



GREEN HALL
Resource Center

Article # 252

A
Level

COMPUTER SCIENCE

Subject Code 9608

PAPER - 2

FAWAD KHAN



GREEN HALL
Resource Center

For Books Order: 0336-5314141

readwrite.org readandwritepublications@gmail.com [f GreenHallResourceCenter](https://www.facebook.com/GreenHallResourceCenter)

*Muatafa
Asif BDC*

A_{Level}

COMPUTER SCIENCE

Paper-2

Article # 252

A LEVEL NOTES SERIES

CIE SYLLABUS CODE 9608

2017 – 18 Edition

Fawad Khan

Visiting teacher at:

Green Hall Academy
Alpha DHA
Beaconhouse School System
City Ravi



GREEN HALL
Resource Center

GULBERG | JOHAR TOWN | WAPDA TOWN | SADDAR CANTT | DHA PHASE-I | DHA PHASE-IV
For Books Order: 0336-531-4141



greenhall.edu.pk



greenhallacademy@gmail.com



greenhalleducation

Contents

Unit 1: Teacher's Guide	4
1.1 Context	4
1.2 Outline	4
1.3 Teaching time	4
1.4 Programming languages	4
1.5 VB.Net Resources	6
Unit 2: Algorithms	7
2.1 Pseudocode	17
2.2 Program Flowcharts	24
Unit-3: Procedural high-level programming languages	24
3.1 Program statements	24
3.2 Programming terms	25
3.3 Writing maintainable programs	26
3.4 Matching Pseudocode and VB codes	28
3.5 Sample illustrations + VB Practical on basic programming constructs	39
3.6 Further exercises on VB basic programming constructs	42
Unit-4: Structure program development	42
4.1 Modular/top down approach/design or stepwise refinement	46
4.2 Top down design/Structure/Jackson diagram	50
Unit-5: Subroutines	50
5.1 Procedure	51
5.2 Function	53
5.3 Parameters and local and global variables	61
5.4 Samples illustrations + VB practical on procedures, function, parameter (known as formal parameters) and arguments (known as actual parameters)	65
5.5 Further exercises on functions and procedures	70
Unit-6: Structure Chart	70
6.1 Symbols used in structure chart	71
6.2 Structure chart representation of sequence code	74
6.3 Structure chart representation for selection code	75
6.4 Structure chart representation of Iteration code	77
6.5 Combination of sequence, selection and iteration code	79
6.6 Further exercises on structure chart	82
Unit-7: VB in Built Functions	82

7.1	Maths functions.....	82
7.2	Random Number generation.....	84
7.3	String handling functions.....	88
Unit-8: Arrays	94
8.1	Array - One dimensional	94
8.2	Sample illustrations + VB practical on 10 arrays	97
8.3	Arrays – Multidimensional	101
8.4	Sample illustration of using 2D array:.....	103
Unit-9: Sorting	106
9.1	Bubble sort	106
Unit-10: Files	114
10.1	Keys terms associate with files.....	114
10.2	Types of files	114
10.3	File manipulation	118
10.4	File extensions	118
10.5	Pseudocode structures for file handling.....	118
10.6	Sample illustration of manipulating serial files	120
10.7	Sample illustration of manipulating sequential files.....	125
Unit-11: Identifier Table	129
Unit-12: Software Development	132
12.1	Stages of program development life cycle (PDLC)	132
12.2	System maintenance.....	133
12.3	Program testing	134
12.4	Testing strategies	135
12.5	Features found in a typical Integrated Development Environment (IDE).....	139

Unit-1

Teacher's Guide

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

- Context
- Outline
- Teaching time
- Programming languages
- VB.Net Resources

Unit 1: Teacher's Guide

1.1 Context

- Paper 2 should be completed prior to starting Paper 4.
- It is recommended that this paper be taught in a practical way with learners having access to a computer that supports VB.Net (console mode).
- Learners should be encouraged to write their own programs, debug and execute them using a computer with the assistance of their teacher

1.2 Outline

Paper 2 provides learners with knowledge and understanding of the following core aspects of problem-solving and programming:

- Algorithm design
- Data representation
- Programming
- Software development

1.3 Teaching time

- It is recommended to spend about 90 hours on Paper 2.

1.4 Programming languages

- VB.Net (console mode) has been chosen as programming language for this book because it is procedural and supports object oriented programming.

1.5 VB.Net Resources

- www.homeandlearn.co.uk/net/vbnet.html
- <http://www.tutorialspoint.com/vb.net/>
- <http://howtostartprogramming.com/vb-net/>
- <http://www.microsoftvirtualacademy.com/training-courses/vbfundamentals-for-absolute-beginners>
- <http://www.studyvb.com/>

Read & Write Publications
For Books Order: 0336-5

Unit-2

Algorithms

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

- Pseudocode
- Program Flowcharts

Unit 2: Algorithms

An algorithm is a set of instructions that either solves a problem or informs the user that there is no solution.

Representing algorithms

There are two main ways of representing algorithms using:

- Pseudocode
- Flowchart

Common keywords used when writing algorithms

Several keywords are often used to indicate common input, output, and processing operations.

- (i) Keywords used to take data from a user
 - INPUT, READ, OBTAIN, GET
 - Example: Input radius
- (ii) Keywords used to give information to a user
- (iii) Keywords used for processing data
 - COMPUTE, CALCULATE, DETERMINE, DO
 - Example: Calculate volume
- (iv) Keywords used for declaring variables
 - INITIALISE, SET
 - Example: Set count to 0 (Le. count = 0)
- (v) Keywords used to increase a variable value
 - ADD, INCREMENT
 - Example: Add 1 to count (i.e. count = count + 1)
- (vi) Keywords used to decrease a variable value
 - SUB, DECREMENT
 - Example: Decrement count by 1 (i.e., count = count - 1)

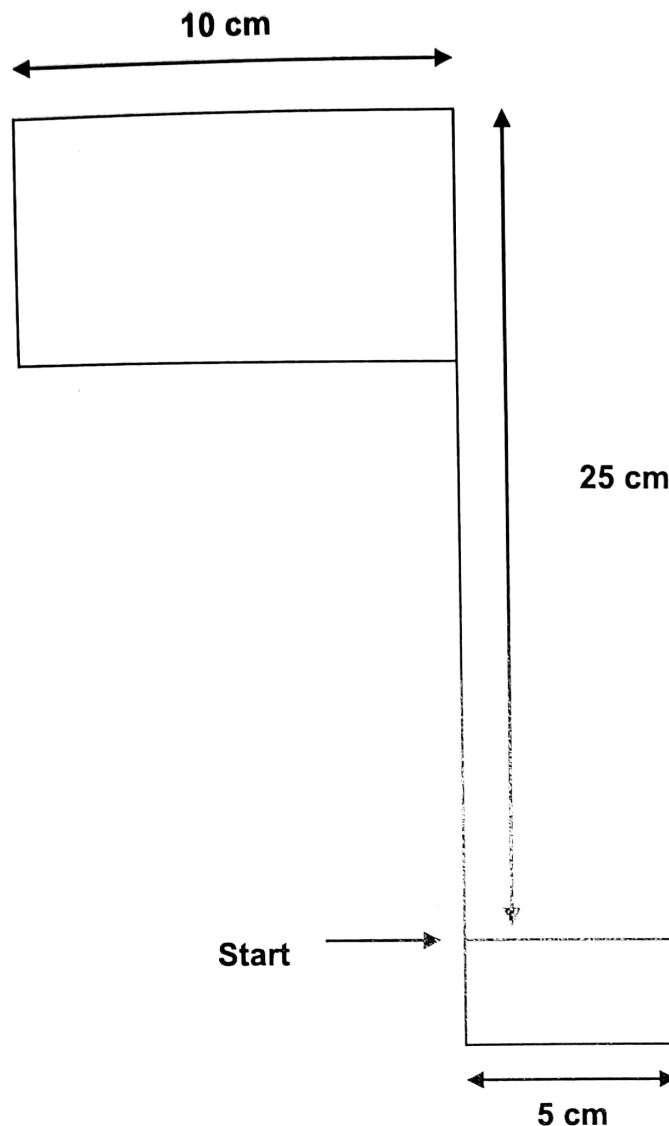
Case Study: Consider the movement of an electronic toy

An electronic toy can move over the floor according to commands that it is given through a keypad. As it moves, it draws a line. It can obey the instructions shown in table below:

Instruction	Meaning
Forward n	Move forward n cm
Backward n	Move backwards n cm
Left n	Turn left n degrees
Right n	Turn right n degrees
Repeat n	Repeat the instruction which follows n times

Class Activity 1:

Write an algorithm for the toy to draw the diagram shown below:

**2.1 Pseudocode**

Pseudocode is the language that combines programming terminology or high-level programming language terms (e.g. if, while, repeat etc.) and ordinary English (nouns and verbs: number, total, set count to 0 etc.). It is a halfway house (i.e. combines two features) between written English and the program code for the problem.

The idea is that, once the pseudocode is written, a programmer using any high-level language (e.g. VB, Java, C ++ etc.) should be able to write the program code from the pseudocode design.

Pseudocode and programming constructs/structures

Algorithms and programs are expressed using 4 basic structures/constructs:

- Assignment construct (Le. assigning values to identifiers)
- Sequence construct (i.e. instructions which are executed one after the other)
- Selection construct (Le. if statements or select case statements)
- Iteration construct (i.e. loops)

2.1.1 Assignment constructs

<variable> = value Or <constant> = value

Example: Declare a variable Name with initial value Fawad and another variable Pie with initial value 3.142

Name ← " Fawad "

Pie ← 3.142

Class Activity 2:

Write a pseudocode that declares two variables (Mark1 and Mark2) that stores initial values 40 and 90 respectively.

2.1.2 Sequence construct

This means a two or more statements are going to be executed one after the other.

Example: Using pseudocode, write an algorithm that calculates the sum of two numbers input by a user.

Input num1

Input num2

Print num1 + num2

Note: The pseudocode below is an example where instructions are not executed in sequence

1. Input num1
2. Goto line 4
3. Print num1 + num2
4. Input num2
5. Goto line 3

Class Activity 3:

Write the Pseudocode that will allow a user to calculate volume of a cylinder using formula.

$$V = \text{pie} * \text{radius} * \text{radius} * \text{height}.$$

2.1.3 Selection constructs

It is used when a decision or comparison has to be made in the form of a condition. If the condition is true, it will execute certain instructions else if it is false, it will execute another set of instructions.

Testing a condition

In selection or condition statements comparisons can be tested using the following operators:

- = (Equals)
- > (Greater than)

- < (less than)
- >= (greater than or equal to)
- <= (less or equal to)
- < > (is not equal to)

Types of selection constructs

We can have two types of selection construct:

- IF construct
- CASE construct

(i) IF construct

There are different types or variations of IF construct/structures including:

- IF structure without 'ELSE' clause
- IF structure with 'ELSE' clause
- Nested IF structure

1. IF structure without 'ELSE' clause

IF <condition> THEN

<statement>

ENDIF

Example: Using pseudocode, write an algorithm that takes a mark from a user and output pass if the mark is above 60.

```

Input mark
If mark > 60 Then
    Print "Pass"
End IF
  
```

Class Activity 4:

Write the pseudocode that take a mark from a user and output fail if the mark is below 40

Class Activity 5:

- (i) Write the pseudocode to input an examination mark and output the grade awarded. The maximum mark is 100. The table below shows the grades that correspond to the marks gained.

Read & Write Publications
 For Books Order: 0336 531441

Mark	Grade
0-59	U
60-69	E
70-74	D
75-79	C
80-84	B
85-89	A
90-100	A*

2. IF structure with 'ELSE' clause

IF <condition> THEN

 <statement>

ELSE

 <statement>

ENDIF

Example: Using pseudocode, write an algorithm that takes a mark from a user and output pass if the mark is above 59 and fail if it is below 59

Input mark

If mark > 59 Then

 Print "Pass"

Else

 Print "fail"

End IF

Class Activity 6:

Write the pseudocode that take a temperature from a user and output high if the temperature is above 40, else output low.

3. Nested IF structure

IF <condition> THEN

 <Statement>

 IF <condition> THEN

 <Statement>

 ELSE

 < Statement >

 END IF

 < Statement >

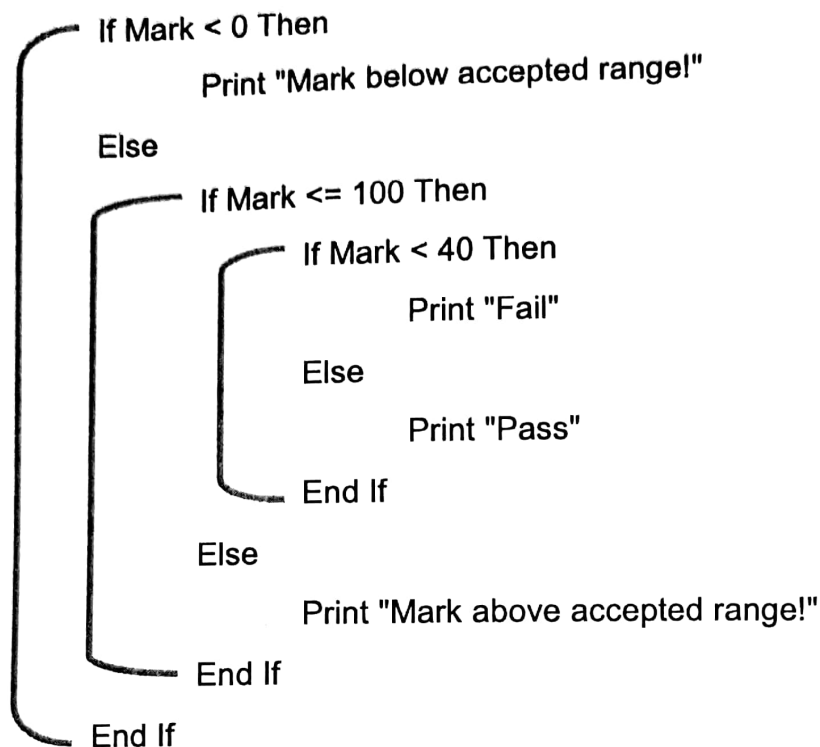
ELSE

 < Statement >

END IF

Read & Write Publications
For Books Order: 0336-537474

An example is illustrated below:



This is called a nested IF structure. It has one IF statement nested (sitting inside) another.

Note: Nested selection

A nested structure (such as an IF statement) is contained inside another similar structure (e.g. IF statement). This is emphasised by indenting the code to show how the statements are nested.

Class Activity 7:

Which algorithm is the more efficient (If structure v/s nested IF structure)?

Example: Using pseudocode, write an algorithm take an age from a user, first check if the age is valid (must be between 0 and 100), if so, print young if the age is below 40 or print old if the age is above 39.

```
Input age
If age > 0 and age <= 100
    If age < 40 Then
        Print "young"
    Else
        Print "old"
    End If
End If
```

Read & Write Publications
For Books Order: 0336-554471

Class Activity 8:

Write pseudocode for a program that takes a mark from a user. Firstly, check that the mark is above 39 and if so, check if the mark is above 90 and below 101, then print "Excellent Mark", else print "Pass Mark". If the mark is below 60, print "fail mark".

(ii) Case construct

Case construct is used when there are multiple options from which to choose.

There are different types or variations of Case structures including:

- Case structure without 'otherwise' clause
- Case structure with 'otherwise' clause

1. Case structure without 'otherwise' clause

CASE OF <identifier>

<value 1>: <statement>

<value 2>: <Statement>

ENDCASE

Example: Using pseudocode, write an algorithm (using case .. structure) that takes a temperature, print "room temperature" if the temperature is from 1 to 37, prints "extreme temperatures" if the temperature is from 38 to 100 or print "freezing temperatures" if temperature is from -15 to 0 degrees Celsius.

Input temperature

Case of temperature

1 to 37: print "room temperature"

38 to 100: print "extreme temperatures"

-15 to 0: print "freezing temperatures"

End Case

Class Activity 9:

Write an algorithm that displays Grade E when student marks are 0 to 20, Grade D when student marks are 21 to 35, Grade C when student marks are 36 to 45, Grade B when student marks are 46 to 60, Grade A when student marks are 60+.

2. Case structure with 'otherwise'

CASE OF <Identifier>

<value 1>: <statement>

<value 2>: <Statement>

OTHERWISE

<statement>

ENDCASE

Read & Write Publications
For Books Order: 0336-27441

Example: -Using pseudocode, write an algorithm that displays Grade E when student marks are 0 to 20, Grade D when student marks are 21 to 35, Grade C when student marks are 36 to 45, Grade B when student marks are 46 to 60, Grade A when student marks are 60+. If students mark are not within the range, print Grade F (use case of ... otherwise structure)

Input grade

Case of grade

0 to 20: Display "Grade E"

21 to 35: Display "Grade D"

36 to 45: Display "Grade C"

46 to 60: Display "Grade B"

60 +: Display "Grade A"

Otherwise:

Display "Grade F"

End Case

Class Activity 10:

Write the pseudocode for a program where someone types in the name of an animal and it outputs the sound the animal makes. The animals it should handle are:

- Pig - Oink
- Cow- Moo
- Bear - Grr
- Sheep - Baa
- Tiger - Grr
- everything else – Meow

2.1.4 Iteration constructs

A loop is used when a block of program instructions is executed a finite number of times according to some condition. Because the instructions inside the loop are repeated, these loops are called iteration, or repetition, constructs.

