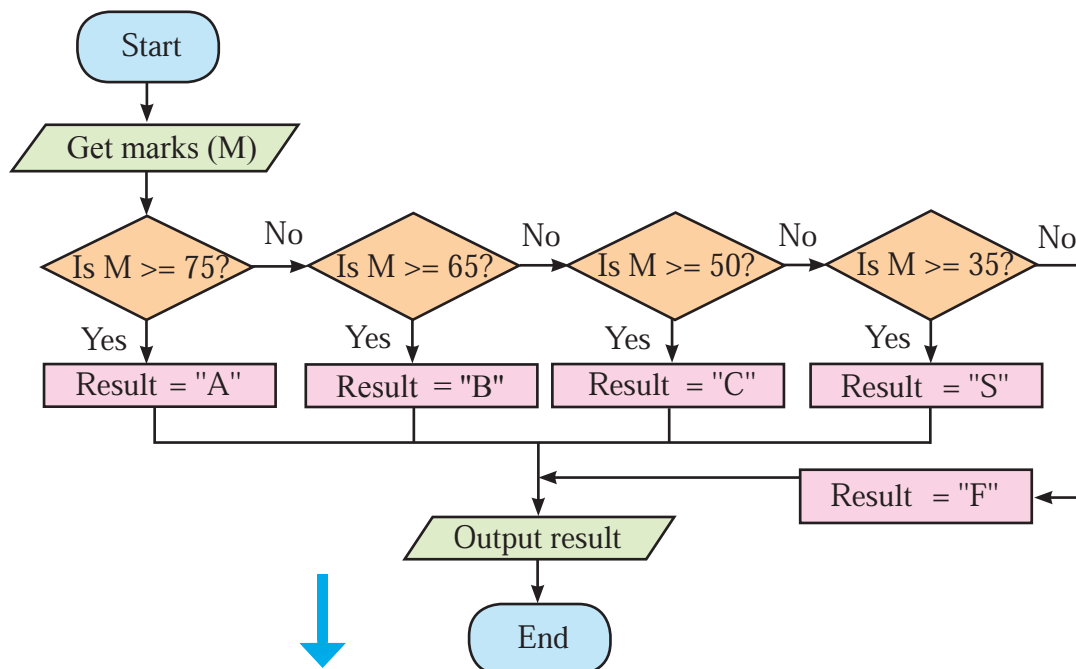
Pascal program

Nested IF

When there are multiple conditions one after the other, Nested IF is used.

i) Use of Nested IF when there are multiple conditions for a single variable

E.g. 3 - Finding the Grade when the marks scored by a student for a subject is given as input.

Flow chart

```
Begin
        Input Marks as M
        IF M >= 75 Then
            Grade = "A"
            ELSE
            IF M >= 65 then
                Grade = "B"
                ELSE
                IF M >= 50 then
                    Grade = "C"
                    ELSE
                    IF M >= 35 then
                        Grade = "S"
                        ELSE
                        Grade = "F"
                    ENDIF
                ENDIF
            ENDIF
        ENDIF
        Display Grade
End.
```

Pseudo Code

```
program GradeForMarks (input,output);
Var    M: integer;
       Grade: char;
Begin
       Writeln('Enter Marks');
       Read(M);
       If M >= 75 then
          Grade := 'A'
       Else
          If M >= 65 then
       Grade := 'B'
          Else
                  If M >= 50 then
              Grade := 'C'
            Else
                If M >= 35 then
                    Grade := 'S'
                Else
                    Grade := 'F';
       Writeln("Grade = ", Grade);
End.
```
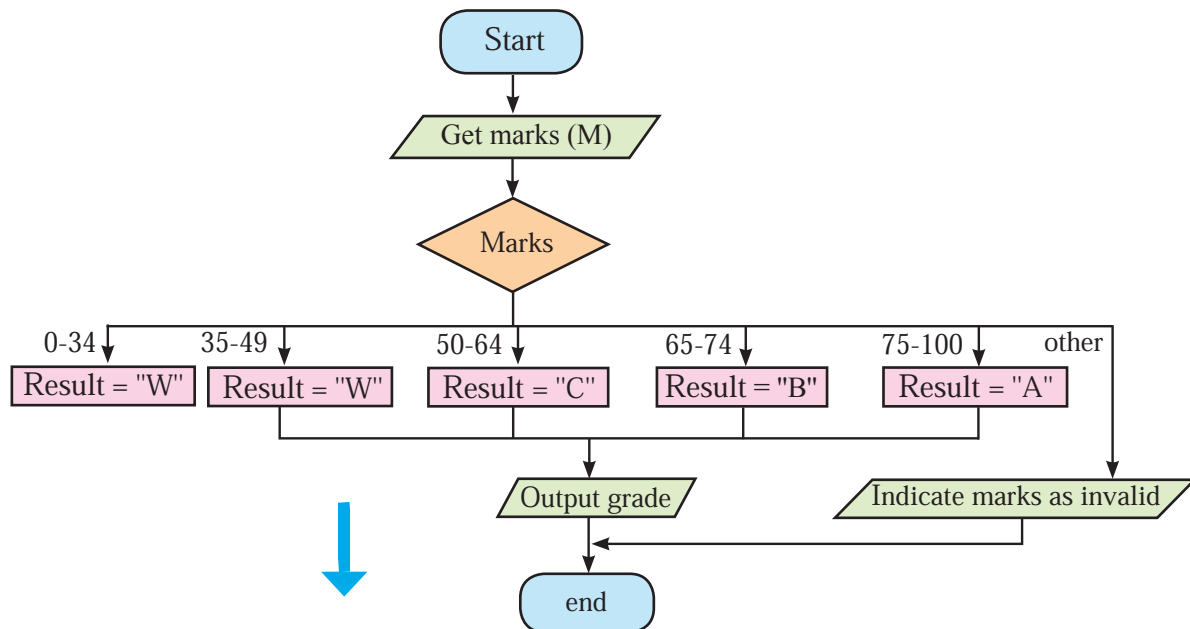
Pascal Code

Using CASE statement when a variable has multiple conditions

Rather than using IF…. THEN… ELSE… ENDIF if conditions repeatedly, it is easier to use CASE statement.

Flow chart

For free distribution

```
program FindGrade(input,output);
var     Marks : integer;
        Grade: char;
Begin
        Writeln('Enter Marks');
        Read(Marks);
        Case Marks of
                0..34 : Grade := 'W';
                35..49 : Grade := 'S';
                50..64 : Grade := 'C';
                65..74 : Grade := 'B';
                75..100 : Grade := 'A';
        Else
                Writeln('Invalid Marks');
        End;
        if (Marks >= 0) AND (Marks <= 100) then
                Writeln('Grade is ' , Grade);
End.
```

Pascal program

1.7 Repetition

Let us examine how repetition structures are used when the number of repetitions is known in advance.

i) FOR – DO structure (1st Method)

FOR Variable := Value_1 TO Value_2 DO

- The data type of Variable , Value_1 and Value_2 should be integer.
- The value of Value_2 should be larger than Value_1 to start repetition.
- Repetition is started with Value_1 and ended with Value_2.
- Hence, FOR – DO structure can be used when the number of repetitions is known in advance.