There are 3 main types of iteration constructs/structures:

- For ... Next loop structure
- Repeat. .. Untill loop structure
- While ... End While structure

(i) For ... Next loop structure

This loop is used when you want to perform the sequence of instructions in a loop a number of times. This construct take the following form:

```
FOR <identifier> ← <start value> TO <end value>
    <statements>
NEXT<identifier>
ENDFOR
```

Read & Write Publications
 For Books Order: 0336-531441

Note: The for loop is a count controlled loop since we know the number of iterations initially (e.g. for count = 1 to 10). The identifier is the variable that will increase or decrease each time and it is known as the loop counter.

Example: Using pseudocode, write an algorithm (use for ... loop structure) that takes 10 numbers from a user and output their total.

```
Total ← 0
For N = 1 to 10
    Read number
    Total ← Total + number
End For
Print Total
```

Class Activity 11:

Write the pseudocode to read an integer, n, and output the squares of the first n integers:

Class Activity 12:

Write the Pseudocode that will allow a user to enter an employee's name, number of hours worked rate of pay. It then calculates the employee's total pay: total pay = rate of pay * number of hours worked. Then it output the name and total pay of the employee. Note: If the name entered is END, the program ends. The Pseudocode must cater for 50 employees.

(ii) Repeat ... Untill loop structure

The sequence on instructions inside the loop is always performed at least once. If the condition is false, the loop repeats, else it stops. This construct take the following form:

REPEAT

<Statement>

UNTIL <condition>

Note: The repeat ... until loop is a post-condition loop since the condition is found at the end, after executing the statements.

Example: Write an algorithm using repeat ... until structure that adds up a series of numbers until the total exceeds 100.

```
Total ← 0
Repeat
    Read number
    Total ← Total + number
Until Total > 100
Print Total
```

Read & Write Publications
For Books Order: 0330-537411

Class Activity 13:

Write an algorithm using repeat ... until structure that will allow a user to continuously input names of students and print them but stop when a name "End" is encountered.

(III) While ... End While loop structure

This structure is used when you do not know how many times the steps inside a loop need to be performed. This loop may not execute at all compared to for ... loop which will execute at least once.

This construct take the following form:

WHILE <condition>

<statement>

ENDWHILE

Note: The while loop is a pre-conditioned loop since the condition is at the start itself, before executing the loop.

Example: Using pseudocode, write an algorithm using while ...loop structure that will allow a user to calculate area of a circle as long as the radius being entered is positive.

```
Pie ← 3.142
Input radius
While radius > 0
    Area ← Pie * radius * radius
    Output area
    Input radius
End While
```

Class Activity 14:

Write an algorithm that will read an item name and display it as long as the total number of item is not equal to 100.

Note: Nested iteration Statements

Loops can also be nested. Consider a program that inputs a student's name followed by a set of examination marks.

```
Hadi 66, 22, 55, -1
Mustafa 34, 22, 77, -1
Fawad 54, 32, 76, -1
```

The process repeats for a number of students and terminates when input of the student name END. The list of marks for each student is terminated by a negative mark. The program outputs the average mark for the student.

Read & Write Publications
For Books Order: 0336-531741

The solution can be:

Input Name

While Name <> "END"

Sum ← 0

Count ← 0

Input Mark

While Mark >= 0

Sum ← Sum + Mark

Count = Count + 1

Input Mark

End While

AverageMark ← Sum/Count

Output "Average mark for" Name, "is " AverageMark

Input Name

End While

Note: Notice the use of nested while loops

2.1.5 Further exercises on pseudocode

- (1) Write an algorithm using pseudocode that asks the user to enter two integers, obtains them from the user and displays the larger number followed by the words "is larger". If the numbers are equal, it must print the message "These numbers are equal".
- (2) Write an algorithm using pseudocode that asks the user to input three integers from the user and displays the sum, average, product, smallest and largest of the numbers.
- (3) Write an algorithm using pseudocode that reads 10 numbers from the user then prints out how many positive numbers and negative numbers user has entered (consider 0 a positive number).
- (4) Write an algorithm using pseudocode that will accept the name, math mark, English mark and French mark of 50 students, calculates the sum, average of the three marks and output the average of each student together with their name.
- (5) Write an algorithm using pseudocode that accept 10 marks and output its average.
- (6) Write an algorithm using pseudocode that counts the number of even and odd numbers when 20 numbers are input.
- (7) (a) Write an algorithm using pseudocode with For... loop structure to input 15 different values for radius and calculate the area of circle in each case.

- (b) Modify the algorithm so that whenever a value less than or equal to 0 is entered for radius, the computer displays a message saying that "Error! Radius must be greater than 0".
- (8) Write an algorithm using pseudocode that accepts 10 temperatures reading in Celsius, calculate and output the temperatures then in Fahrenheit. (Fahrenheit = $32 + 9/5 * \text{Celsius}$).
- (9) Write an algorithm using pseudocode that outputs the smallest and largest number when 10 numbers are input.
- (10) Using Pseudocode, write an algorithm to print the even and odd numbers from 0 until 26.

2.2 Program Flowcharts

- A program flowchart is a diagram that can be very helpful to explain an algorithm.
- Flowchart is combination of two words i.e. flow and chart. Charts consist of different symbols to display information about any program. Flow indicates the direction of processing that takes place in the program. Flowchart is a graphical representation of an algorithm. It is used to show all the steps of an algorithm in a sequence

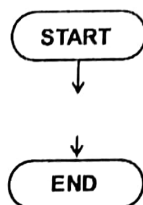
Uses of logic flowcharts

Flowchart is used for the following reasons:

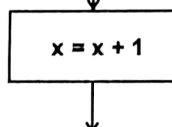
- Flowcharts are used to represent an algorithm in simple graphical manner.
- Flowcharts are used to show the steps of an algorithm in an easy way.
- Flowcharts are used to understand the flow of the program.
- Program can be reviewed and debugged easily.

2.2.1 Symbols used in program flowcharts

The Start and end box:



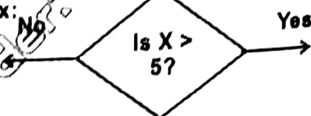
The process box:



Input/Output box:



Decision/query box:



2.2.2 Trace table

It is a technique used to test Pseudocode or flowcharts to make sure that no logical errors occur.

The trace table when built must consist of the

- Inputs: These are data taken from the user

- Variables: They stores changing values
- Output: This is data that is to be printed

2.2.3 Hand tracing/Dry run

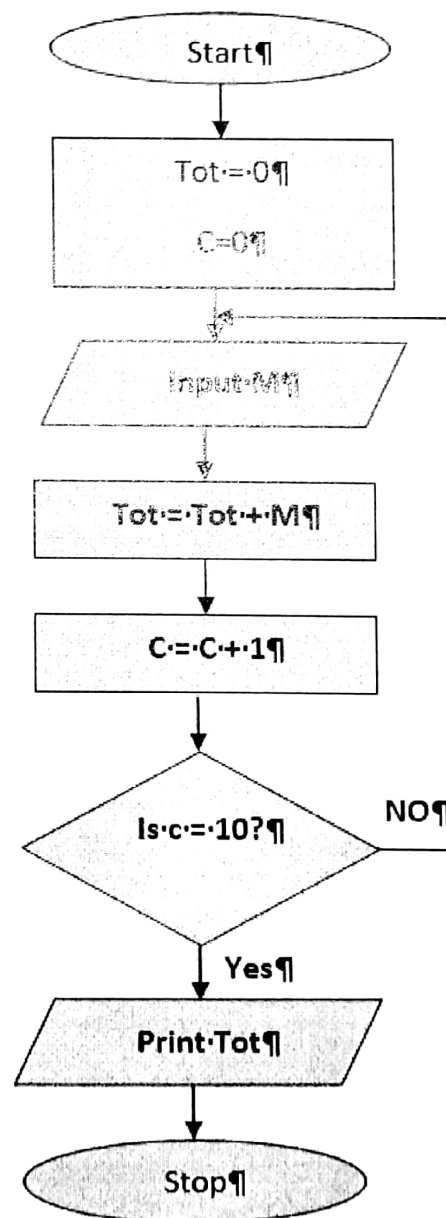
It is the process where you use a trace table to:

- see what the pseudocode or flowchart will do before you have to run it
- find where errors in your pseudocode or flowchart are

2.2.4 Test data

Test data are the values that will be used as sample in the trace table to test whether the pseudocode or flowchart is doing what it should do.

Example: Design a flowchart that takes 10 numbers as input and outputs their total



ite Publications
Order: 0336-531411

For

Draw a trace table using the following test data: M = 2,4,6,8,1,3,5,7,9,10,11

Solution:

Tot	C	M	Output
0	0		
2	1	2	
6	2	4	
12	3	6	
20	4	8	
21	5	1	
24	6	3	
29	7	5	
36	8	7	
45	9	9	
55	10	10	55

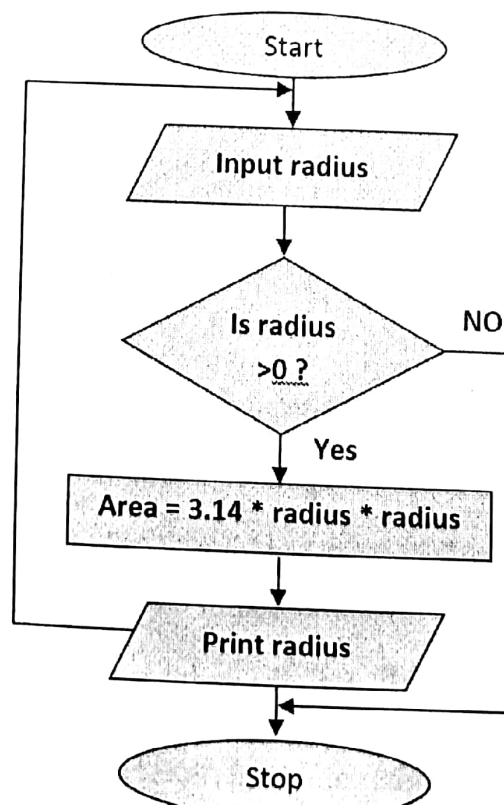
- Note:** (1) Notice that the test data 11 is not taken into consideration since the flowchart cater for 10 values only.
- (2) This flowchart makes use of a counter-controlled loop as shown by the arrow which goes upwards. Counter-controlled loops are used when a flowchart need to take input and processed a fixed number of items.

Class Activity 15:

- (i) Design a flowchart to calculate area of a circle while radius is positive
- (ii) Draw a trace table with test data: $r = 1, 2, -3, 5, 6$

2.2.5 Condition-controlled loop

In the flowchart below, we do not know in advance the number of times we wish the loop to repeat. The flow of data in this case is interrupted by using a rogue value/data terminator. For example here, the loop will repeat until the special data item (a rogue value is that is less 0) is met and thus terminates the loop.



e Publications
Order: 0336-5314141

Class Activity 16:

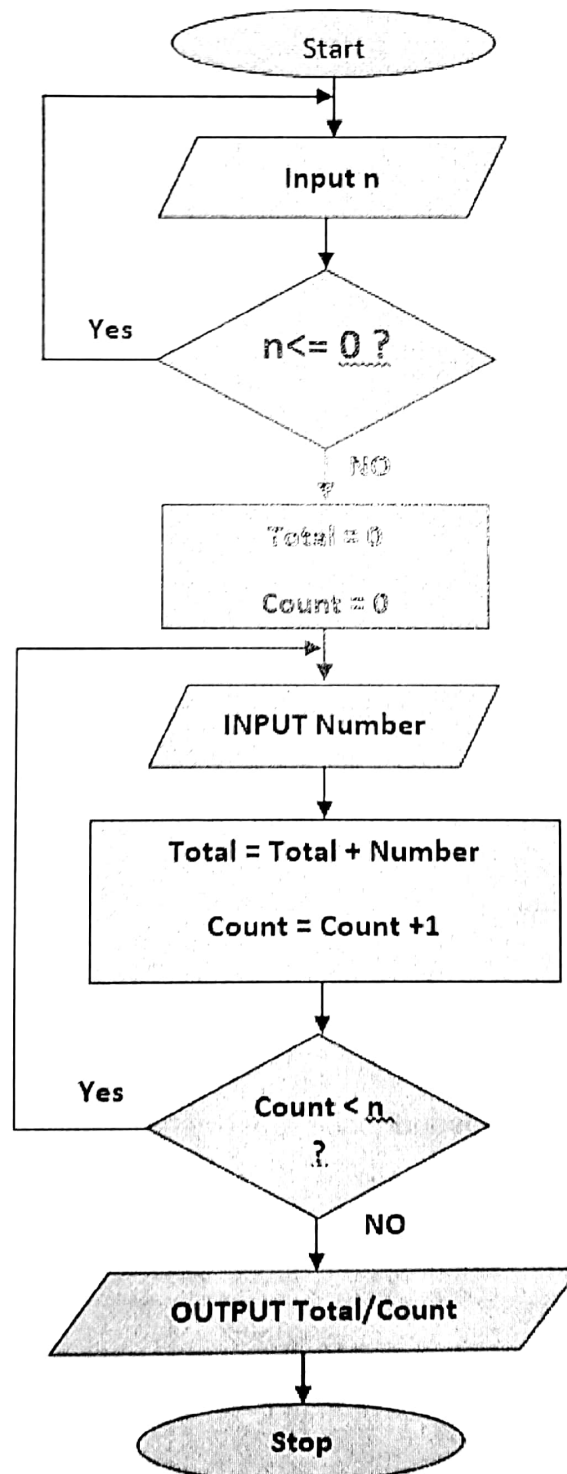
- (i) Design a flowchart to calculate the smallest mark when 5 marks are input (note: marks are from 0 to 100)
- (ii) Draw the trace table using test data for Marks: 80, 50, 60, 40, 10, and 70

Class Activity 17:

- (i) Dry run the following flowchart using the following test data:

N=3

Number = 10, 20, 15



Publications
ver: 0336-5314141

- (ii) State the purpose of the flowchart.
- (iii) Write the pseudocode algorithm for the flowchart above.

2.2.6 Further exercises on flowcharts

1.
 - (i) Design a flowchart to calculate the smallest mark when 5 marks are input (note: marks are from 0 to 100)
 - (ii) Draw the trace table using test data for Marks: 80, 50, 60, 100, 10, 70
2. The pseudocode below that finds the roots of three inputs: a, b and c. [Assuming there is a function SQRT(x) which return the square root of x (x is an integer.)]

Input a

Input b

Input c

$d \leftarrow b * b - 4 * a * c$

If $d < 0$ Then

Output "No real roots"

Else

SquareRoot = SQRT(d)

Root1 $\leftarrow (-b + \text{SquareRoot}) / (2 * a)$

Root2 $\leftarrow (-b - \text{SquareRoot}) / (2 * a)$

Output Root1 , Root2

End If

Draw the flowchart for the above pseudocode and dry run it using your own test data.

3.

Total $\leftarrow 0$

For N = 1 to 10

Read number

Total \leftarrow Total + number

Next N

End For

Print Total

Draw the flowchart for the above pseudocode and dry run it using your own test data.

Read & Write Publications
For Books Order: 0336-5314141

Unit-3

Procedural High-level Programming Languages

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

- Program statements
- Programming terms
- Writing maintainable programs
- Matching Pseudocode and VB codes
- Sample illustrations + VB Practical on basic programming constructs
- Further exercises on VB basic programming constructs

Unit-3: Procedural high-level programming languages

These are languages that require the programmer to specify precisely how a computer is to solve a problem. They consist of a number of instructions that must be executed in the order specified.

Examples of procedural high-level programming languages:

- FORTRAN (FORMula TRANslation) was developed to solve mathematical/scientific problems
- PASCAL (named after the mathematician Pascal) was developed as a teaching language
- BASIC (Beginners All-purpose Symbolic Instruction Code) was developed so that as many people as possible could write computer programs.

These procedural high-level programming languages have been designed to solve different types of problem.

NOTE: All these languages have developed over the years and some languages are derivatives of others. For example, Modula-2 was developed from Pascal. All versions of Visual Basic are derived from the original BASIC.

3.1 Program statements

A statement in a programming language is simply an instruction that can be executed. Each statement can be simple or complex.

Simplex statements:

- Declaration statement: e.g. Dim count as integer
- Assignment statement: e.g. count = count + 1
- Return statement: e.g. return 5
- Call statement: grade (80)

Complex (compound) statements:

- If statement
- While-loop statement
- For-loop statement
- Case statement

3.2 Programming terms

3.2.1 Variables

A variable is defined as a sequence of one or more characters starting with a letter. The characters which follow can be any letter, digit or the underscore character. Variables stores changing values of identifiers throughout the execution of a program compared.

Re & Write Publications
Order: 0336-531411

Re
For Ex

3.2.2 Constants

Constants have the same value throughout the execution of the program.

3.2.3 Identifiers

Identifiers are valid names that represent the data items that the problem or program uses. E.g. name, num, num7, stud7name.

Different programming languages impose constraints on valid names. For example, identifiers in Visual Basic cannot be more than 64 characters and must not start with a digit. E.g. 7name is an invalid identifier and so cannot be used since it starts with a digit.

Example of valid identifiers:

- Name
- Name78
- Num1
- Num 1
- Num1stud2

Example of invalid identifiers:

- 5address
- _number
- \$name
- ..num

Generally, an identifier can be a variable, a constant, function name, procedure name etc. Identifiers in a program code can be represented in an Identifier table. This will be discussed in details in the sections which follow.

3.2.4 Comments

They are programmer-readable annotation in the source code of a computer program that helps programmers understand the code but is usually ignored at run time.

3.3 Writing maintainable programs

We need to make use of various techniques when writing pseudocode to help with the "readability" of the program code or pseudocode. These include:

(i) Use of Indentation

The indentation shows the start and end of conditional statements to help with reading and understanding the algorithm. Indentation also shows the start and end of statements that are repeated.

(ii) Use of white space

The previous pseudocode examples have left a line of white space above and below the loop and conditional structures.

(iii) Use of Numbering (for pseudocode)

It is sometimes useful to number the steps of an algorithm. The need for this is more apparent in more complex algorithms.

(iv) Use of comments/annotation

Program code should be well commented so that any person can easily go through the code and make modifications if necessary.

(v) Use of meaningful identifiers

Identifier should have meaningful name so that they are understood at first glance. E.g. declare a variable as radius instead of r.

(vi) Use standard convention for programming terms

It is a good way of programming to use standard convention for identifiers such as all variable names start with var followed by the its name, all constant start win cons followed by its name. E.g. varRadius

3.4 Matching Pseudocode and VB codes

In this section, you are going to have an idea of the similarities between pseudocode and VB program code.

3.4.1 Data types

A computer use different types of data in the operation of the system.

Database data types	VB data types
Number	Integer, Single, Double (for floating point numbers), Long (Int32), Short (Int16), Decimal (for decimal numbers)
Text	String
Yes/No (Boolean values)	Boolean
Date/Time	Date
Char	Char (Single Character)

3.4.2 Programming Constructs

Construct	In Pseudocode	In VB code
Assignment	←	=
Iteration	For <variable> = X to Y <instructions> END FOR	For <variable> = X to Y <instructions> Next
Iteration	Repeat <instructions> Until <condition>	Do <instructions> Loop Until <condition>
Iteration	While <condition> <instructions> End While	Do While <condition> <instructions> Loop
Selection	IF <condition> Then <instructions> End IF	IF <condition> Then <instructions> End IF
Selection	Case <variable> of <value1> to <value2> : <instructions> <value3> to <value4> : <instructions> Otherwise <instructions> End Case	Select Case <variable> Case <value1> : <instructions> Case <value2> : <instructions> Case Else <instructions> End Select

3.4.3 Operators

Visual Basic .NET provides a basic set of operators to calculate simple arithmetic.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Integer Division (e.g. $7 \setminus 2 = 3$, $25 \setminus 4 = 6$) It gives the result without the remainder
Mod	Remainder division (e.g. $8 \text{ Mod } 2 = 0$, $7 \text{ Mod } 2 = 1$)
&	String concatenation (e.g. "Cat" & "Dog" will result in "Cat Dog")
^	Exponentiation (e.g. $7 \wedge 2 = 49$)

3.4.4 BODMASS

You have probably learnt about the order of operations in maths. BODMAS also applies to computer calculations. This means that when calculating a sum, the program will calculate:

- Brackets
- Order (powers n^2 etc.)
- Division
- Multiplication
- Addition
- Subtraction

3.4.5 Syntax in VB and their meaning

Syntax	Meaning
Dim <variable> as <data type>	Use to declare variables
Const <constant> as <data type> =<constant value>	Use to declare constants
<variable> = Console.readline()	Use to take and store input from user
Console.writeline("<output>")	Use to output instructions to user on console
Console.writeline(<variable>)	Use to output the value of the variable to user on console

3.5 Sample illustrations + VB Practical on basic programming constructs

Example 1: Using pseudocode, write an algorithm that declare a variable Name with initial value "Fawad" and another variable Pie with initial value 3.142

```
Name ← " Fawad "
Pie ← 3.142
```

VB program code

```
Sub main()
```

```
    Dim name As String ' declare variables using Dim keyword
```

```
    Const pie As Decimal = 3.142 'declare constants using Const keyword
```

```
    name = "Fawad" 'assign value to variable name
```

```
End Sub
```

Class Activity 18

Write the VB program code that declares two variables (Mark1 and Mark2) that stores initial values 40 and 90 respectively.

Example 2: Using pseudocode, write an algorithm that calculates the sum of two numbers input by a user.

```
Input num1
Input num2
Print num1 + num2
```

VB Program code

```
Sub main()
    Dim num1 As Integer
    Dim num2 As Integer
    Num1= console.ReadLine() 'take value from user and store in num1
    num2 = Console.ReadLine() 'take value from user and store in num2
    Console.WriteLine(num1 + num2) 'adds the two values and output their sum
End Sub
```

Class Activity 19

Write the VB program code that will allow a user to calculate volume of a cylinder using formula. $V = \pi * \text{radius} * \text{radius} * \text{height}$.

Example 3: Using pseudocode, write an algorithm that takes a mark from a user and output pass if the mark is above 59.

```
Input mark
If mark > 59 Then
    Print "Pass"
End IF
```

VB program code

```
Dim mark As Integer
console.WriteLine("please enter a mark: ")
mark = console.ReadLine()
If mark > 59 Then
    Console.WriteLine("You have passed!")
End If
```

Class Activity 20

Write the VB program code de that takes a mark from a user and output fail if the mark is below 60.

Read & Write Publications
For Books Order: 0336-5314141

Example 4: Using pseudocode, write an algorithm that takes a mark from a user and output pass if the mark is above 59 and fail if it is below 59

```
Input mark
If mark > 59 Then
    Print "Pass"
Else
    Print "fail"
End IF
```

VB program code

```
Dim mark As Integer
Console.WriteLine("please enter a marks")
mark = Console.ReadLine()
If mark > 59 Then
    Console.WriteLine("You have passed! ")
Else
    Console.WriteLine("You have failed!")
End If
```

Class Activity 21

Write the VB program code that take a temperature from a user and output high if the temperature is above 40, else output low.

Example 5: Using pseudocode, write an algorithm take an age from a user, first check if the age is valid (must be between 0 and 100), if so, print young if the age is below 40 or print old if the age is above 39

```
Input age
If age > 0 and age <= 100
    If age < 40 Then
        Print "young"
    Else
        Print "old"
End If
```

VB Program Code

```
Dim age as integer
Console.WriteLine("Enter an age: ")
age = Console.ReadLine()
If age > 0 And age <= 100 Then
    If age < 40 Then
        console.WriteLine("You are young!")
    Else
```

Read & Write Publications
For Books Order: 0336-531441

```
        Console.WriteLine("You are old! ")
    End If
Else
    Console.WriteLine ("oh..an alien in the vicinity ")
End If
```

Class Activity 22

Write the program code that takes a mark from a user. Firstly, it checks that the mark is above 39 and if so, checks if the mark is above 90 and below 101, then print "Excellent Mark", else print "Pass Mark". If the mark is below 40, print "fail mark".

Example 6: Using pseudocode, write an algorithm (using case ..structure) that takes a temperature, print "room temperature" if the temperature is from 1 to 37, prints "extreme temperatures" if the temperature is from 38 to 100 or print "freezing temperatures" if temperature is from -15 to a degrees celcius.

```
Input temperature
Case of temperature
    1 to 37: print "room temperature"
    38 to 100: print "extreme temperatures"
    -15 to 0: print "freezing temperatures"
End Case
```

VB program code

```
Dim temp As Integer
Dim output As String = ""
Console.WriteLine ("Enter a temperature: ")
temp = Console.ReadLine()

Select Case temp
Case -15 To 0
    Output "Freezing temperature!"
Case 1 To 37
    output "Room temperature!"
case 38 to 100
    output "Extreme temperatures!"
End select
Console.WriteLine (output)
```

Read & Write Publications
For Books Order: 0336-531411

Class Activity 23

Write the VB program code that displays Grade E when student marks are 0 to 20. Grade D when student marks are 21 to 35, Grade C when student marks are 36 to 45, Grade B when student marks are 46 to 60, Grade A when student marks are 60+.

Example 7: Using pseudocode, write an algorithm that displays Grade E when student marks are 0 to 20, Grade D when student marks are 21 to 35, Grade C when student marks are 36 to 45, Grade B when student marks are 46 to 60, Grade A when student marks are 60+. If students mark are not within these range, print Grade F (use case of ...otherwise structure)

Input grade

Case of grade

0 to 20: Display "Grade E"

21 to 35: Display "Grade D"

36 to 45: Display "Grade C"

46 to 60: Display "Grade S"

60 +: Display "Grade An"

Otherwise:

Display "Grade F"

End Case

VB program code (Sample 1)	VB program code (Sample 2)
<pre> Dim mark As Integer Console.WriteLine("Enter a mark: ") mark = Console.ReadLine() Select case mark Case 0 To 20 Console.WriteLine("Grade E") Case 21 To 35 Console.WriteLine("Grade D") Case 36 To 45 Console.WriteLine("Grade C") Case 46 To 60 Console.WriteLine("Grade B") Case Is > 60 Console.WriteLine("Grade A") Case Else Console.WriteLine("Grade F") End Select </pre>	<pre> Dim mark As Integer Dim grade As String console.WriteLine("Enter a mark: ") mark = Console.ReadLine() Select Case mark Case 0 To 20 Grade = "Grade E" Case 21 To 35 Grade = "Grade D" Case 36 To 45 Grade = "Grade C" Case 46 To 60 Grade = "Grade B" Case Is > 60 Grade = "Grade A" Case Else Grade = "Grade F" End Select Console.WriteLine(Grade) </pre>

Class Activity 24

Write program code for the following using case structure:

Mark	Comment
0 to 49	Need to work harder
50 to 59	Average
60 to 69	Above Average
70 to 84	Good
85 to 100	Excellent
Any other mark	Wrong entry

Example 8

Write the program code which asks a user if he or she is single and output the following comments:

Answer	Output
Yes	You will save a lot of money
No	You will go bankrupt soon
Any other answers	Life doesn't look promising

```
Dim answer As string
```

```
Console.WriteLine("Are you single. Type yes or no ")
```

```
answer = Console.ReadLine()
```

```
Select Case answer
```

```
    Case "yes"
```

```
        console.WriteLine("You will save a lot of money in future ")
```

```
    Case "no"
```

```
        Console.WriteLine("You are going to be bankrupt soon ")
```

```
    Case else
```

```
        Console.WriteLine("life doesn't look promising!")
```

```
End Select
```

Class Activity 25

Write the program code for a program where someone types in the name of an animal and it outputs the sound the animal makes. The animals it should handle are:

- Pig - Oink
- Cow - Moo
- Bear - Grr
- Sheep - Baa
- Tiger - Grr
- everything else - Meow

Note: We can also use case structure to select from a specific set of values as shown below:

Read & Write Publications
For Books Order: 0351-531441

```
Dim number As Integer = 8
```

```
    Select Case number
```

```
        Case 1 To 5
```

```
            Console.WriteLine("Between 1 and 5, inclusive ")
```

'The following is the only case clause that evaluates to true.

```
        Case 6, 7, 8
```

```
            Console.WriteLine("Between 6 and 8, inclusive ")
```

```
        Case 9 To 10
```

```
            Console.WriteLine("Equal to 9 or 10 ")
```

```
        Case Else
```

```
            Console.WriteLine("Not between 1 and 10, inclusive ")
```

```
    End Select
```

Example 9: Using pseudocode, write an algorithm (use for ... loop structure) that takes 10 numbers from a user and output their total.

```
Total = 0
```

```
For N = 1 to 10
```

```
    Read number
```

```
    Total = Total + number
```

```
Next N
```

```
End For
```

```
Print Total
```

VB program code

```
Dim number, N As Integer
```

```
    Dim Total As Integer = 0
```

```
    Total = 0
```

```
    For N = 1 To 10
```

```
        Console.WriteLine("Enter a Number ")
```

```
        number = Console.ReadLine
```

```
        Total = Total + number
```

```
    Next N
```

```
    Console.WriteLine("The total of these numbers is: " & Total)
```

Class Activity 25

Write the program code that that will allow a user to enter an employee's name, number of hours worked rate of pay. It then calculates the employees' total pay: total pay = rate of pay * number of hours worked. Then it output the name and total pay of the employee. Note: If the name entered is END, the program ends. The Pseudocode must cater for 50 employees.

Note: We can use a special reserved word "Step" when using for loop in VB to increment or decrement the count value by a specific amount

(i) Increment value by 1 each time

For count = 1 To 10

 Console.WriteLine(count)

Next

file:///C:/Users/Fawad Khan/AppData/Lc

```
1
2
3
4
5
6
7
8
9
10
Press any key to continue.....
```

(ii) Increment value by 2 each time

For count = 1 To 10 Step 2

 Console.WriteLine(count)

Next

file:///C:/Users/Fawad Khan/AppData/L

```
1
3
5
7
9
Press any key to continue.....
```

(iii) Increment value by 3 each time

For count = 1 To 10 Step 3

 Console.WriteLine(count)

.....Next

file:///C:/Users/Fawad Khan/AppData/L

```
1
4
7
10
Press any key to continue.....
```

(iv) Decrement value by 1 each time

For count = 10 To 1 Step -1

 Console.WriteLine(count)

Next

file:///C:/Users/Fawad Khan/AppData/L

```
10
9
8
7
6
5
4
3
2
1
Press any key to continue.....
```


(v) **Decrement value by 2 each time**

For count = 10 To 1 Step -2

 Console.WriteLine(count)

Next

file:///C:/Users/Fawad Khan/AppData/

10

8

6

4

2

Press any key to continue.....

Example 10: Using pseudocode, write an algorithm using nested for loop that ask the user to enter a value *n* and draws the following figure if:

n=3

n = 4

Input *N*

For *i* = 1 to *N*

 For *j*= 1 to *N*

 Print "*"without changing line

 End For

 Change line

End For

VB program code

Dim N As Integer

 Console.WriteLine("Enter A value N: ")

 N = Console.ReadLine()

 For i = 1 To N

 For j = 1 To N

 Console.Write("*")

 Next

 Console.WriteLine()

 Next

Class Activity 26

Write the program code using nested for loop that ask the user to enter a value *n* and draws its identity matrix.

Read & Write Publications
For Books Order: 0336-531441

E.g. If $n = 4$, then output

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

Example 11: Using pseudocode, write an algorithm using repeat ... until structure that adds up a series of numbers until the total exceeds 100.

Total = 0

Repeat

 Read number

 Total = Total + number

Until Total > 100

Print Total

VB program code

Dim total As Integer

 Dim number As Integer

 total = 0

 Do

 Console.WriteLine("Enter a number ")

 number = Console.ReadLine

 total = total + number

 Loop Until total > 100

 Console.WriteLine("Total is: " & total)

Class Activity 27

Write the program code using repeat. ..Until structure that will allow a user to continuously input names of students and print them but stop when a name "End" is encountered.

Example 12: Using pseudocode, write an algorithm using while ... loop structure that will allow a user to calculate area of a circle as long as the radius being entered is positive.

Pie = 3.142

Input radius

While radius > 0

 Area = Pie * radius * radius

 Output area

 Input radius

EndWhile

Read & Write Publications
For Books Order: 0306-301141

VB Program Code (Sample 1)

Dim area As Decimal

Const pie As Decimal = 3.142

Dim radius As Integer

Console.WriteLine("Enter a radius ")

radius = Console.ReadLine

While radius > 0

area = pie * radius * radius

Console.WriteLine("Area is " & area)

Console.WriteLine("Enter a radius ")

radius = Console.ReadLine

End While

VB Program Code (Sample 1)

Dim area As Decimal

Const pie As Decimal = 3.142

Dim radius As Integer = 1

While radius > 0

Console.WriteLine("Enter a radius ")

radius = Console.ReadLine

area = pie * radius * radius

Console.WriteLine("Area is " & area)

End While

Note: radius has been given the initial value 1 simply so that the loop is executed. The radius is then overwritten by the value input by the user. We can use do while loop also instead of while ... end while as shown below

Dim area As Decimal

Const pie As Decimal = 3.142

Dim radius As Integer = 1

Do While radius > 0

Console.WriteLine("Enter a radius ")

radius = Console.ReadLine

area = pie * radius * radius

Console.WriteLine("Area is " & area)

Loop

Read & Write Publications
For Books Order: 0336-531441

Class Activity 28

Write an algorithm that will read an item name and display it as long as the total number of item is not equal to 100.