Repetition structure	Starting value	Ending value	Number of repetitions
FOR X :=1 TO 5 DO	1	5	5
FOR X := 0 TO 4 DO	0	4	5
FOR X := 5 TO 10 DO	5	10	6

E.g. - Printing the values from 1 to 10

```
Program print10Nos(input,output);
Var     count : integer;
Begin
        For  count := 1 to 10 do
        Writeln(count);
End.
```

Here the count variable value is changed from 1 to 10 while printing the output values, and two loop is executed 10 times.

ii) FOR – DO Loop (Method 2)

FOR Variable := Value_1 DOWNTO Value_2 DO

- The value of Value_1 should be smaller than Value_2 to start repetition.
- Repetition is started with Value_1 and ended with Value_2.

Repetition structure	Starting value	Ending value	Number of repetitions
FOR X := 10 DOWNTO 5 DO	10	5	6
FOR X := 4 DOWNTO 0 DO	4	0	5

E.g. - Printing of values from 10 to 1

```
Program print Reverse (input,output);
Var     count : integer;
Begin
        For  count := 10  downto 1 do
        Writeln(count);
End.
```

Count variable value is changed from 10 to 1 while printing the output and and the loop in executed 10 times.

Finding the total and average of ten numbers

```
program total_avg (input,output);
var     I,num,total : integer;
        avg: real;
Begin
        total := 0;
        for I := 1 to  10 do
        begin
                writeln('Enter Number');
                read(num);
                total :=  total+num;
        end;
                avg := total/I;
                writeln('Total is ' , total);
                writeln('Average is ',avg);
        end.
```

A block of statments in repetition.

When the number of repetitions are not known in advance, "While do" or "repeat until" structures are used.

i) WHILE DO Loop

- Conditions are checked at the beginning of the loop.
- Statements inside the loop are executed (ie: a loop that runs for ever) only if the condition is true.
- Statements inside the loop never executed if the condition is false.
- The condition becomes false at the end of the repetition.
- If the condition does not become false while the repetition is executed, it will be an infinite loop.

E.g. 1 - while number > 0 do
        Repetition is executed if only the value of the variable number is positive.

E.g. 2 - number := 1;
        while number <= 10 do
                number := number + 1;

- Condition is true since the starting value of the variable 'number' is 1.
- Hence, the repetition is executed.
- Each time loop is executed, 1 is added to the value of "number".
- Hence, Loop is executed when the value of "number" is 10 or less than 10.
- Loop stops when the value of the "number" variable is 11.

ii) REPEAT UNTIL Structure

- Condition is not checked at the beginning of loop.
- Condition is checked, after the statements are executed once.
- Loop is started if the condition is false only.
- Loop stops when the condition becomes true.
- If the condition does not become true while the loop is executed, loop will be an infinite loop.

E.g. 1 -

```
count = 0;
Repeat
        writeln ('Pascal');
        count := count + 1
Until count > 5;
```

- The starting value of count variable is 0.
- The word Pascal is displayed on the screen.
- 1 is added to the count variable.
- It is checked whether the value of 'count' variable is larger than 5.
- Loop is executed till the value of the 'count' variable becomes 5.
- Loop stops when the value of count is 6.
- When the Loop stops, the word Pascal is displayed six times on the screen.

E.g. 2 -

```
sum := 0;
repeat
        sum := sum + 5;
        writeln(sum);
until sum < 50;
```

- The starting value of sum variable is 0.
- 5 is added to the value of sum.
- The value of sum is 5 and it is displayed on the screen.
- It is checked whether the value of sum variable is less than 50.
- The condition sum < 50 is satisfied (True).
- Hence, Loop stops.

E.g. 3 -

```
sum := 0;
repeat
        sum := sum + 5;
        writeln(sum);
until sum >=  50;
```

- The starting value of sum variable is 0.
- 5 is added to the value of sum.
- The value of sum is displayed on the screen.
- It is checked whether the value of sum variable is greater than or equal to 50.
- Loop is executed till sum >= 50 condition is satisfied (True).
- When the loop stops, the value of sum is 50.
- Loop is executed 10 times.
- Multiples of 5 from 5 to 50 will be given as output.

1.8 Nested control structures

Inside an algorithm, a control structure(s) may be included inside another. For example, a repetition loop may be included inside another repetition. Similarly a selection may be included inside another selection. Hence nested controlling structures should be used in programming.

1.8.1 Repetition inside selection

A repetition may be executed depending on a condition of a selection being satisfied or not.

E.g. - Depending on the user's selection, an ascending or descending number sequence can be produced as output.

```pascal
program orderNos(input,output);
var    num:integer;
       cho:char;
begin
   writeln('Select Assending(A) or Desending(D)');
   read(cho);
   if cho = 'A' then
     begin
        writeln('Asending Order');
        for num := 1 to 6 do
           writeln(num);
     end;
   if cho = 'D' then
     begin
        writeln('Desending Order');
        for num := 6 downto 1 do
           writeln(num);
     end;
end.
```

1.8.2 Selection in repetition

Let us consider how selection takes place while the repetition control structure being executed.

E.g. - Determining whether the numbers input by the user are odd or even and calculating the total number of odd, even numbers separately.

```pascal
program rep_sel(input,output);
var num,rem,count,e_count,o_count:integer;
begin
   for count := 1 to 10 do
     begin
        writeln('Enter Number');
        read(num);
        rem := num mod 2;
        if rem = 0 then
          begin
             writeln('Even number ');
                e_count := e_count+1;
          end
```

```
        else
           begin
              writeln('Odd number ');
              o_count := o_count + 1;
           end;
     end;
  writeln(e_count,'Even Number/s');
  writeln((o_count ,'Odd Number/s');
end.
```

1.9 Arreys

It is essential in programming to use variables to store data items in memory. Further, such variables have suitable data types. Various variables which are different in names are needed to store data items which belong to the same data type.