3.6 Further exercises on VB basic programming constructs

1. Write the program code that will accept the name, math mark, English mark and French mark of 50 students, calculates the sum, average of the three marks, and output the average of each student together with their name.
2. Write a program in VB that accept 10 marks and output its average.
3. Write the program code that counts the number of even and odd numbers when 20 numbers are input.
4.
 - (a) Write a VB program with For ... Do loop structure to input 15 different values for radius and calculate the area of circle in each case.
 - (b) Modify the program so that whenever a value less than or equal to 0 is entered for radius, the computer displays a message saying that "Error! Radius must be greater than 0"
5. Write the program code that accepts 10 temperature reading in Celsius, calculate and output the temperatures then in Fahrenheit. (Fahrenheit = $32 + 9/5$ Celcius)
6. Write a program in VB that outputs the smallest and largest number when 10 numbers are input.
7. Write the program code to print the even number from 0 until 26.
8. Write a program in VB that calculates the average of 3 numbers using for loop.
9. Write a program in VB that calculates the average of 3 numbers using while loop.
10. Write the program code that solves a simultaneous equation.
11. Write a program to calculate the estimated journey time of a train. The program should request and read the distance in miles and the average speed in miles per hour, and display the estimated journey time in hours and minutes to the nearest minute.

Read & Write Publications
For Books Order: 0336-531414

Unit-4

Structure Program Development

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

- Modular/top down approach/design or stepwise refinement
- Top down design/Structure/Jackson diagram

Unit-4: Structure program development

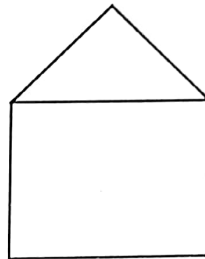
Structured programming is a method of writing computer programs that use

- Top-down analysis for problem solving
- Modularization for a program structure and organisation and
- Structured code for individual modules

The full explanation of these three ideas is illustrated in the sections which follow.

Case study:

We need to find the area of a 'house' made up from a square and a triangle as shown below:



To do so:

- Firstly, we work out the area of the triangle
- Secondly, we work out the area of the square
- Finally, we add the two areas to obtain the area of the house

Thus, as you can deduce, we had our main problem as finding the area of a house. To do so, we broke our main problem in 3 sub-problems, that is, the 1st sub-problem was to work out the area of the triangle, the 2nd sub-problem was to work out the area of the square and 3rd sub problem was to add the result from the 1st and 2nd sub-problems. This approach is known as **top down approach**.

This approach is also adopted when writing computer programs. Before writing the program, we need to have a thorough understanding of the problem so that we can carefully plan how to solve it.

4.1 Modular/top down approach/design or stepwise refinement

Modular approach is an approach to problem solving where the main problem is divided up into a number of smaller problems known as modules which are more manageable. Each of the smaller problems is solved separately. The individual solutions are combined to give a solution to the whole problem.

In general, top down stepwise refinement help a person think through an algorithm by doing the following:

- Start with the initial problem statement
- Break it into a few general steps
- Take each "step" and break it further into more detailed steps

- Keep repeating the process on each "step", until you get a breakdown that is pretty specific, and can be written more or less in pseudocode
- Translate the pseudocode into program code

Example 1

Stepwise refinement for making tea:

Original algorithm

1. Make tea
 - 1.1. Boil Water
 - 1.2. Put tea leaves
 - 1.3. Put milk and sugar

First Refinement

1. Make tea
 - 1.1. Boil Water
 - 1.1.1. Put water in recipient
 - 1.1.2. Put gas on
 - 1.2. Put tea leaves
 - 1.2.1. Get tea leaves from cupboard
 - 1.2.2. Tear tea bag
 - 1.3. Put milk and sugar
 - 1.3.1. Get milk from milk tub
 - 1.3.2. Get sugar from sugar tub

Note: Refinement means replacing existing steps/instructions with a new version that fills in more details. There can be a 2nd refinement, 3rd refinement and so forth if you want to add more details. Stepwise refinement may differ depending on how much details you want to include in it. However, the more details it contains, the easier it becomes to write the program code.

Class Activity 29

Do the stepwise refinement for doing your homework at home

Example 2

Stepwise refinement for brushing the teeth:

Original algorithm

1. Brush Teeth
 - 1.1. Find toothbrush
 - 1.2. Find toothpaste tube
 - 1.3. Open toothpaste tube
 - 1.4. Squeeze tube onto toothbrush

Read & Write Publications
For Books Order: 0336-531414

1.5. Clean teeth

1.6. Clean up

First Refinement

1. Brush Teeth

1.1. Find toothbrush

1.2. Find toothpaste tube

1.3. Open toothpaste tube

1.3.1. Put thumb and pointer finger on cap

1.3.2. Turn fingers counter-clockwise

1.3.3. Repeat prior step until cap falls off

1.4. Squeeze tube onto toothbrush

1.5. Clean teeth

1.5.1. Put brush on teeth

1.5.2. Move back and forth vigorously

1.5.3. Repeat above step 100 times

1.6. Clean up

1.6.1. Rinse brush

1.6.2. Put cap back on toothpaste

Second Refinement

1. Brush Teeth

1.1. Find toothbrush

1.2. Find toothpaste tube

1.3. Open toothpaste tube

1.3.1. Put thumb and pointer finger on cap

1.3.2. Turn fingers counter-clockwise

1.3.3. Repeat prior step until cap falls off

1.4. Squeeze tube onto toothbrush

1.5. Clean teeth

1.5.1. Put brush on teeth

1.5.2. Move back and forth vigorously

1.5.3. Repeat above step 100 times

1.6. Clean up

1.6.1. Rinse brush

1.6.1.1. Place brush under tap

1.6.1.2. Open tap and hold toothbrush for 30 seconds

1.6.1.3. Close tap

1.6.2. Put cap back on toothpaste

Class Activity

Do the stepwise refinement for taking your breakfast (show up to 3 levels)

Example 3

Stepwise refinement for calculating area of a circle:

1. Area of circle
 - 1.1. Input radius
 - 1.2. Calculate area
 - 1.3. Output area

Class activity 30

Do the stepwise refinement for calculating the volume of a cylinder

Example 4

Stepwise refinement for calculating the wages for an hourly paid worker:

Original algorithm

1. Calculate Wages
 - 1.1. Input number of hours
 - 1.2. Calculate gross pay
 - 1.3. Calculate deductions
 - 1.4. Calculate net pay
 - 1.5. Output wage slip

1st Refinement

1. Calculate Wages
 - 1.1. Input number of hours
 - 1.2. Calculate gross pay
 - 1.2.1. Calculate normal wages
 - 1.2.2. Calculate overtime
 - 1.3. Calculate deductions
 - 1.3.1. Calculate tax
 - 1.4. Calculate net pay
 - 1.5. Output wage slip

Class Activity 31

Do the stepwise refinement for calculating the average mark of a student in 3 tests.

Read & Write Publications
For Books Order: 0336-5314141

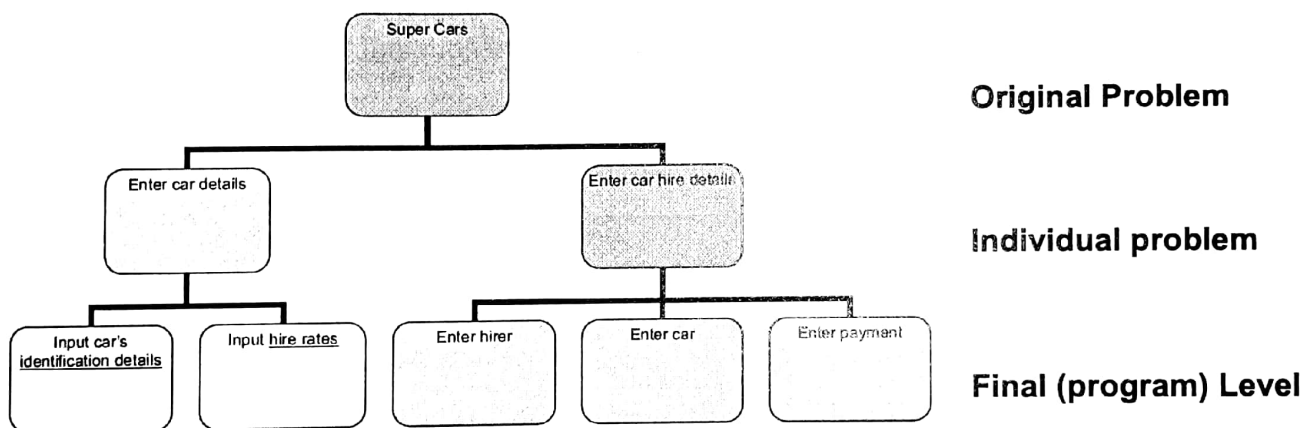
4.1.1 Advantages of top-down design

- Modules are more manageable since complex problems are made simple to solve.
- More than one person can be engaged on solving parts of the same problem simultaneously. Different people are good at different things.
- The person producing a module is likely to make fewer mistakes compare to trying to solve a complex problem as a whole and also the module algorithms are much shorter than for the whole problem
- It is also much easier to test the program code and eliminate errors.
- Modules can be used in more than one project which requires similar solutions. Software developers store the modules that they have produced so that they can be used again if the opportunity arises. Collections of modules like this are known as software or program libraries. (Do not re-invent the wheel!)

4.2 Top down design/Structure/Jackson diagram

A diagram can be used to show the top down/stepwise/modular approach.

The figure below depicts such a diagram:



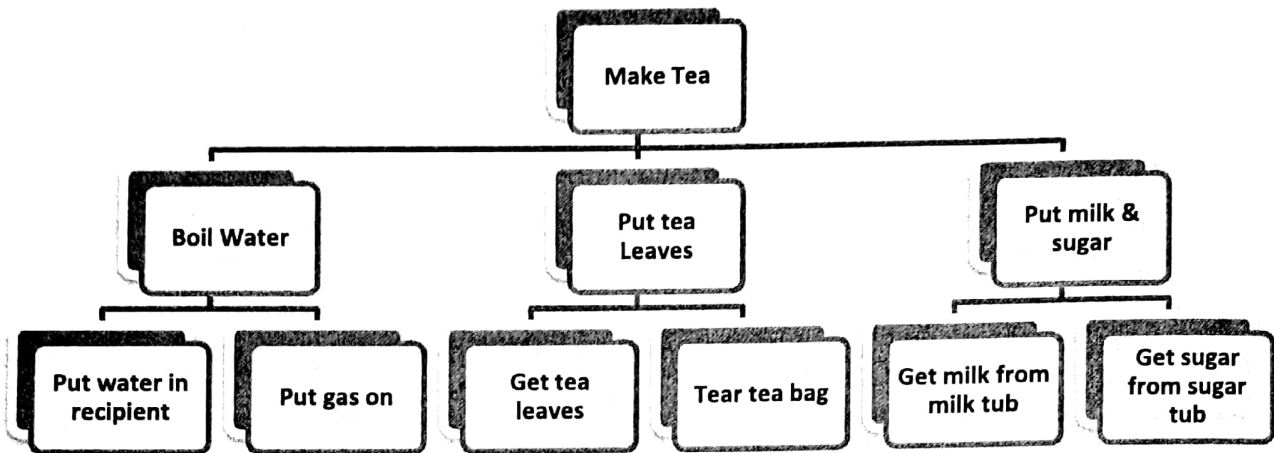
Description:

- It starts with the original problem at the highest level.
- The next, and subsequent, levels show how the problems in the previous levels are split up to form smaller, more manageable problems.
- This continues until each of the blocks in the lowest level is a self-contained, easily solvable problem.
- These individual problems can then be solved and combined according to the links that have been used.
- If the links between the blocks are used correctly, the result is a solution to the original problem.

Note: For the examples below, the diagrams refer to the stepwise refinements covered in previous section. The diagrams are read from top to bottom, left to right. These are only simple Jackson diagram with sequence statements. In Paper 4, Jackson diagrams are covered in detailed that include selection and iteration statement as well.

Example 1

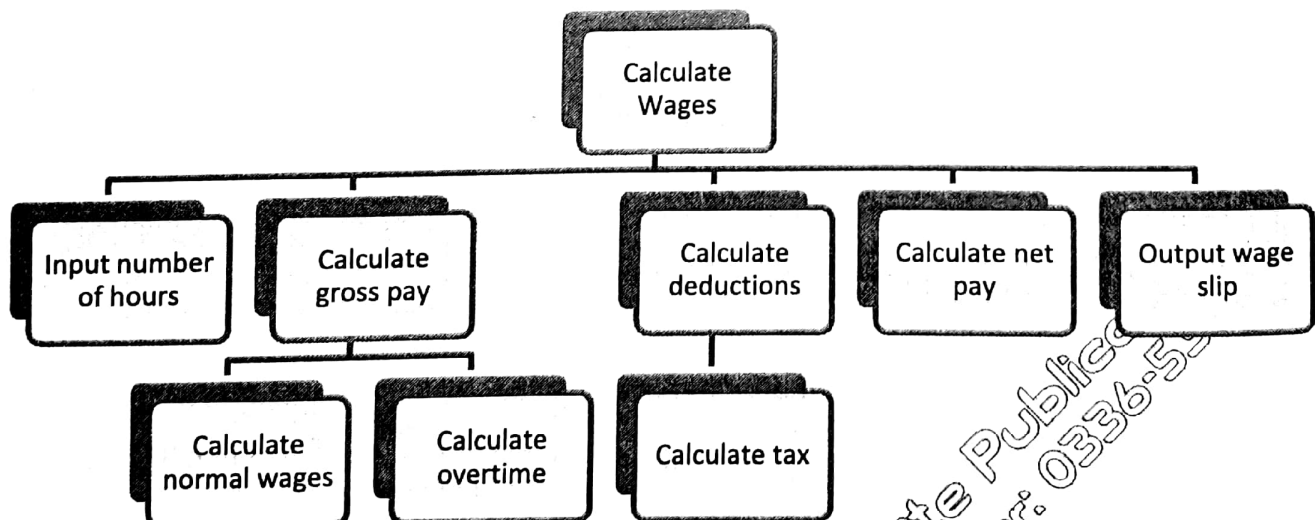
Structure diagram for making tea:

**Class activity 32**

Draw the top down design diagram for Example 2 in the previous section Stepwise refinement.

Example 2

Structure diagram for calculating wage of an hourly-paid worker:



These designs (modular design using numbered statements and structure diagrams) can be implemented as **subroutines**.

Class Activity 33

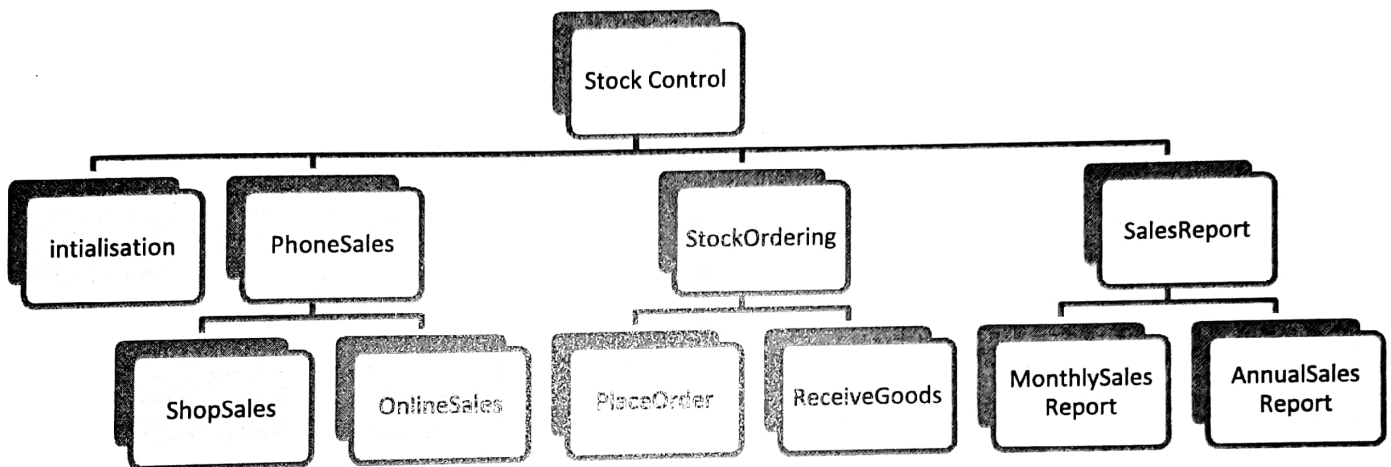
Draw the top down design diagram for Example 3 in the previous section Stepwise refinement.

Example 3

A person is designing a software solution for stock control in a mobile phone shop. He modularises the solution by splitting into 4 main areas: Initialisation, PhoneSales, StockOrdering, SalesReport.

- PhoneSales is made up of two modules, ShopSales and OnlineSales
- StockOrdering is made up of two modules, PlaceOrder and ReceiveGoods
- SalesReport is made up of two modules, MonthlySalesReport and AnnualSalesReport

This can be represented on a structure diagram as follows:

**Class Activity 34**

Fawad works for a software house which has been asked to design software for a car hire company, ExtremeCars.

Fawad decides on the main tasks:

- enter car details
 - input the car's identification details
 - input the hire rates for that car
- enter car hire details for
 - hirer
 - car
 - payment

Draw the structure diagram to represent the design of the software.

Read & Write Publications
For Books Order: 0336-5314141

Unit-5

Subroutines

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

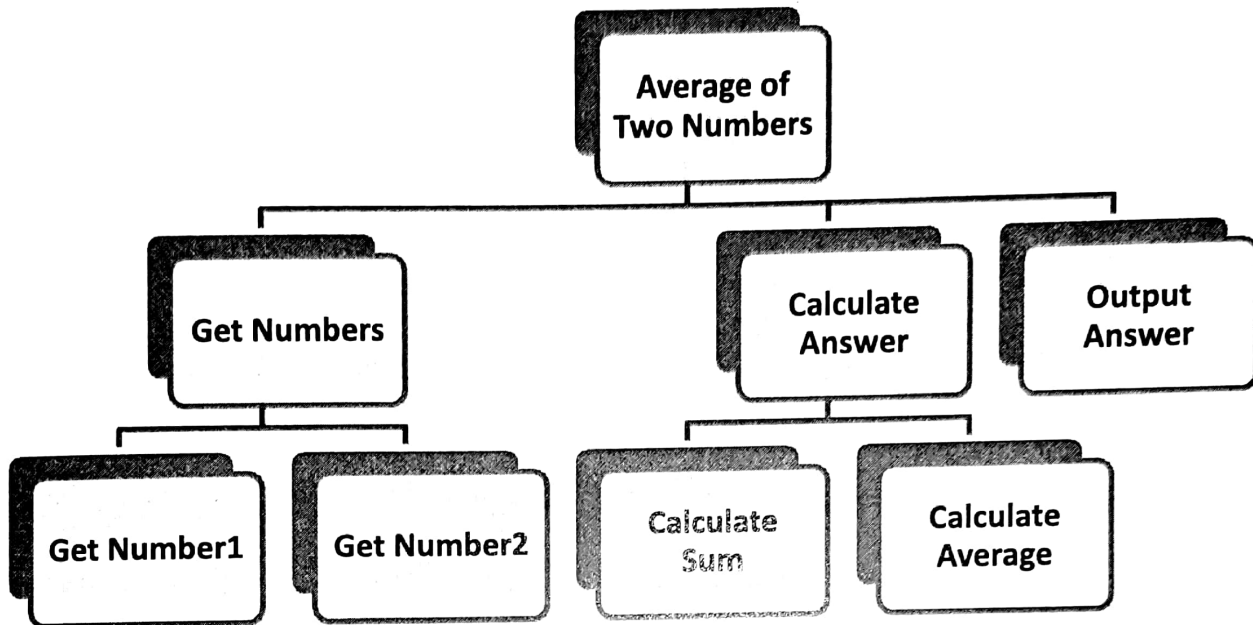
| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

- Procedure
- Function
- Parameters and local and global variables
- Samples illustrations + VB practical on procedures, function, parameter (known as formal parameters) and arguments (known as actual parameters)
- Further exercises on functions and procedures

Unit-5: Subroutines

Stepwise refinement and structure diagrams lead to a design from which the programmer can write modular code. Each module is a solution to an individual problem. The figure below depicts different modules in a top down design diagram.



Each of these modules can be written using a subroutine. A subroutine

- is a self-contained block of code that has an identifier name which must follow the same rules as for variables
- performs a task and is a program in its own right
- can be either procedures or functions

Subroutines are used to split a problem into simpler tasks in order to solve a complex problem. Instead of re-writing codes again and again, sub-routines can be used.

Advantages of sub-routines:

- Allows for code-reuse
- Allows structuring of programs
- Easily incorporate other peoples code (share codes with other programs)
- Allows for independent test of the rest of the code

5.1 Procedure

A procedure is a block of code that can receive values from another program and can return none, one or many values back to that program.

Pseudocode structure for procedure:

```
PROCEDURE <identifier>  
    <Statements>  
ENDPROCEDURE
```

Representing a procedure in VB:

```
Sub procoutput()  
    Console.WriteLine("Hello, I am in a procedure ")  
End Sub
```

Description:

- Sub indicates that this is a procedure
- ProcOutput is the procedure name

In short, a procedure is a block of program code statements designed to carry out a definable task.

5.2 Function

A function is given an identifier name (that can be defined as variable or constant etc.) and can also receive values from another program but returns only one value to the calling program.

Pseudocode procedure for function that has:

- (1) No parameters

```
FUNCTION <identifier> RETURNS <datatype>  
    <statement>
```

```
END FUNCTION
```

Representing a function with no parameters in VB:

```
Function FuncUserNum() As Integer  
    Console.WriteLine("Enter a number ")  
    Dim varEnterNum = Console.ReadLine  
    Return varEnterNum
```

```
End Function
```

Read & Write Publications
For Books Order: 0336-531411

Description:

- Function indicates this is a function
- FuncUserNum is the function name/identifier
- The function returns a value of type Integer

(2) One or more parameters

```
FUNCTION <identifier> «identifier> : <data type> RETURNS <data type>  
    <Statement>  
ENDFUNCTION
```

Representing a function with parameters in VB:

```
Function funcArea(ByVal radius As Double) As Double  
    Return 3.142 * radius * radius  
End Function
```

Description:

- Function indicates this is a function
- FuncArea is the function name/identifier
- radius is the parameter
- The function returns a value of type double

In short, a function is a block of program code statements that returns a single value to the program that called it.

Note: When to use function or procedure?

Using function

- In general if a single value is to be calculated, the simplest technique is to code a function.

Using procedure

- If there are no values to be returned, then a procedure should be used.
- If more than one value is to be returned, a procedure must be used.

5.3 Parameters and local and global variables

5.3.1 Module

Stepwise refinement leads to a design, from which the programmer can write modular code.

- Each module is a solution to an individual problem
- Each module has to interface with other modules.
- As long as the interfaces are clearly specified, each module could be given to a different programmer to code.

Interaction between modules

The programmers need to know their problem and how its solution must communicate with the other modules that make up the overall design. The programmers need to pass values to other modules and be able to accept values from other modules.

5.3.2 Passing parameters

Data can be input to a module (be it a function or a procedure) by means of parameters.

Example of a function in VB: Function to finds the perimeter of a rectangle given its length and breadth

Function FuncPeri(ByVal X As Integer, ByVal Y As Integer) As Integer

X = X * 2

y = Y * 2

FuncPeri = X + y 'or use return X + Y

End Function

In this function, X and Y are integers, the values of which are passed to the function before it can find the perimeter of the rectangle. The variables X and Y are called **formal parameters or parameters**.

Notice that the function follows the rules for any function:

- It has an identifier name, FuncPeri.
- It requires two parameters of the stated data type, integer.
- Its definition says it will return a value (an integer).
- It returns a single integer value to the calling program variable, varPerimeter, shown below.

To use this function, the program calls it and specifies the values for X and Y.

This can be done by means of a statement of the following forms:

Dim VarPerimeter As Integer

VarPerimeter = FuncPeri(2, 3)

In this statement that calls the function, the data values (i.e. 2, 3) inside the brackets are called **actual parameters or arguments**.

The whole program can be defined as follows:

Sub Main()

Dim VarPerimeter As Integer ' local variable in the main program

VarPerimeter = FuncPeri(2, 3) ' VarPerimeter stores the value returned by FuncPeri

Console.WriteLine("When length = 2 and breadth = 3 , the perimeter is " & VarPerimeter)

Console.ReadLine() ' prevents the console from closing

End Sub

Function FuncPeri(ByVal X As Integer, ByVal Y As Integer) As Integer

X = X * 2

y = Y * 2

FuncPeri = X + y 'or use return X + Y

End Function

The output is as follows:

file:///C:/Users/Fawad Khan/AppData/Local/Temporary Projects/Consol
When length = 2 and breadth = 3 , the perimeter is 10

5.3.3 Parameters V/S Arguments

When passing values between a main program and a subroutine, the values must be given variable names:

- When the main program calls a subroutine or a function, the values being passed to the subroutine or functions are called *arguments*.

Sub Main()

Dim num1, num2, sum As Integer

num1 = 3

num2 = 3

sum = funcsumNum(num1, num2)

Console.WriteLine("The sum is " & sum)

Console.ReadLine()

End Sub

- In the subroutine or function defined, the values are called *parameters*.

Function funcsumNum(ByVal a As Integer, ByVal b As Integer) As Integer

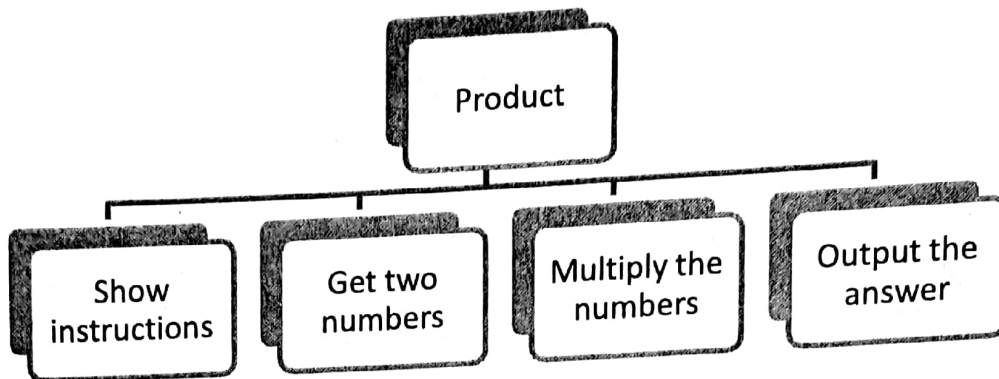
Return a + b

End Function

Read & Write Publications
For more Order: 0336-5314141

Example illustrating arguments and parameters

The structure diagram below shows the product of two numbers



The coded solution for the main program can be:

```

Sub Main()
    Dim num1, num2 As Double
    Dim product As Double
    Console.WriteLine("This program calculates the product of two number ")
    num1 = 5.5
    num2 = 4.0
    product = num1 * num2
    Console.WriteLine("The product of " & num1 & " and " & num2 & " is " & product)
End Sub
  
```

A procedure could be written to indicate the user the purpose of the program.

```

Sub explainPurposeofProgram()
    Console.WriteLine("This program calculates the product of two numbers ")
End Sub

Sub Main()
    Dim num1, num2 As Double
    Dim product As Double
    explainPurposeofProgram()
    Console.WriteLine("This program calculates the product of two number ")
    num1 = 5.5
    num2 = 4.0
    product = num1 * num2
    Console.WriteLine("The product of " & num1 & " and " & num2 & " is " & product)
End Sub
  
```

Another procedure could be written to output the numbers and their product.

```

Sub explainPurposeofProgram()
    Console.WriteLine("This program calculates the product of two numbers ")
End Sub
  
```



```
Sub OutputResult(ByVal FirstNumber As Double, ByVal SecondNumber As Double, ByVal result As Double)
```

```
    Console.WriteLine("The product of " & FirstNumber & " and " & SecondNumber & " is " & result)
```

```
End Sub
```

```
Sub Main()
```

```
    Dim num1, num2 As Double
```

```
    Dim product As Double
```

```
    explainPurposeofProgram()
```

```
    Console.WriteLine("This program calculates the product of two number ")
```

```
    num1 = 5.5
```

```
    num2 = 4.0
```

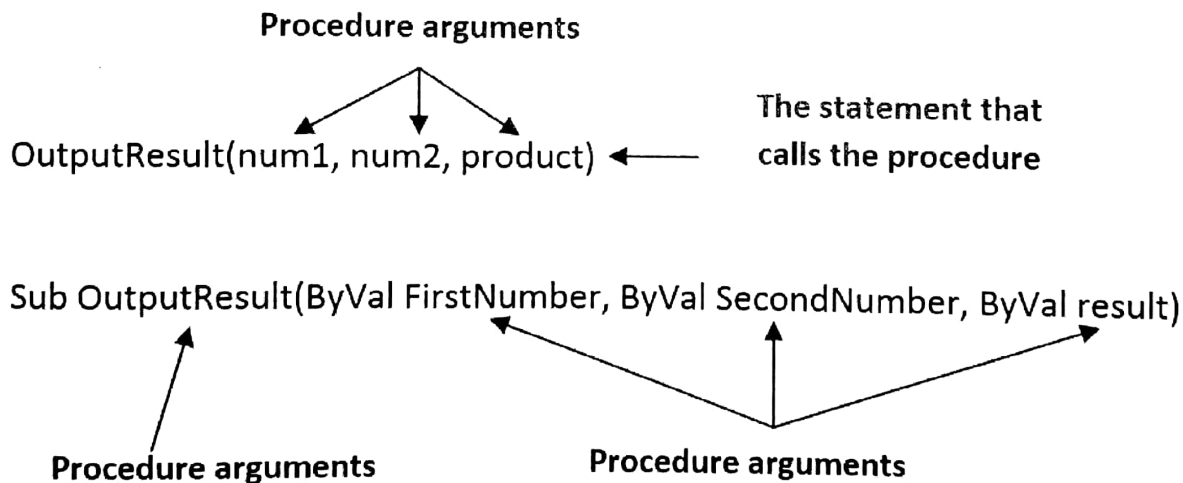
```
    product = num1 * num2
```

```
    OutputResult(num1, num2, product)
```

```
End Sub
```

Comments on the above program code:

The computer matches the arguments in line 16 to the parameters in line 25.



Passing parameters by value

The values of Num1, Num2 and Product are passed to the variables FirstNumber, SecondNumber and Result, respectively. This is called passing by value.

Memory for main

Program

Num1	→
Num2	→
product	→

Values	→
Copied	→
across	→

Memory for procedure

FirstNumber
SecondNumber
Result

Read & Write Publications
For Book's Order: 0336-531-1141

The memory for the main program can only be accessed by the main program (not by the procedure). Similarly, the memory for the procedure can only be accessed by the procedure (not by the main program).

Note: In VB, the syntax ByVal is put in front of the parameters to show passing parameters by value.

(i) Parameter passed by value

If a parameter is passed by value, the actual value is passed and the original value is retained when the subroutine is finished

Pseudocode representation of parameter passed by value for procedure:

PROCEDURE <identifier> (BYVALUE<identifier> : <data type>

<Statement>

END PROCEDURE

Note: A call is made to the procedure using CALL <identifier> ()

Example

This example illustrates that when parameters are passed by value, there is no change to original variable value in call by value whatever changes are made to local variable copy.

A Perimeter function with name **FuncPeriVal** is defined below:

'Perimeter Function

Function FuncPeri(ByVal X As Integer, ByVal Y As Integer) As Integer

X = X * 2

y = Y * 2

FuncPeri = X + y 'or use return X + Y

End Function

Notice that we have to use the ByVal keyword (highlighted above) to pass parameter X and Y by value in VB.

A program variable (highlighted below) is VarPerimeter calls the function FuncPeriVal:

Sub Main()

Dim varPerimeter, num1, num2 As Integer

num1 = 4

num2 = 6

Console.WriteLine("The initial value of num1 is " & num1)

Console.WriteLine("The initial value of num2 is " & num2)

varPerimeter = FuncPeriVal(num1, num2)

```

Console.WriteLine("parameter is " & varPerimeter)
Console.WriteLine("The final value of num1 is " & num1)
Console.WriteLine("The final value of num2 is " & num2)
End Sub

```

In this example of calling the Perimeter function FuncPeriVal, actual values (4 from num1 and 6 from num2) are passed to the function and stored in the variables X and Y.

The output is as follows:

Notice the initial value of num1 and num2 is the same when passing parameters by value. They have not changed even though they have been manipulated in the subroutine VarPerimeter. This is called passing parameters by value.

file:///C:/Users/Fawad Khan/AppData/Local/T

```

The initial value of num1 is 4
The initial value of num2 is 6
parameter is 20
The final value of num1 is 4
The final value of num2 is 6

```

(ii) Perimeter passed by reference/address

If a parameter is passed by reference, the address of the values is passed and any change to the value of the parameter in the subroutine changes the original value.

Pseudocode representation of parameter passed by reference for procedure:

PROCEDURE <identifier> (BYREF<identifier> : <data type>

<Statement>

ENDPROCEDURE

Note: A call is made to the procedure using CALL <identifier> 0

Example

This example illustrates that when a parameter is passed by reference, any change to local variable in call by reference changes original variable value.

A Perimeter function with name FuncPeriVal is defined below:

'Perimeter Function

Function FunPerival(ByVal x As Integer, ByVal y As Integer)

x = x * 2

y = y * 2

Return (x + y)

End Function

Read & Write Publications
For Books Order: 0332-531441

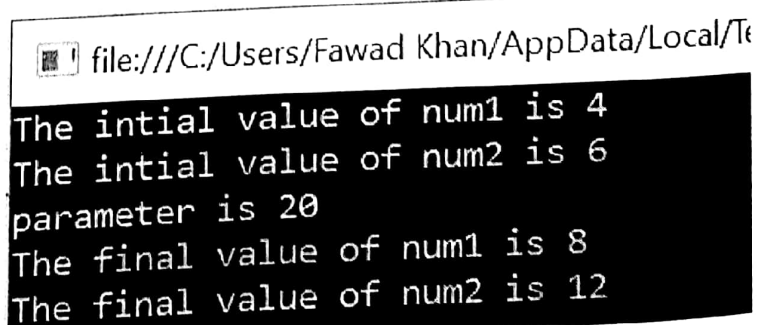
Notice that we have to use the ByRef keyword (highlighted above) to pass parameter X and Y by reference in VB.