E.g. - For instance, five variables are needed to save five round numbers in the memory. Before using such variables, they should be declared as given below.
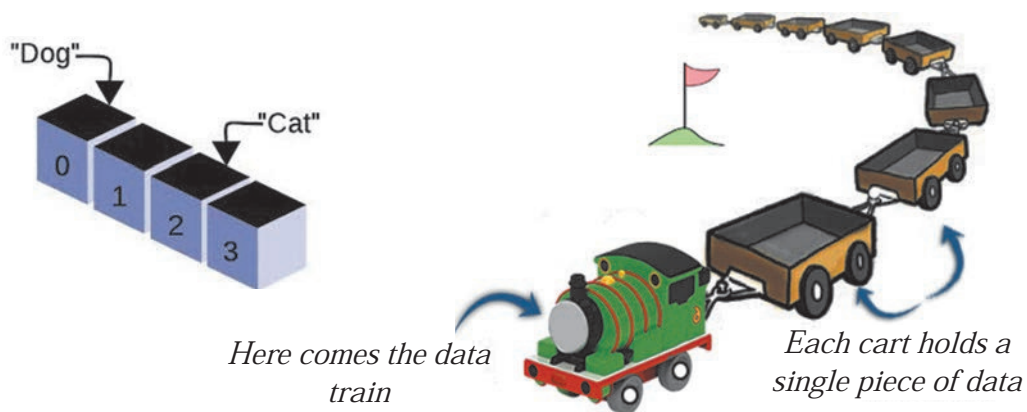
Var   p, q, r, s, t : integer;
       n1, n2, n3, n4, n5 : real;

1.9.1 Use of arrays

An array is used to save data items of the same type in memory using a single variable identifier name. Hence, use of array enables to store data as required under a single variable identifier name without providing different variable names to each item.

Here comes the data train
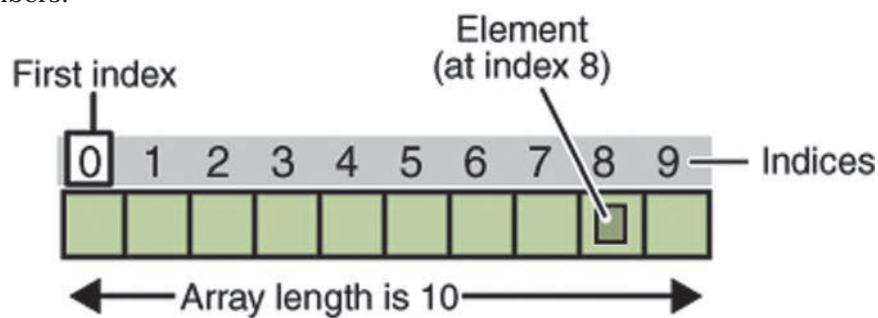
Each cart holds a single piece of data

1.9.2 Defining a one-dimensional array

This is a data structure used to save data of the same type sequentially. An array uses a group of adjoining memory spaces. A one dimensional array can be as follows.

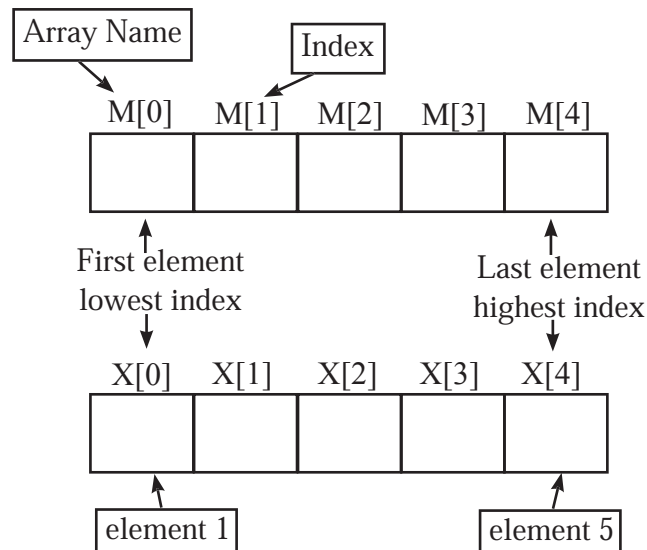Var *Name_of_Array* : array [*first index .. last index*] of *data type*

E.g. - var marks : array [0..9] of integer;

- From this, the array named marks is defined which can include 10 whole numbers.

1.9.3 Attributes of an array

- Elements of an array are positioned next to each other. (re adjacently)
- The index of an array (sequential number) is indicated with the array name in square brackets.

E.g. - the size of the array Var M : Array[0..4] of integer; is 5.

From M[0] to M[4], it consists of 5 elements.

Index is indicated in square brackets.

Depending on the way array is defined, index positions get changed.

E.g. - ; Var X : Array[1..5] of integer;

Only data items which belong to the same type can be stored in the array.

Any element of the array can be accessed randomly. Hence, an array can be accessed easily through a repetition control structure.

E.g. - Entering Maths marks of 40 students into an array

```
var     maths : array[0..39] of integer;
        i,marks : integer;
for i := 0 to 39 do
        begin
                writeln('Enter marks');
                read(marks);
                maths[i] := marks;
        end;
```

1.9.4 Assigning values to an array

My_ pets [0] : = 'Dog'
My_ pets [1] : = 'Cat'

Give the array a name   *Assign a value to it*

Let us consider integer array with five elements which can input whole numbers.

var   num : array[0..4] of integer;

num[0] := 45;

num[2] := 36,num[4] := 60;

num[1] := num[4] + 15;
num[3] := num[0] + num[2]

num[0]	num[1]	num[2]	num[3]	num[4]
45		36		60
45	75	36	81	60

1.9.5 Declaring values of an array

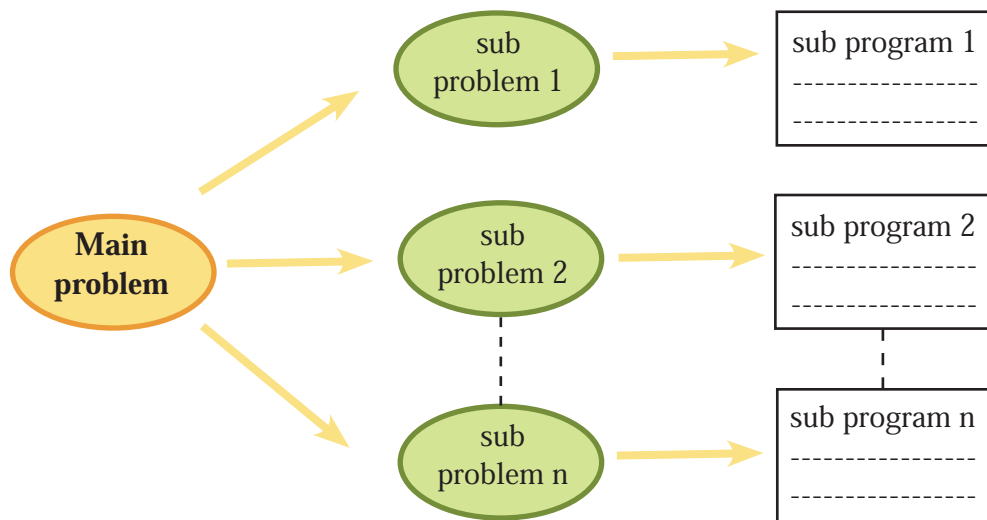The elements declare the values of an array.

writeln (num[3]);	←	Print the 4th element (81)
writeln (num[1], num[4]);	←	Declaring 2nd and 5th elements (36, 60)
for x := 0 to 3 do	←	Print the first 4 elements of the array (45, 75, 36, 81)
writeln (num[x]);		
for x := 2 to 4 do	←	Print the 3 elements - 3rd, 4th, 5th - of the array (36, 81, 60)
writeln (num[x]);		
for x := 0 to 4 do	←	Print all the elements of the array (45, 75, 36, 81, 60)