A program variable (highlighted below) is VarPerimeter calls the function FuncPeriRef:

```
Sub Main()  
    Dim varPerimeter, num1, num2 As Integer  
    num1 = 4  
    num2 = 6  
    Console.WriteLine("The intial value of num1 is " & num1)  
    Console.WriteLine("The intial value of num2 is " & num2)  
    varPerimeter = FunPerival(num1, num2)  
    Console.WriteLine("parameter is " & varPerimeter)  
    Console.WriteLine("The final value of num1 is " & num1)  
    Console.WriteLine("The final value of num2 is " & num2)  
    Console.ReadLine()  
End Sub
```

In this example of calling the Perimeter function FuncPeriRef, the addresses of num1 and num2 are passed to the function, i.e., references to the values num1 and num2 are passed and the function then has to go and get the values

The output is as follows:



```
file:///C:/Users/Fawad Khan/AppData/Local/Temp/  
The intial value of num1 is 4  
The intial value of num2 is 6  
parameter is 20  
The final value of num1 is 8  
The final value of num2 is 12
```

Notice the initial value of num1 and num2 is not the same when passing parameters by reference. The values were changed after they have been manipulated in the subroutine VarPerimeter. This is called passing parameters by reference.

5.3.4 Local and global variables

1. Scope of variables

Variables can have different values in different parts of the program

Case 1: Using the same variable name in main program and functions

```

Module ModVarScope
    Sub Main ( )
        Dim a As Integer
        a = 2
        Console.WriteLine("value of a in main program" & a)
        ProcOutput ( )
        Console.ReadLine
    End Sub

    Sub ProcOutput ( )
        Dim a As Integer
        a = 3
        Console.WriteLine("Value of a in sub procedure" & a)
    End Sub
End Module

```

The output when the above program is run is shown below:

```

file:///C:/Users/Fawad Khan/AppData/Local/Te
Value of a in main program 2
Value of a in Sub procedure 3

```

Notice that "a" is a local variable and the "a" used in ProcOutput is a reference to a different address from the address referenced by "a" in the main program. Local variables only exist in the block in which they are declared. So the two variables called "a" that are declared in two different parts of an algorithm are two entirely separate variables.

Advantage of this approach:

- This is very helpful as it means that programmers writing different routines do not have to worry about the names of variables used by other programmers.

Case 2: Using a variable declared as global

In Visual Basic 10, a variable is made global by declaring it after the module identifier name but before the main procedure as shown below:

If we make "a" global in the example code, it becomes:

```

Module Module1
    Dim a As Integer 'can also use public a as interger
    Sub Main()
        a = 2
        Console.WriteLine("Value of a in main program " & a)
        ProcOutput ( )
        Console.ReadLine ( )
    End Sub

    Sub ProcOutput ( )
        Console.WriteLine("Value of aa in sub proceduro " & a)
        a = 3 ' value a was intially 2 and is replaced with value 3
        Console.WriteLine("Value of a in Sub procedure after assignment " & a)
    End Sub
End Module

```

The output when the above program code is run is shown below:

```
file:///C:/Users/Fawad Khan/AppData/Local/Temporary Project
Value of a in main program 2
Value of aa in sub procedure 2
Value of a in Sub procedure after assignment 3
```

Observation:

- The value of C, when changed in the function Perimeter, is also changed in the calling routine. In fact, it is changed throughout the program.

Advantage of this approach:

- Sometimes it is useful to be able to use the same variable in many parts of a program.

5.4 Samples illustrations + VB practical on procedures, function, parameter (known as formal parameters) and arguments (known as actual parameters)

Example 1

Below is a program that declares a variable Name with initial value Fawad Khan and another variable Pie with initial value 3.142

```
Sub Main()
    Dim name As String
    Const pie As Decimal = 3.142
    name = "Fawad Khan"
End Sub
```

Question - Write a procedure to output the name. The solution is shown below:

```
Sub Main()
    Dim name As String
    Const pie As Decimal = 3.142
    name = "Fawad Khan"
    proOutName(name)
End Sub

Sub proOutName(ByVal paraname As String)
    Console.WriteLine(paraname)
End Sub
```

Note: You can also use the syntax: Call proOutName (name) to call the procedure

Class Activity 35

Below is a VB program code that declares two variables (Mark1 and Mark2) that stores initial values 40 and 90 respectively.

```
Sub Main()  
    Dim mark1, mark2 As Integer  
    mark1 = 40  
    mark2 = 90  
End Sub
```

Write a procedure to output the two marks.

Example 2

Below is a program that calculates the sum of two numbers input by a user.

```
Sub Main()  
    Dim num1, num2 As Integer  
    num1 = Console.ReadLine  
    num2 = Console.ReadLine  
    Console.WriteLine(num1 + num2)  
End Sub
```

Question - Write a function that calculates the sum of the two numbers and returns to the main program where it is output. The solution is shown below:

```
Sub Main()  
    Dim num1, num2 As Integer  
    Dim total As Integer  
    num1 = Console.ReadLine  
    num2 = Console.ReadLine  
    total = funcGetSum(num1, num2)  
End Sub  
Function funcGetSum(ByVal paranum1 As Integer, ByVal paranum2 As Integer)  
    Return paranum1 + paranum2  
End Function
```

Class Activity 36

Below is the VB program code that will allow a user to calculate volume of a cylinder using formula. $V = \pi \times \text{radius} \times \text{radius} \times \text{height}$.

```
Sub Main()  
    Dim radius As Double  
    Dim height As Double
```

Read & Write
For Books Order


```

Dim volume As Double
Const pie As Double = 3.142
radius = Console.ReadLine
height = Console.ReadLine
volume = pie * radius * radius * height
Console.WriteLine(volume)

End Sub

```

Write a function that calculates the volume and returns it to the main program for output.

Example 3

Below is a main program that takes a mark from a user and output pass if the mark is above 59.

```

Dim mark As Integer

Console.WriteLine("Please enter a mark ")
mark = Console.ReadLine

If mark > 59 Then
    Console.WriteLine("You have passed ")
End If

```

Question -Write (i) a function that check if the mark is above 39 and return "pass" to the main program if so. (ii) a procedure to print the value return from the function. The solution is shown below:

```

Sub Main()
    Dim mark As Integer
    Dim comment As String
    Console.WriteLine("Please enter a mark ")
    mark = Console.ReadLine
    comment = FunccheckPass(mark)
    Call ProOutComment(comment)
End Sub

```

```

Function FunccheckPass(ByVal paramarks As Integer) As String
    Dim checkpass As String = "Not Pass"
    If paramarks > 39 Then
        checkpass = "pass"
    End If
    Return checkpass
End Function

```

```

Sub ProOutComment(ByVal paracomment As String)
    Console.WriteLine(paracomment)
End Sub

```

Class Activity 37

Below is a main program that takes a mark from a user and output fail if the mark is below 60

```
Dim mark As Integer
```

```
    Console.WriteLine("Please enter a mark ")
```

```
    mark = Console.ReadLine
```

```
    If mark < 60 Then
```

```
        Console.WriteLine("you have failed ")
```

```
    End If
```

Write (i) A function that check if the mark is below 40 and return "fail" to the main program if so.

(ii) A procedure to print the value return from the function.

Example 4

Below is a program that displays the square of each element in an array.

```
Dim ArrayInput() = {1, 2, 3}
```

```
For i = 0 To 2
```

```
    Console.WriteLine(ArrayInput(i) * 2)
```

```
Next
```

Question - Write a function that takes as parameter the array in the main program and returns another array containing the square of each element of the array taken as parameter.

Solution:

```
Sub Main()
```

```
    Dim ArrayInput() As Integer = {1, 2, 3}
```

```
    Dim ArrayOutput(2) As Integer
```

```
    ArrayOutput = funRetArray(ArrayInput)
```

```
    For i = 0 To 2
```

```
        Console.WriteLine(ArrayOutput(i))
```

```
    Next
```

```
End Sub
```

```
Function funRetArray(ByVal arrayIn() As Integer) As Integer()
```

```
    Dim arrayFill(2) As Integer
```

```
    For i = 0 To 2
```

```
        arrayFill(i) = arrayIn(i) * 2
```

```
    Next
```

```
    Return arrayFill
```

```
End Function
```

For

Question - Write a procedure to output the values of the array returned by the function.

Solution:

```

Sub Main()
    Dim ArrayInput() As Integer = {1, 2, 3}
    Dim ArrayOutput(2) As Integer
    ArrayOutput = funRetArray(ArrayInput)
    ProPrintArrayVal(ArrayOutput)
    Console.ReadKey()
End Sub

Function funRetArray(ByVal arrayIn() As Integer) As Integer()
    Dim arrayFill(2) As Integer
    For i = 0 To 2
        arrayFill(i) = arrayIn(i) * 2
    Next
    Return arrayFill
End Function

Sub ProPrintArrayVal(ByVal arrayOut() As Integer)
    For i = 0 To 2
        Console.WriteLine(arrayOut(i))
    Next
End Sub

```

Class Activity 38:

Write a function that returns the cube of each element of the array in the main program. Then, write a procedure to output the values of the returned array.

5.5 Further exercises on functions and procedures

- Below is a program that takes an age from a user, first check if the age is valid (must be between 0 and 100), if so, print young if the age is below 40 or print old if the age is above 39.

```
Dim age As Integer
```

```
Console.WriteLine("Enter an age ")
```

```
age = Console.ReadLine
```

```
If age > 0 And age <= 100 Then
```

```
    If age < 40 Then
```

```
        Console.WriteLine("You are young ")
```

```
    Else
```

```
        Console.WriteLine("You are old ")
```

```
    End If
```

```
Else
```

```
    Console.WriteLine("Oh.... an alien in the vicinity....")
```

```
End If
```

Read & Write Publications
 For Books Order: 0336-531441

End Sub

- (i) Write a function that checks if the age is greater than 0 and less than 100, and return the appropriate comment to the main program.

- (ii) Write a procedure to output the comment returned by the function in part (i)

2 Below is a program that takes a temperature, print "room temperature" if the temperature is from 1 to 37, prints "extreme temperatures" if the temperature is from 38 to 100 or print "freezing temperatures" if temperature is from -15 to 0 degrees Celsius.

Dim temp As Integer

Console.WriteLine("Enter a Temperature ")

temp = Console.ReadLine

Select Case temp

Case -15 To 0

Console.WriteLine("Freeszing temperature ")

Case 1 To 37

Console.WriteLine("Room Temperature ")

Case 38 To 100

Console.WriteLine("Extreme temperature ")

End Select

- (i) Write a function that takes the temperature from the user as parameter and return the comment for the type of temperature to the main program.

- (ii) Write a procedure to output the comment returned by the function in part (i)

3. Below is a program that takes 10 numbers from a user and output their total.

Dim number As Integer

Dim total As Integer

total = 0

For count = 1 To 10

Console.WriteLine("Enter a Number ")

number = Console.ReadLine

total = total + number

Next

Console.WriteLine("The total of these numbers is: " & total)

- (i) Write a function for the For Loop which will return the total to the main program!

- (ii) Write a procedure to output the total returned by the function in part (i)

4. Below is a program that will allow a user to calculate area of a circle as long as the radius being entered is positive.

Dim area As Decimal

Const pie As Decimal = 3.142

Dim radius As Integer = 1

While radius > 0

Console.WriteLine("Enter a radius ")

radius = Console.ReadLine

area = pie * radius * radius

Console.WriteLine("Area is " & area)

End While

- (i) Write a function for the while loop which will return the area to the main program!
- (ii) Write a procedure to output the area returned by the function in part (i)

Can we create a function in VB 10 that returns more than one values?

Read & Write Publications
For Books Order: 0336-531411

Unit-6

Structure Chart

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

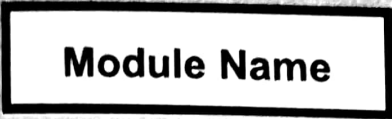
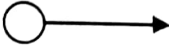

Topics

- Symbols used in structure chart
- Structure chart representation of sequence code
- Structure chart representation for selection code
- Structure chart representation of iteration code
- Combination of sequence, selection and iteration code
- Further exercises on structure chart

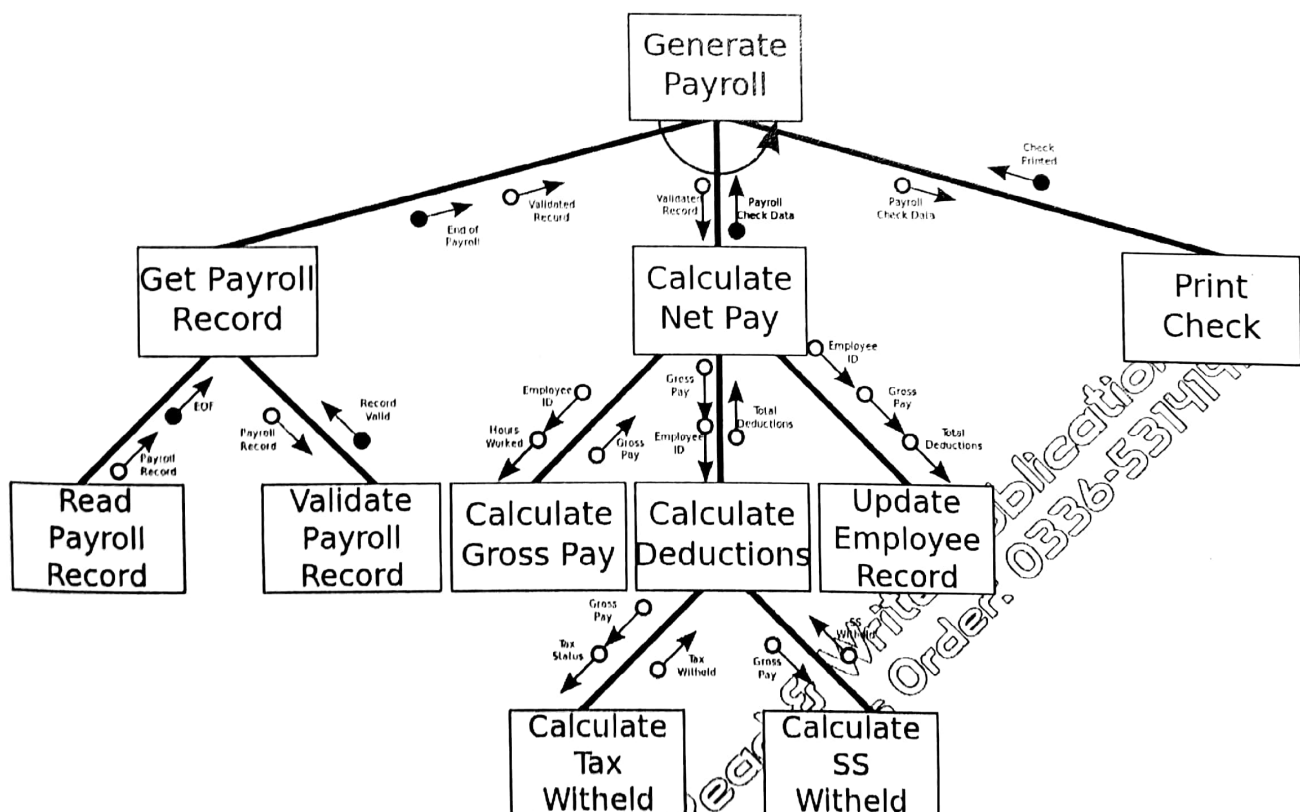
Unit-6: Structure Chart

A structure chart in software engineering is a chart which shows the breakdown of a system to its lowest manageable parts. It represents a top-down design where a problem (the program) is broken into its components.

6.1 Symbols used in structure chart

Symbol	Name	Meaning
	Process	Each Box represents a programming module, this Module Process might be something that Name calculates the average of some figures, or prints out some pay slips
	Data Couple	Data being passed from module to module that needs to be processed.
	Flag	Check data sent to process to stop or start processes. For example to say that an End of a File is reached, or to say whether data sent was in the correct format

Sample structure chart:



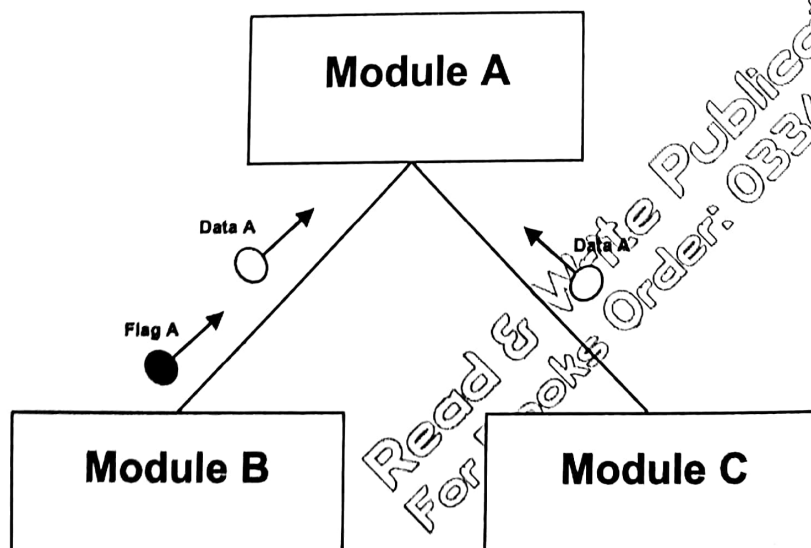
The figure above depicts the symbol set used for structure charts.