E.g. - Entering Information and Communication Technology marks of 35 students, determining the highest mark and calculating the average.

```
program ictMarks(input,output);
var marks : array[0..34] of integer;
   i,tot,max : integer;
   avg : real;
begin
   for i := 0 to 34 do
      begin
         writeln('Enter Marks');
         read(marks[i]);(* Read Marks to array *)
         tot := tot + marks[i];(* Add marks *)
      end;
   avg := tot/35;
   max := marks[0];
   for i := 1 to 34 do
      if marks[i] > max then max := marks[i];
   writeln('Maximum marks = ', max);
   writeln('Average marks = ',avg);
end.
```
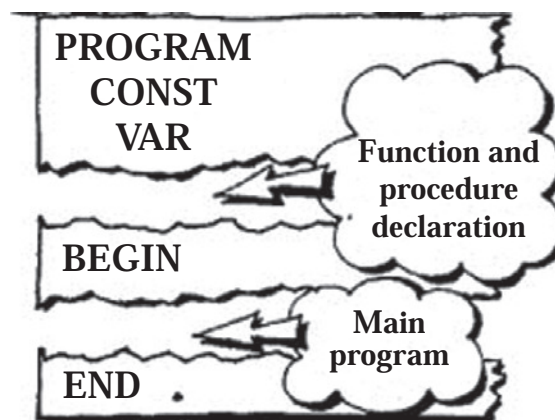
1.10 Use of sub-programs

As a program becomes complex, when the number of sub processes increase, it will be difficult to read and understand and also to maintain. Therefore, using sub programs as much as possible while writing is useful.

1.10.1 Types of sub program

There are two types of sub programs in addition to the main program. A sub program which returns an output back to the main program and a sub program which does not return an output back to the main program. A sub program which returns an output back is called a **Function** and a sub program which does not give an output is called a **Procedure.**

1.10.2 Introducing sub programs

Before starting the main program, functions and procedures should be declared. Sub programs can be called for in the main program. (Calling a function or a Procedure).

The following is the syntax to define a procedure.

Procedure Name_of_Procedure(name_of_variable : data type);

E.g. -   Procedure to find the area of a circle
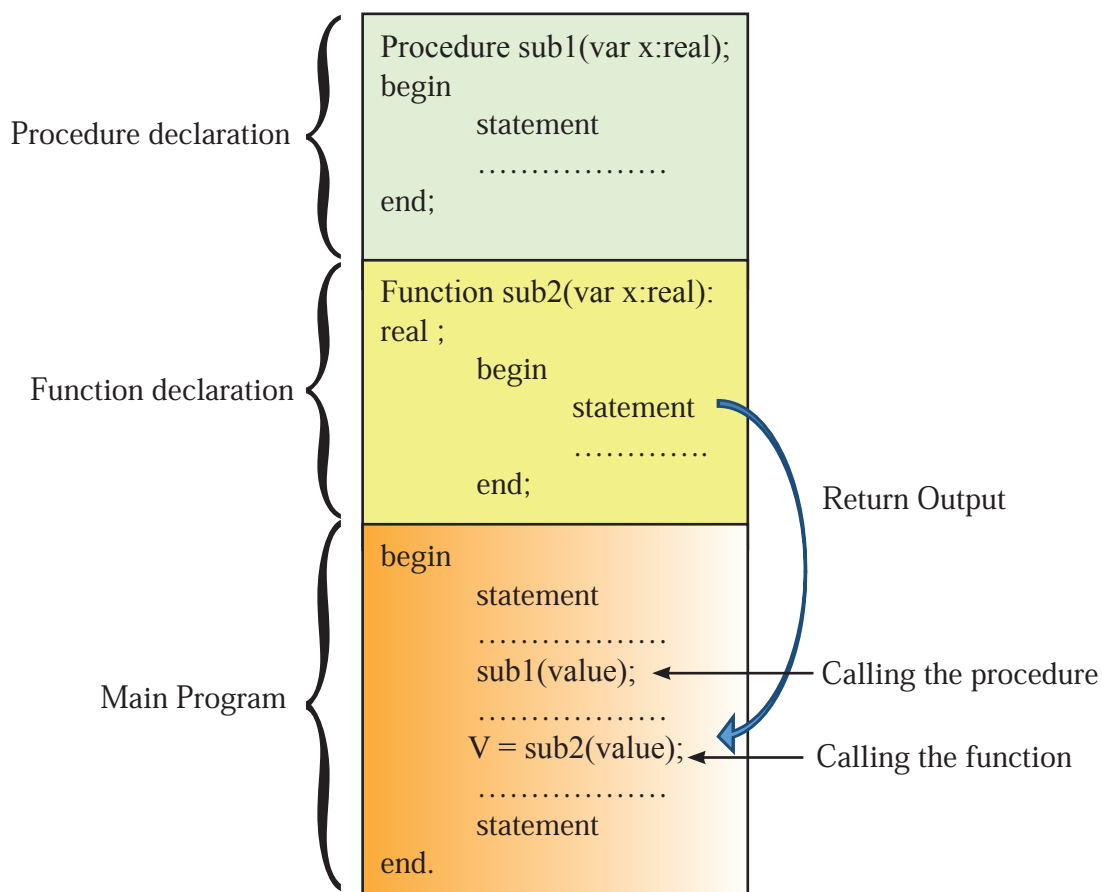
Procedure calculateArea(var radius:real);

Function Name_of_Function(name_of_variable : data type) : data type ;

E.g. -   Function to find the area of a circle

Function calculateArea(var radius : real): real ;

The following syntax define a function.

```
Procedure sub1(var x:real);
begin
        statement
        ................
end;

Function sub2(var x:real):
real ;
        begin
                statement
                ............
        end;

begin
        statement
        ................
        sub1(value);
        ................
        V = sub2(value);
        ................
        statement
end.
```

Procedure declaration

Function declaration

Main Program

Return Output

Calling the procedure

Calling the function

E.g. - Let us consider the program to calculate the area and circumference of a circle.

1. A program built with procedures

```
program procedure_circle(input,output);
const pie = 22/7;
var radius:real;

procedure getData(var radius: real);
      begin
          writeln('Enter Radius');
          read(radius);
      end;
procedure processArea(var radius:real);
      var area:real;
      begin
          area := pie * radius * radius;
          writeln('Area = ',area);
      end;
procedure processCircumferance(var radius:real);
      var circum:real;
      begin
          circum := 2 * pie * radius;
          writeln('Circumferance = ',circum);
      end;
begin
   getData(radius);
   processCircumferance(radius);
   processArea(radius);
end.
```

2. A program with functions

```
program function_circle(input,output);
const pi = 22/7;
var radius:real;

function processArea(var radius:real):real;
      var area:real;
      begin
          area := pi * radius * radius;
```
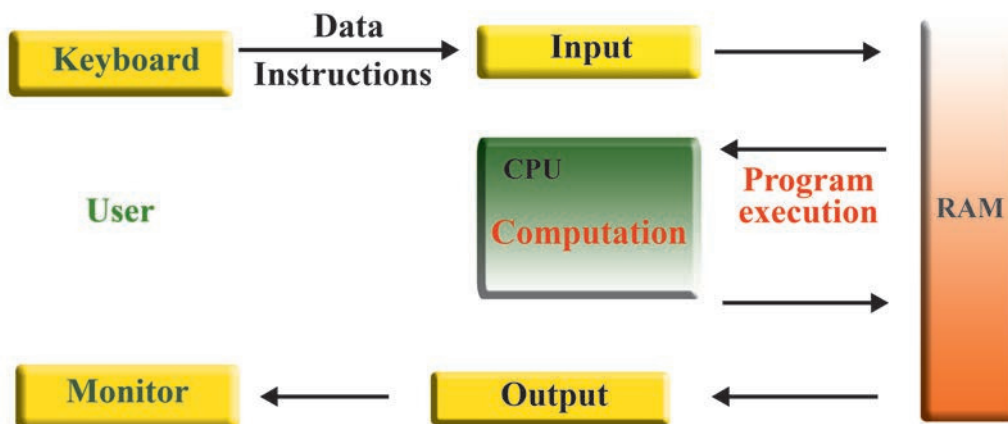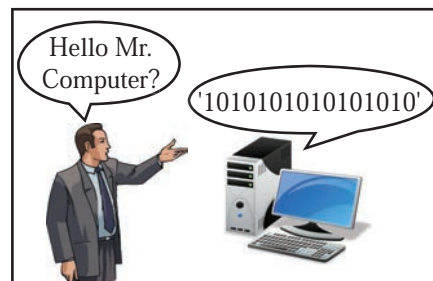
```
        processArea := area;
      end;
function processCircumference(var radius:real):real;
      var circum:real;
      begin
        circum := 2 * pi * radius;
        processCircumference := circum;
      end;
begin
   writeln('Enter Radius');
        read(radius);
   writeln('Circumference = ',processCircumference(radius));
   writeln('Area = ', processArea(radius));
end.
```

1.11 Evolution of programming languages

1.11.1 Need of a programming language

A program is a sequence of instructions which performs a certain task using the computer. A computer language is needed to provide the instructions.

1.11.2 Low level programming languages

Machine language

This is a language which can be directly understood in the computer. Binary numbers such as 0s and 1s (Bits) are used to provide instructions. Hence, the processor could directly run a program written in machine language.

A program written in machine language has the following features;

- Executed directly on the machine
- Fast in operation
- No need of language translating programs
- Dependency on machines (a program written to one computer may not run on another computer)
- Difficult to understand by humans as it is written using 0 and 1.

Assembly language

Instead of commands written in machine language using 0 and 1, assembly language is designed using simple symbols.

A program written in assembly language has the following features;

- Operation is comparatively slower than machine language.
- Assembly language should be translated to instructions using the language translating program called assembler.
- Dependency on machines (a program written for one computer cannot be run on another computer.)
- The use of symbols makes it more simple to understand.

A program written in Assembly language → Assembler → Machine language instructions
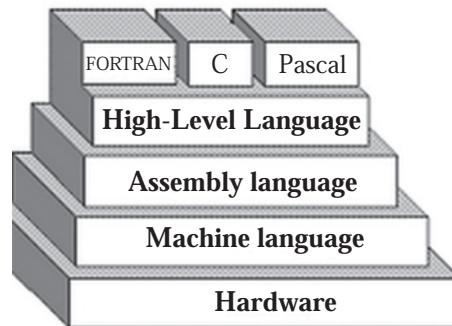
1.11.3 High-level programming languages

Languages which are designed with simple English words enabling the programmer to understand it easily are called high-level computer languages.

Examples for high level computer languages

FORTRAN, BASIC, COBOL, PASCAL, C

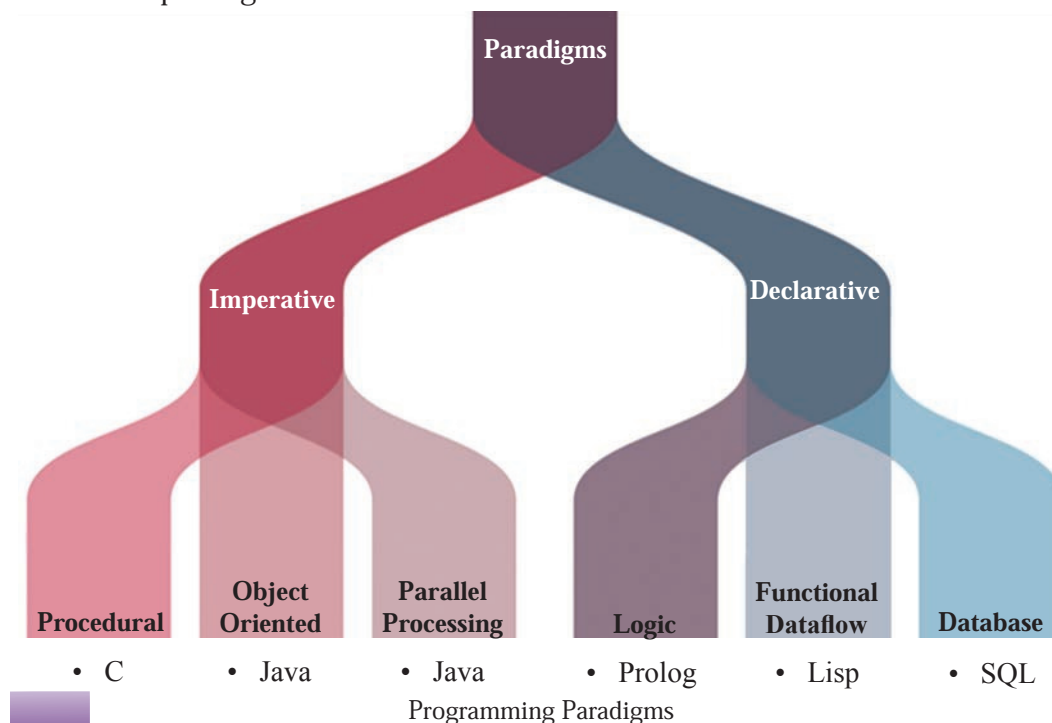A program written in a high-level language has the following features;

- Easy to understand.
- High level languages should be translated to instructions before executing on a computer.
- Do not depend on the machine.

1.11.4 Programming language types

Programming is a creative task where a computer programmer to provide instructions to the computer on how to perform a particular task done. A program can be defined as a set of instructions which instruct the computer which task should be carried out to find a solution to a certain problem.