Description of sample structure chart:

- Structure chart modules are depicted by named rectangles. Modules are factored, from the top down, into sub-modules. The highest level module is referred to as the system or root module. It serves to coordinate or "boss" the modules appearing directly beneath it. In turn, those modules may serve to coordinate those modules appearing immediately below them.
- Structure chart modules are presumed to execute in a top-to-bottom, left-to-right sequence. The line connecting two modules represents a normal call. For example, SYSTEM MODULE calls MODULE B.
- An arc shaped arrow located across a line (representing a module call) means that the module makes iterative calls. Thus, SYSTEM MODULE calls MODULE A to be performed N number of times, or until some condition is met.
- A diamond symbol located at the bottom of a module means that the module calls one and only one of the other lower modules that are connected to the diamond. Thus, MODULE A can call MODULE C or MODULE D. Notice however that MODULE B can call MODULE G and it can call *either* MODULE E or MODULE F. This diagram construct is referred to as a *transaction center*.
- Program modules communicate with each other through passing of data. Data being passed is represented by named arrows with a small circle on one end. The direction of the arrow is significant. Note that DATA A is being passed "up" from MODULE C to its parent, MODULE A. The downward direction of the arrow for DATA B implies that SYSTEM MODULE is passing it to MODULE B.
- Programs may also communicate with each other through passing of messages or control parameters, called flags. Control flags are depicted by an arrow with a darkened circle on one end. As with data, the direction of the arrow implies the source and receiving modules.

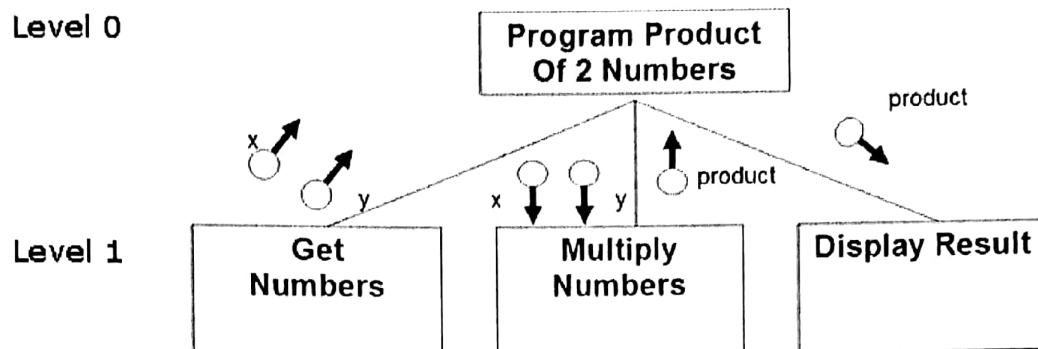
6.2 Structure chart representation of sequence code

A sequence in a Structure Chart means that modules are executed one after the other. For example, in the figure below, Module B is executed followed by Module C



Example 1

The Diagram below shows a structure chart for calculating the product of two numbers:

**Comments:**

- The Get Numbers module gets two numbers from a user
- The Multiply Numbers module uses the two numbers from the user and returns their products
- The Display Result module uses the product and prints it to the user.

The program code for this structure chart can be:

```

Module Module1
    Dim x As Integer
    Dim y As Integer
    Sub Main()
        Dim product As Double
        Getnumber()
        product = MultiplyNumbers(x, y)
        DisplayResult(product)
        Console.ReadKey()
    End Sub
    Sub Getnumber()
        x = Console.ReadLine
        y = Console.ReadLine
    End Sub
    Function MultiplyNumbers(ByVal paraX As Double, ByVal paraY As Double) As Double
        Return paraX * paraY
    End Function
    Sub DisplayResult(ByVal paraProduct As Double)
        Console.WriteLine(paraProduct)
    End Sub

```

Read & Write
For Books Order

Example 2

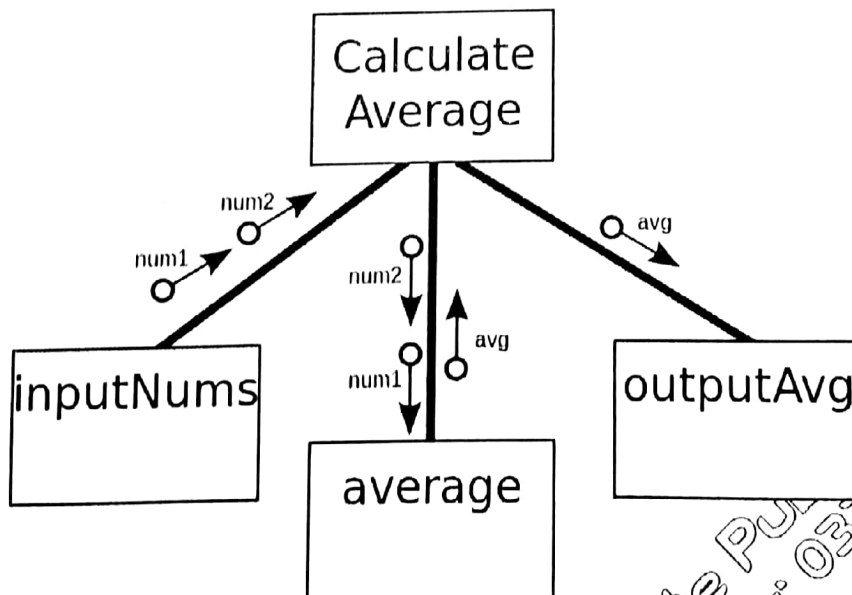
Consider the following program code that calculates the average of two digits.

```

Module Fawad
|
| Dim num1, num2 As Integer
|
| Sub Main()
|   Dim avg As Integer
|   inputNums()
|   avg = average(num1, num2)
|   outputAvg(avg)
| End Sub
|
| Function average(ByVal a, ByVal b)
|   Return (a + b) / 2
| End Function
|
| Sub inputNums()
|   num1 = Console.ReadLine()
|   num2 = Console.ReadLine()
| End Sub
|
| Sub outputAvg(ByVal x)
|   Console.WriteLine("average = " & x)
| End Sub

```

The corresponding structure chart for the above code is:

**Comments:**

- For the procedure **inputNums**, **num1** and **num2** are shown to return values to the main program as they are declared as global variable and are processed again in the next lines which follow.
- Instead of writing **CalculateAverage**, you could have written **main** which is more appropriate.
- You could also have used **a** instead of **num 1** and **b** instead of **num2** to show data being passed between the main module and the average module.

Class activity 39:

Create structure charts for the following program code.

Module Fawad

Sub Main()

Dim num1 As Integer

Dim num2 As Integer

Dim avg As Integer

sayHello()

num1 = 34

num2 = 89

avg = average(num1, num2)

End Sub

Function average(ByVal a, ByVal b)

Return (a + b) / 2

End Function

Sub sayHello()

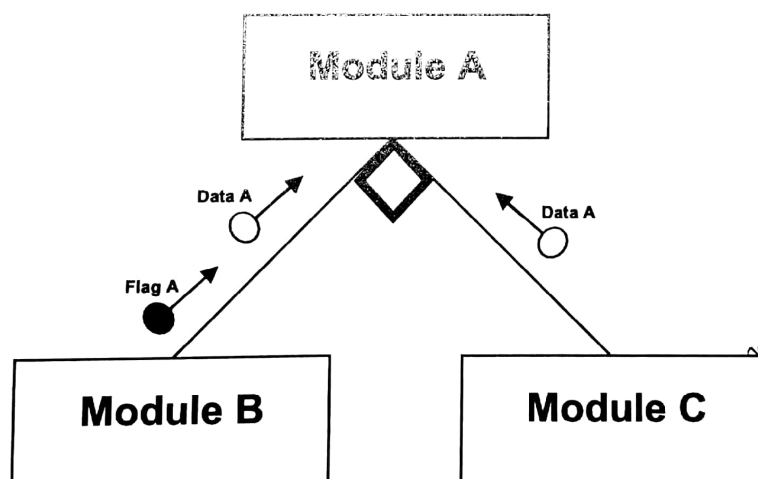
console.writeline("hello")

End Sub

End Module

6.3 Structure chart representation for selection code

A selection in a Structure Chart is determined by the diamond symbol. This means a condition will be checked and depending on the result, different modules will be executed. For example, in the figure below, either Module C will be executed or Module D.



Example 1

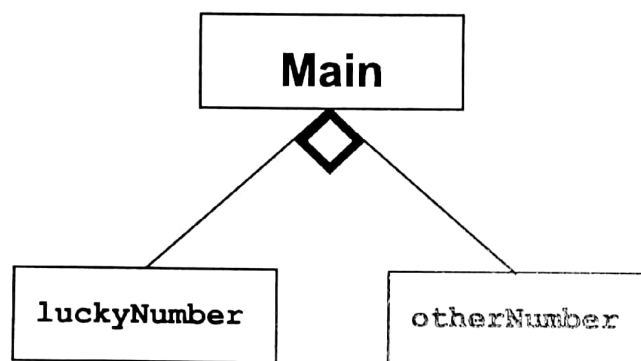
Consider the following code that checks for a lucky number. Draw its structure chart.

Read & Write Communications
For Books Order: 336-5314141

```

Module Fawad
  Sub Main()
    Dim num1 As Integer
    num1 = Console.ReadLine
    If num1 = 7 Then
      luckyNumber()
    Else
      otherNumber()
    End If
  End Sub
  Sub luckyNumber()
    Console.WriteLine("This is your lucky number ")
  End Sub
  Sub otherNumber()
    Console.WriteLine("Luck is not on your side ")
  End Sub
End Module

```



Class activity 40:

Consider the following program code that displays different grade depending on the mark entered. Draw its structure chart

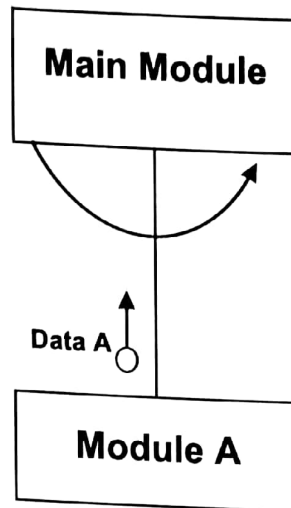
```

Module Fawad
  Sub Main()
    Dim mark As Integer
    If mark >= 0 And mark <= 60 Then
      gradeFail()
    Else
      If mark > 60 And mark <= 100 Then
        gradePass()
      End If
    End If
  End Sub
  Sub gradeFail()
    Console.WriteLine("You have failed ")
  End Sub
  Sub gradePass()
    Console.WriteLine("You have passed ")
  End Sub
End Module

```

6.4 Structure chart representation of Iteration code

Using the semi-circular arrow we can represent iteration in Structure Charts. The arrow encompasses a link to a module, implying that module is executed multiple times. For example, in the figure below, Module A is iterated several times

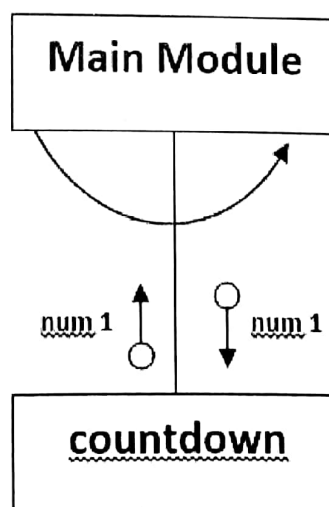
**Example 1**

Consider the following program code:

```

Module Fawad
  Sub Main()
    Dim num1 As Integer
    Console.WriteLine("Enter a positive number ")
    num1 = Console.ReadLine()
    Do While num1 > 0
      num1 = countdown(num1)
    Loop
  End Sub
Function countdown(ByVal n As Integer)
  Return n - 1
End Function
End Module
  
```

Its structure chart is drawn below:

**Class activity 41**

Draw the structure chart for the following program code:

Read & Write Publications
For Books Order: 0336-531441

Module Fawad**Sub Main()**

```

    Dim radius As Double
    Dim area As Double
    radius = Console.ReadLine
    Do While radius > 0
        area = calcArea(radius)
        printArea(area)
        radius = Console.ReadLine
    
```

```

    Loop

```

End Sub

```

Function calcArea(ByVal rad As Double) As Double

```

```

    Return 3.142 * rad * rad

```

End Function

```

Sub printArea(ByVal area As Double)

```

```

    Console.WriteLine(area)

```

End Sub**End Module****6.5 Combination of sequence, selection and iteration code**

Create the structure chart for the following program code.

Module Fawad**Sub main()**

```

    Dim num1, count, total As Integer
    num1 = startMsg()
    count = 0
    total = 0
    Do While num1 > 0
        checkNumber(count, total, num1)
        num1 = num1 - 1
    
```

```

    Loop

```

```

    endMsg(count)

```

End Sub**Function startMsg()**

```

    Console.WriteLine("program started, enter your number")
    Return Console.ReadLine()

```

End Function

```

Sub checkNumber(ByRef c As Integer, ByRef t As Integer, ByVal n As Integer)

```

```

    If n Mod 3 = 0 Then

```

```

        c = divBy3(c)

```

```

    Else

```

```

        t = add(n, t)

```

```

    End If

```

End Sub

```

Function divBy3(ByVal x As Integer) As Integer

```

```

    Return x + 1

```

End Function

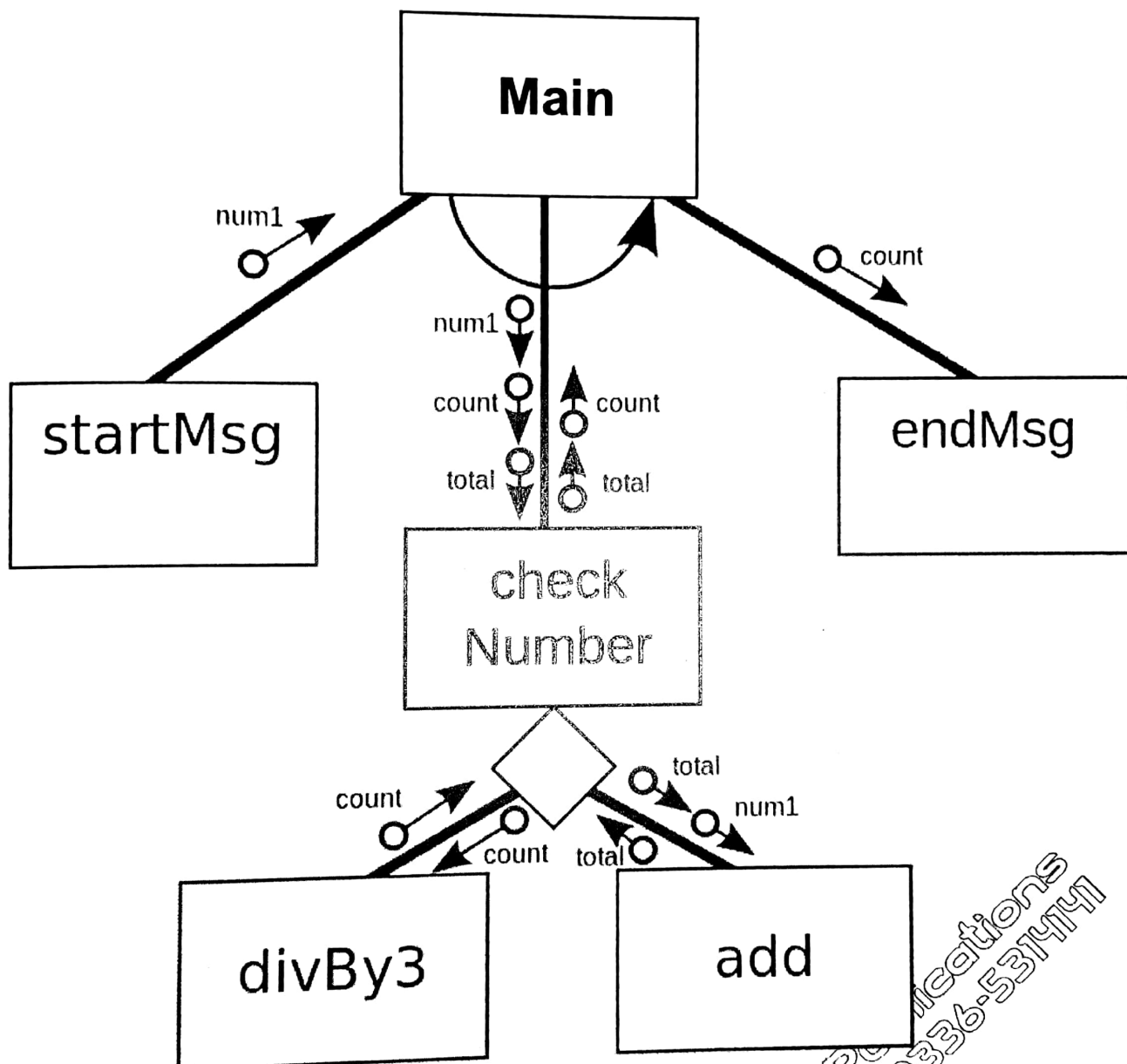
Read
For Book

```

Function add(ByVal n As Integer, ByVal t As Integer) As Integer
    Return n + t
End Function
Sub endMsg(ByVal n As Integer)
    Console.WriteLine("number of threes : " & n)
End Sub
End Module

```

Solution:



Comments:

- For the function **divBy3**, instead of **count**, **c** could have been used as label for showing data transfer.
- For the function **add**, instead of **total** and **num1**, **v** and **n** could have been used as label for showing data transfer.
- Notice that for the procedure **checkNumber**, **count** and **total** are returned to the main module because their values change since they are passed by reference.