There are many different approaches to computer programming. These are called programming paradigms. Different approaches develop solutions to problems using programs using different paradigms. Even though most of the programming languages come under one paradigm type, certain languages show elements related to different paradigms.
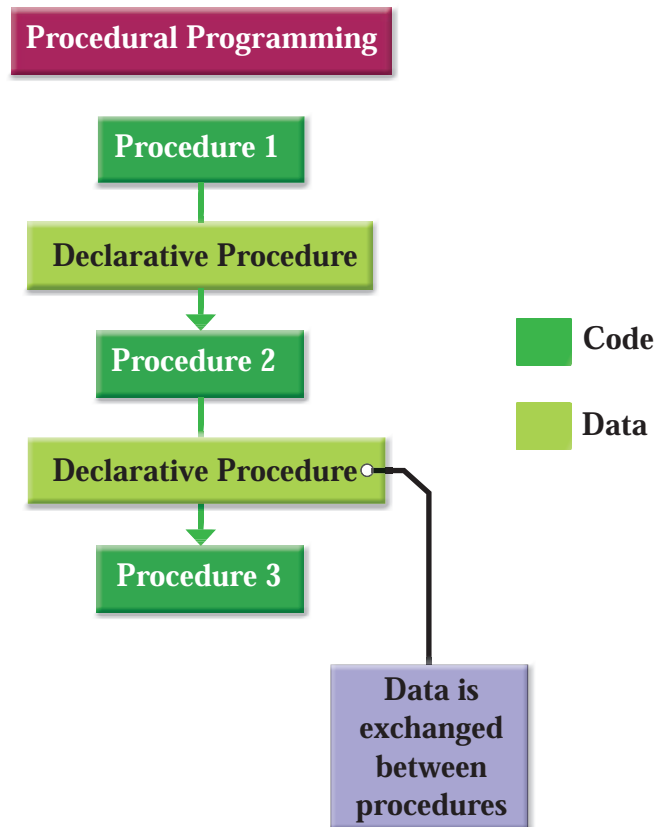

Programming Paradigms

Imperative/ Algorithmic	Declarative		Object- Oriented
	Functional Programming	Logic Programming	
Algol Cobol PL/1 Ada C Modula - 3	Lisp Haskell ML Miranda APL	Prolog	Smalltalk Simula C++ Java

There are a number of programming. The differences among different programming language types.

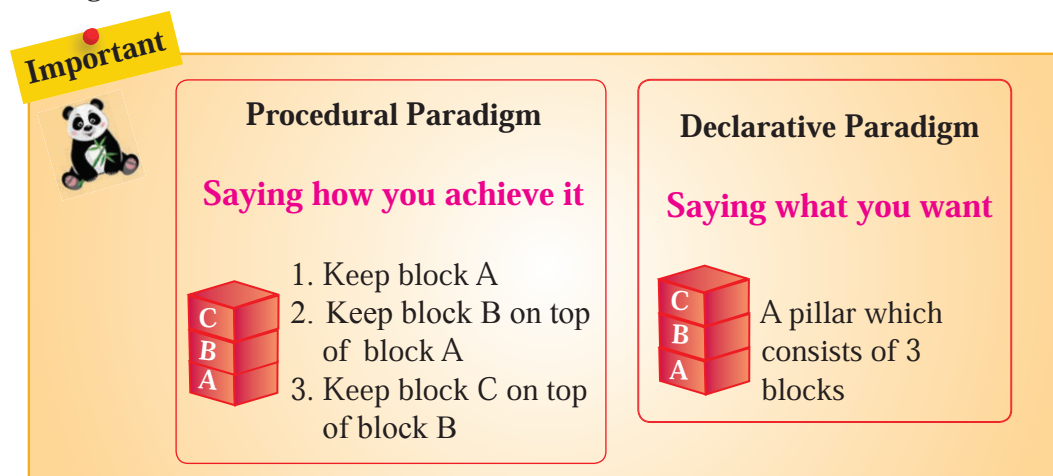## Difference between procedural and declarative paradigms

A procedural language is a computer programming language which consists of a well structured set of steps and procedures. This includes statements for problem solving steps.

Example -

Procedural Programming

Procedure 1

Declarative Procedure

Procedure 2

Declarative Procedure

Procedure 3

■ **Code**

■ **Data**

Data is exchanged between procedures

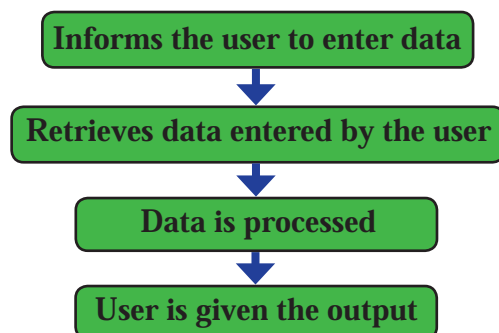The Pascal programming has procedural paradigm features.

A declarative paradigm develops a structure and elements of computer program by indicating calculations and/or logic without a control flow. This helps to reduce or eliminate side effects. In declarative programming, program is designed to solve problems explaining what you want rather than stating how to solve the problem as in primary programming languages. The program itself does not explain how it is executed. This means the computer is provided only what the problem and the required solutions are, not how to solve it. The computer finds solutions related to the given problem. This is completely different from procedural paradigms which execute algorithm as explanatory steps. Declarative paradigms related to Artificial Intelligence.

Important

Procedural Paradigm	Declarative Paradigm
Saying how you achieve it	**Saying what you want**
1. Keep block A 2. Keep block B on top of block A 3. Keep block C on top of block B	A pillar which consists of 3 blocks

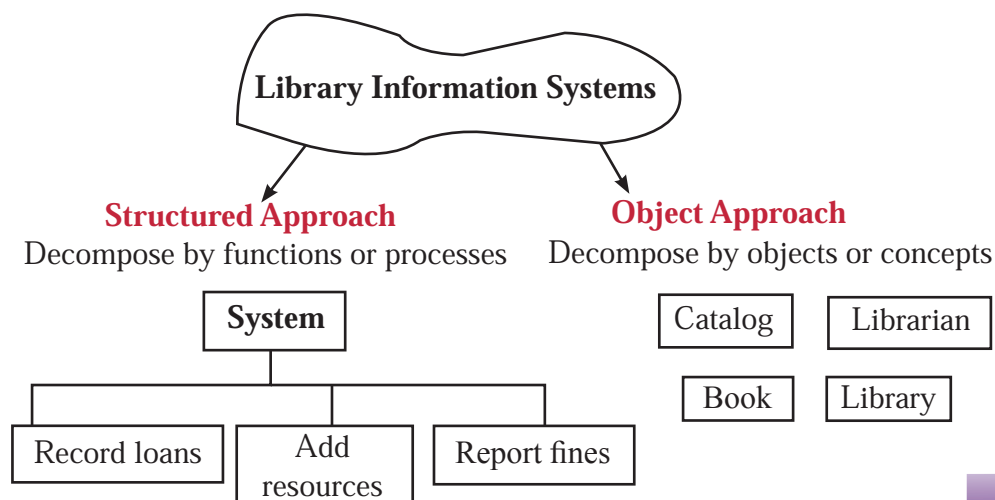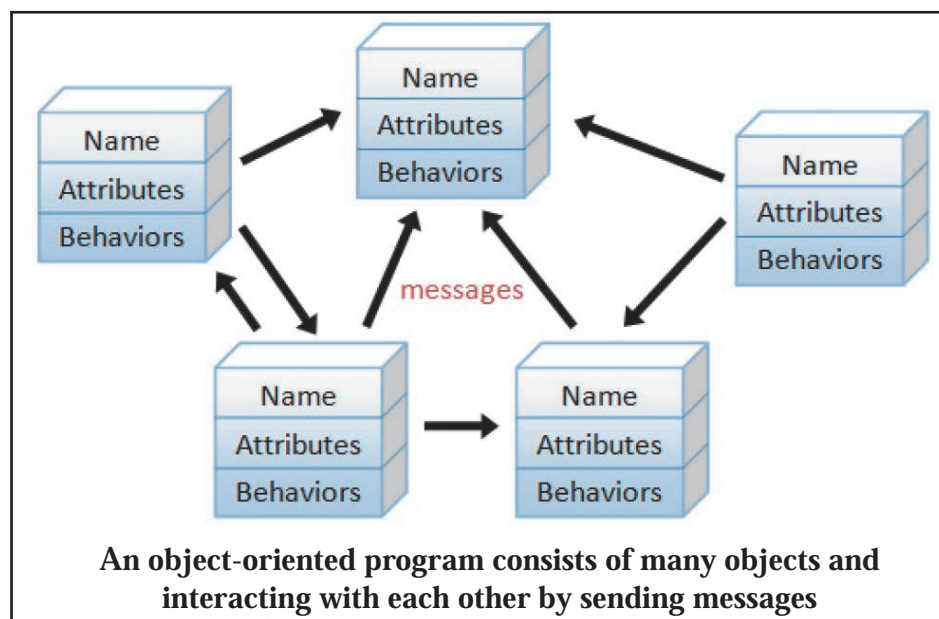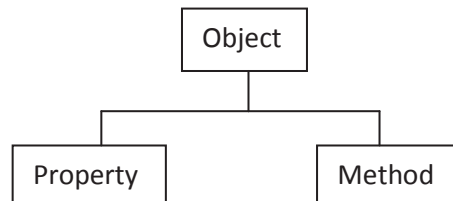**Comparison of structured and object oriented paradigms**

A structured program is a logic based paradigm and it is the pre discussed object oriented program. A structured programming paradigm provides facilities to understand and modify the program. The system is divided into sub systems and there is a top – down flow in it.
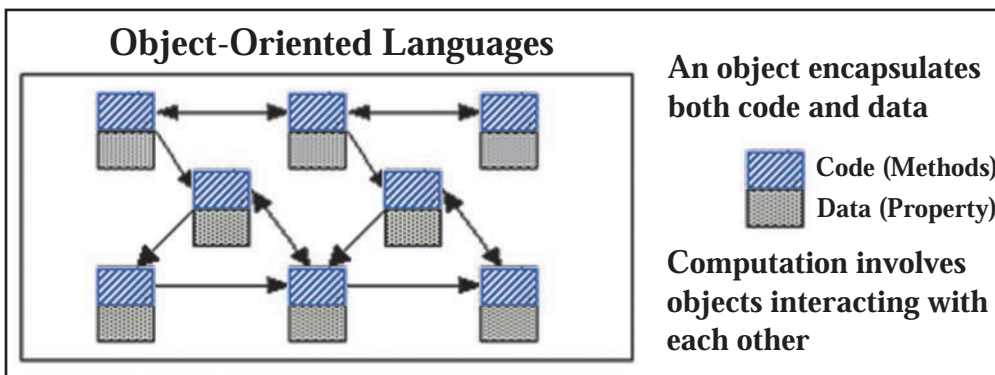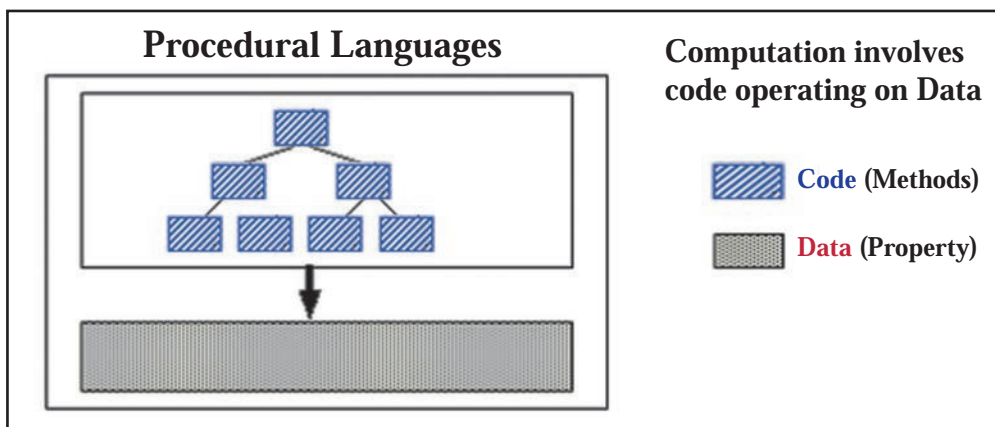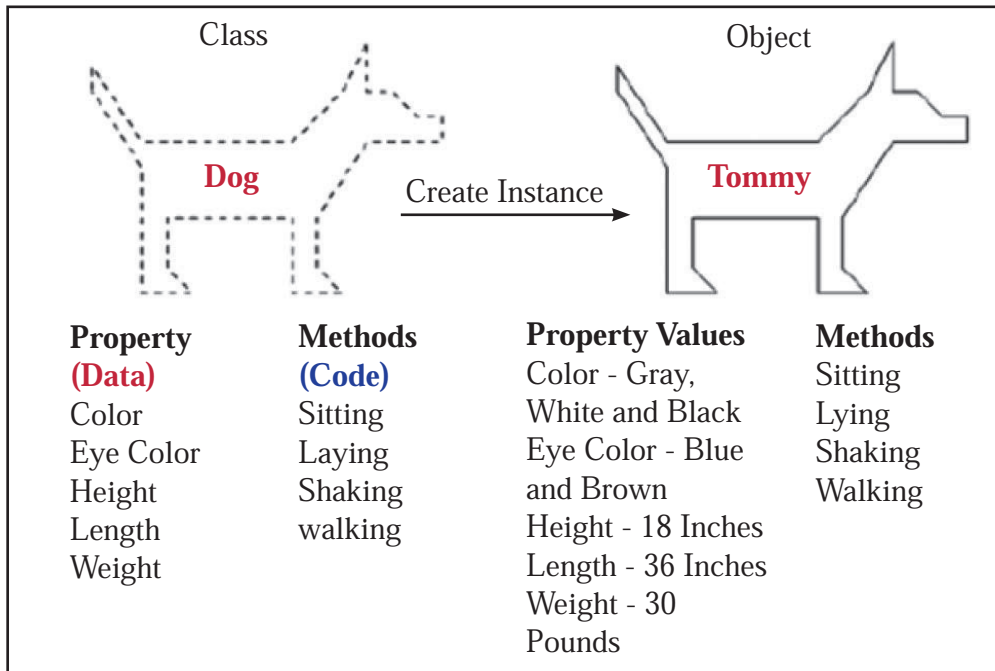
Structured Program

Informs the user to enter data

↓

Retrieves data entered by the user

↓

Data is processed

↓

User is given the output

The Pascal programming has the structure of structured programming as well.

Object oriented programming is a programming paradigm based on the concept of **objects**. Objects consists of data and methods. Methods are codes that are in the form of procedures that handle data. These data structures exist in the form of fields which are called **Attributes**. Class is the basic structure of object oriented programming. Class describes the behaviours of data and instances. **Class** can create objects of same type.

An object-oriented program consists of many objects and interacting with each other by sending messages

Library Information Systems

Structured Approach
Decompose by functions or processes

Object Approach
Decompose by objects or concepts

Class   Object

Dog   Create Instance   Tommy

Property (Data)	Methods (Code)	Property Values	Methods
Color	Sitting	Color - Gray, White and Black	Sitting
Eye Color	Laying	Eye Color - Blue and Brown	Lying
Height	Shaking	Height - 18 Inches	Shaking
Length	walking	Length - 36 Inches	Walking
Weight		Weight - 30 Pounds	

Procedural Languages

Computation involves code operating on Data

Code (Methods)

Data (Property)

Object-Oriented Languages

An object encapsulates both code and data

Code (Methods)

Data (Property)

Computation involves objects interacting with each other

Programming and scripting

Usually, there are hard and fast syntax rules in programming languages. These should often be compiled. Programming languages need to be compiled. Further, scripting languages should be interpreted.

Both JavaScript and PHP are scripting languages.

1.11.5 Language translators

Programs written in any computer language except in machine language (object code) should be translated to machine language instructions before execution.

A program written in Assembly language is translated to machine language instructions using a language translator called Assembler.

Two language translators are used to translate a program written in high-level language to machine language instructions.

1. Interpreter
2. Compiler

Interpreter

This is the language translator which translates each statement written in a high-level computer language to machine language commands one by one and the translated program is executed using the necessary commands instantly.

When translating object codes in computer languages which use interpreters,

1. If there are no syntax errors in the program, the ststement will be executed
2. If there are syntax errors in the program, it will not execute to the end. (it is possible to operate it till the error is reached)

Important

An interpreter translates code each time the program is executed.

| A program written in high level language | → | Interpreter | → | Machine language instructions |

Compiler

Compiler translates the entire program written in a high level language to machine language as a whole, before it could be executed.

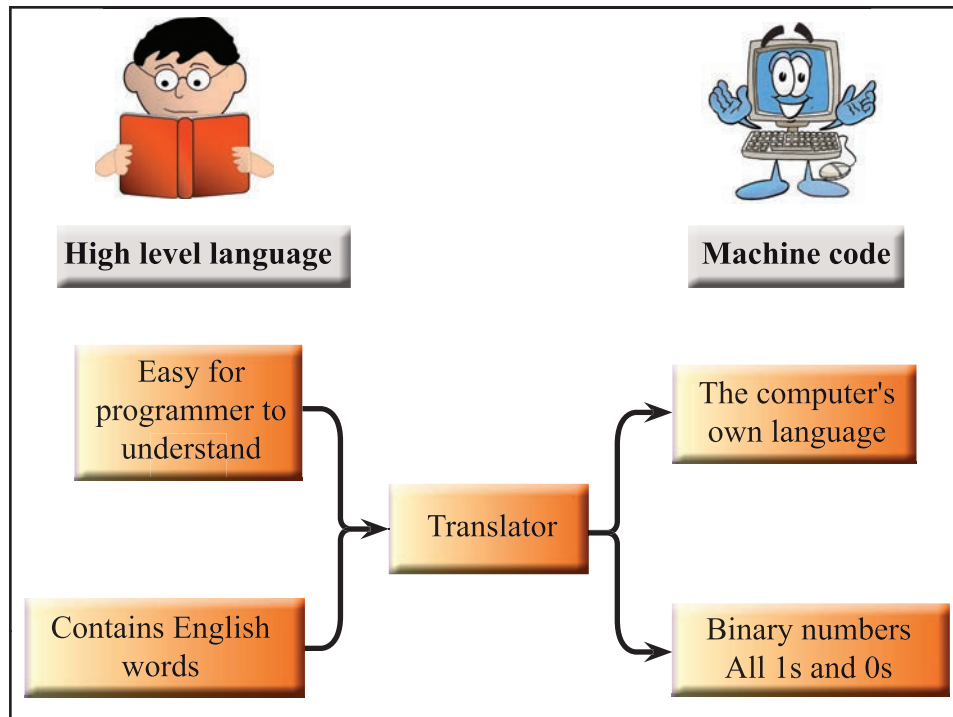| A program written in high level language | → | Compiler | → | Machine language instructions |

When translating source code to machine code in computer languages which use compilers.

1. If there are no syntax errors in the program, code is translated to machine code.
2. If there are syntax errors in the program, it is not be translated to machine code. These errors are highlighted.

Important

After the program is translated to machine code once, it can be execute any number of times. A translation is needed again only if the source code is changed.

High level language		Machine code
Easy for programmer to understand		The computer's own language
	Translator	
Contains English words		Binary numbers All 1s and 0s

Summary

- Input, output and the process can be identified by analyzing a problem.
- Flowcharts and pseudo codes can be used to build an algorithm.
- The basic sequence used in writing any sequence is selection and repetition.
- Sequence is processing some or all the steps in an algorithm one after the other from the beginning to the end.
- Selection is a situation where the step executed should be decided depending on a condition being fulfilled or not.
- If a step or some steps in an algorithm are repeated until a condition is fulfilled or to maintain the condition being fulfilled, it is called repetition.
- A name used to identify a variable, constant or a program is called an identifier.
- If the values assigned for identifiers are changed when the program is being executed, these are called variables.
- Pascal is a high level programming language.

- Evaluation of an expression occurs depending on the priority level of functions.

- Machine language and Assembly language are considered low–level programming languages.

- Languages like PASCAL, BASIC, C and JAVA are examples of high–level programming languages.

- A program written using Machine language can be directly run in the processor.

- A program written in any computer language except in Machine language should be translated to Machine language instructions before it runs.

- Interpreter and Compiler are two translation programs.