6.6 Further exercises on structure chart

1. Create the structure chart for the following program code:

```

Module Fawad
    Sub Main()
        Dim mark As Integer
        Dim comment As String
        Console.WriteLine("Please enter a mark ")
        mark = Console.ReadLine
        comment = FunCheckPass(mark)
        Call proOutComment(comment)
    End Sub
    Function FunCheckPass(ByVal paramark As Integer) As String
        Dim checkpass As String = "Not pass"
        If paramark > 60 Then
            checkpass = "pass"
        End If
        Return checkpass
    End Function
    Sub proOutComment(ByVal paraComment As String)
        Console.WriteLine(paraComment)
    End Sub
End Module

```

2. Create the structure chart for the following program code:

```

Sub Main()
    Dim ArrayInput() As Integer = {1, 2, 3}
    Dim ArrayOutput(2) As Integer
    ArrayOutput = funRetArray(ArrayInput)
    ProPrintArrayVal(ArrayOutput)
    Console.ReadKey()
End Sub
Function funRetArray(ByVal arrayIn() As Integer) As Integer()
    Dim arrayFill(2) As Integer
    For i = 0 To 2
        arrayFill(i) = arrayIn(i) * 2
    Next
    Return arrayFill
End Function
Sub ProPrintArrayVal(ByVal arrayOut() As Integer)
    For i = 0 To 2
        Console.WriteLine(arrayOut(i))
    Next
End Sub

```

Unit-7

Topics

- Maths functions
- Random Number generation
- String handling functions

VB in Built Functions

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Unit-7: VB in Built Functions

These programming routines come in-built into most common programming languages. They are very useful and save time and effort in writing code to perform common tasks.

7.1 Maths functions

The methods of the **System.Math** class provide trigonometric, logarithmic, and other common mathematical functions. To use these functions:

1. Without qualification, import the System.Math namespace into your project by adding the following code to the top of your source file:

```
Imports System.Math
Module module1
    Sub Main()
        Console.WriteLine(Round(1.94, 1)) ' print 1.9

        Console.ReadKey()
    End Sub
End Module
```

2. Or the use functions (also known as methods) directly by putting math.x() where x is the function name.

The main mathematical functions that are commonly used are:

- (i) **Abs** - Returns the absolute value of a number

```
Imports System.Math
Module module1
    Sub Main()
        'math.abs() returns the number, i.e the positive number
        Console.WriteLine(Math.Abs(34.78)) ' prints 34.78
        Console.WriteLine(Math.Abs(-34.78)) ' prints 34.78

        Console.ReadKey()
    End Sub
End Module
```

file:///C:/Users/Fawad Khan/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/ConsoleApplication1.Ex

34.78

34.78

- (ii) **Pow** - Returns a specified number raised to the specified power.

```
Imports System.Math
Module module1
    Sub Main()
        Console.WriteLine(Math.Pow(2, 3)) ' prints 8
        Console.ReadKey()
    End Sub
End Module
```

file:///C:/Users/Fawad Khan/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Ct

8

(iii) **Round** - The round function is used to round numbers to a limited number of decimal places using the Math.Round() function.

```
'math.round(number,number of decimal places )
Console.WriteLine(Math.Round(1.94, 0)) ' prints 2
Console.WriteLine(Math.Round(1.95, 0)) ' prints 2
Console.WriteLine(Math.Round(1.96, 0)) ' prints 2
Console.WriteLine("*****")
Console.WriteLine(Math.Round(1.45, 0)) ' prints 1
Console.WriteLine(Math.Round(1.55, 0)) ' prints 2 ' 0.5 round up
Console.WriteLine(Math.Round(1.65, 0)) ' prints 2
Console.WriteLine("*****")
Console.WriteLine(Math.Round(1.45, 1)) ' prints 1.4
Console.WriteLine(Math.Round(1.46, 1)) ' prints 1.5
Console.WriteLine(Math.Round(1.47, 1)) ' prints 1.5
Console.WriteLine("*****")
Console.WriteLine(Math.Round(1.94, 1)) ' prints 1.9
Console.WriteLine(Math.Round(1.95, 1)) ' prints 2
Console.WriteLine(Math.Round(1.96, 1)) ' prints 2
Console.WriteLine("*****")
Console.WriteLine(Math.Round(1.9445, 2)) ' prints 1.94
Console.WriteLine(Math.Round(1.9455, 2)) ' prints 1.95
Console.WriteLine(Math.Round(1.9456, 2)) ' prints 1.95
```

(iv) **Sqrt** - Returns the square root of a specified number.

```
Imports System.Math
```

```
Module module1
```

```
Sub Main()
```

```
Console.WriteLine(Math.Sqrt(9)) ' prints 3
```

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```

```
Me:///C:/Users/Fawad Khan/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE
```

(v) **Truncate** - The truncate function returns the integer part of a number regardless of the decimal places.

```
Imports System.Math
```

```
Module module1
```

```
Sub Main()
```

```
'math.truncate(decimal_number)
```

```
Console.WriteLine(Math.Truncate(4.56)) ' prints 4
```

```
Console.WriteLine(Math.Truncate(54.565)) ' prints 54
```

```
Console.WriteLine(Math.Truncate(5455.565)) ' prints 5455
```

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```

```
Me:///C:/Users/Fawad Khan/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE
```

```
4
54
5455
```

This is particularly useful when you are trying to perform DIV in modular arithmetic.

(vi) **Floor** - This function returns the largest integer that's less than or equal to the specified Decimal or Double number.

```
Imports System.Math
```

```
Module module1
```

```
Sub Main()
```

```
'math.truncate(decimal_number)
```

```
Console.WriteLine(Math.Floor(4.56)) ' prints 4
```

```
Console.WriteLine(Math.Floor(4.88)) ' prints 4
```

```
Console.WriteLine(Math.Floor(5.99)) ' prints 5
```

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```

```
file:///C:/Users/Fawad Khan/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/
```

```
4  
+  
5
```

(vii) **Ceiling** - This function returns the smallest integral value that's greater than or equal to the specified Decimal or Double.

```
Module module1
```

```
Sub Main()
```

```
'math.truncate(decimal_number)
```

```
Console.WriteLine(Math.Ceiling(4.44)) ' prints 5
```

```
Console.WriteLine(Math.Ceiling(4.88)) ' prints 5
```

```
Console.WriteLine(Math.Ceiling(5.99)) ' prints 6
```

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```

Note: You can also use:

- Exp - Returns e (the base of natural logarithms) raised to the specified power.
- Log - Returns the natural (base e) logarithm of a specified number or the logarithm of a specified number in a specified base.
- Pow - Returns a specified number raised to the specified power.
- Sin - Returns the sin of the specified angle.
- Tan - Returns the tangent of the specified angle.
- Cos - Returns the cosine of the specified angle.

7.2 Random Number generation

In some situations we have to generate random numbers for different purposes.

Visual Basic .NET supports two ways to generate random numbers:

- Random-number generator (Rnd())
- A class for generating random numbers(Random())

7.2.1 Random-number generator of VB6 (Using Rnd() function

A random number generator is an object that produces a sequence of pseudo-random values.

Rnd()

This function returns a random value which is less than 1, but greater than or equal to zero. (i.e. between 0 and 0.99999999 inclusive)

```
Module module1
    Sub Main()
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.ReadKey()
    End Sub
End Module
```

file:///C:/Users/Fawad Khan/AppData/Local/Temporary P
0.7055475
0.533424
0.5795186
0.2895625

Note: The sequence of random numbers produced by VB is always the same. Consider an application that displays three random numbers. If you stop and re-run the application, the same three numbers will be displayed. This is not a bug. It's a feature in VB that allows you to debug applications that use random numbers. If the sequence were different, you wouldn't be able to re-create the problem.

To change this default behavior, call the Randomize statement at the beginning of your code. This statement will initialize the random-number generator based on the value of the computer's Timer, and the sequences of random numbers will be different every time you run the application.

Therefore re-running the code below again produces the same result.

```
Module module1
    Sub Main()
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.ReadKey()
    End Sub
End Module
```

file:///C:/Users/Fawad Khan/AppData/Local/Temporary
0.7055475
0.533424
0.5795186
0.2895625

To prevent this, call the Randomize function to have different random number:

Read & Write Publications
For Books Order: 0336-531441

```

Module module1
    Sub Main()
        Randomize() ' call this function to have different number
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.WriteLine(Rnd())
        Console.ReadKey()
    End Sub
End Module

```

file:///C:/Users/Fawad Khan/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/ConsoleApplication1.1

0.2481958
0.3273541
0.9321035
0.02868295

In most cases, you don't need a random number between 0 and 1 (where 0 only is inclusive), but between two other integer values. To produce random integers in a given range, use the following formula:

$$\text{randomValue} = \text{INT} ((\text{upperbound} - \text{lowerbound} + 1) * \text{Rnd}()) + \text{lowerbound}$$

Where

- upperbound is the highest value in the range
- lowerbound is the lowest value in the range

Example 1: Generate random numbers from 1 to 40

```

Module module1
    Sub Main()
        Randomize()
        Console.WriteLine(Int(Rnd() * (40 - 1 + 1)) + 1)
        Console.ReadKey()
    End Sub
End Module

```

Class activity 42:

Create a program that generates random numbers from 1 to 50.

Example 2: Generate random numbers from 5 to 40

```

Module module1
    Sub Main()
        Randomize()
        Console.WriteLine(Int(Rnd() * (40 - 5 + 1)) + 5)
        Console.ReadKey()
    End Sub
End Module

```

Class activity 43

Create a program that generates random numbers from 10 to 50.

Publications
0336-531441

Example 3: Generate random numbers in range 5 excluding 5

```

Sub Main()
    Randomize() ' call this function to have different
    'each time you re-run the program
    Console.WriteLine(Int(Rnd() * 5))
    Console.ReadKey()
End Sub

```

The "* 5" says how many numbers it will go by. The output of that will either be 0, 1, 2, 3, or 4; i.e. 5 different numbers. **Notice that 5 is not generated here.**

Class activity 44

Create a program to generate random numbers in range 10 excluding 10

Example 4: Generate random numbers in range 5 including 5

```

Sub Main()
    Randomize() ' call this function to have different
    'each time you re-run the program
    Console.WriteLine(Int(Rnd() * 6))
    Console.ReadKey()
End Sub

```

The "* 6" says how many numbers it will go by. The output of that will either be 0, 1, 2, 3, 4 or 5; i.e. 6 different numbers. **Notice that 5 is generated here but not 6.**

Class activity 45

Create a program to generate random numbers in range 10 including 10

7.2.2 A class for generating random numbers, the System.Random class.

This is the new way in VB.Net to generate random numbers. The code below will give you a random number between 1 and 2,147,483,647.

```

Sub Main()
    Dim rndGen As New Random
    Dim randomnumber As Integer
    randomnumber = rndGen.Next
    Console.WriteLine(rndGen.Next()) 'generate between 1 and 2,147,483,647.
    Console.ReadKey()
End Sub

```

However, you might well require a number that is a little smaller. To get a random number between two set numbers, for example between 1 and 10 inclusive you can use the following:

```

Sub Main()
    Dim rndGen As New Random
    Console.WriteLine(rndGen.Next(1, 11))
    Console.ReadKey()
End Sub

```

Example 1: Generate 6 numbers between 1 and 40 inclusive randomly (similar to Ludo game)


```

sub Main()
    Dim count As Integer
    Dim rndGen As New Random
    For count = 1 To 6
        Console.WriteLine(rndGen.Next(1, 41))
    Next
    Console.ReadKey()
End Sub

```

Class activity 46

Write a program that will generate 10 random numbers between 1 and 50 inclusive.

7.3 String handling functions

Many examination questions involve manipulating strings.

7.3.1 Part 1: Treating strings as arrays

A string can be considered as an array and manipulated in the same way.

Example 1: Print a character in a string

```

Sub Main()
    Dim name As String = "Fawad Khan"
    Console.WriteLine(name(1)) 'print a
    Console.ReadKey()
End Sub

```

Note: Notice that the index/subscript of F starts at 0

Class activity 47

Write a program to print the character "K" in the string "Fawad Khan".

Example 2: Print all characters in a string

```

Sub Main()
    Dim name As String = "Fawad Khan"
    'print each character in name including space
    For count = 0 To Len(name) - 1
        Console.WriteLine(name(count))
    Next
    Console.ReadKey()
End Sub

```

Class activity 48

Write a program to print all characters in the string "I am Legend"

Example 3: Splitting a string to store each element in an array

```

Sub Main()
    Dim name As String = "Fawad Khan"
    Dim arraySplit() As String
    arraySplit = name.Split(" ") ' split the string by space
    Console.WriteLine("The element in the arraySplit are ")
    Console.WriteLine(arraySplit(0))
    Console.WriteLine(arraySplit(1))
    Console.ReadKey()
End Sub

```

Class activity 49

Write a program that splits the string "I-am-Legend" by "_", storing the elements in an array and finally display each element on console.

7.3.2 Part 2: Using in-built functions of strings

These simple functions will help you in manipulating strings.

Example 1:**(1) Length**

This function is used to find the length of any string you pass it, counting all the characters, including the spaces. In visual basic to find the length of a string we use the Len ("some string") function that returns the integer length of the string that it has been passed:

For example, the program below prints the length of the string "Fawad Khan"

```
Sub Main()
    Dim name As String = "Fawad Khan"
    Console.WriteLine(Len(name)) ' prints 10
    Console.ReadKey()
End Sub
```

Class activity 50

Write a program the print the length of the string "Hello World!"

(2) Position

This function allows us to find the position of an item within a given string and returns the position's location. In visual basic this is performed by the following command: InStr([string], [item])

For example, the program below prints the position of M and L respectively in the string "Fawad Khan"

```
Sub Main()
    Dim name As String = "Fawad Khan"
    Dim getposition As Integer
    getposition = InStr(name, "F") ' prints 1
    Console.WriteLine(getposition)
    getposition = InStr(name, "K") ' print 7
    Console.WriteLine(getposition)
    Console.ReadKey()
End Sub
```

Class activity 51

Write a program to get the position of letter e in the string "I am Legend"

Note: In the above class activity, the program will only print the position of the first character in the string. Notice also that the index of F is 1 when using InStr function as compared to when treating the string as an array where the index of F is 0. We can also use this command to search for strings within strings.

For example if we were to look for to see if the string "Fawad Khan" contains the word "Khan":

```
Sub Main()
    Dim name As String = "Fawad Khan"
    Dim searchitem As Integer
    'use Instr command to search for strings within strings
    searchitem = (Instr(name, "Khan"))
    If searchitem = 0 Then
        Console.WriteLine("Item not found ")
    Else
        Console.WriteLine("Item found at position " & searchitem)
    End If
    Console.ReadKey()
End Sub
```

Class activity 52

Write a program that take a word from a user and verify if that word is found in the string "Just do it". If no, tell the user that the word was not found else tell the user the word was found and prints the position of the 1st character in the word found in the string.

(3) Substring

This function allows you to snip items out of a string and return a substring.

Visual basic uses the following command:

[string].Substring([startPosition] , [lengthOfReturnString])

For example, we want to get "Mr Faw" from the string "Mr Fawad Khan":

```
Sub Main()
    Dim name As String = "Mr Fawad Khan"
    Dim subret As String
    subret = name.Substring(0, 6) 'stores "Mr Faw"
    Console.WriteLine(subret)
    Console.ReadKey()
End Sub
```

Class activity 53

Write a program to print the substring "Khan" from the string "Mr Fawad Khan".

(4) Concatenation

This function allows you to stick strings together (concatenate) so that you can start to build strings using variables. Visual Basic uses the following command:

[stringA & stringB]

For example, we can concatenate the word "am" with "I" and "Legend":

Read & Write Publications
 For Books Order: 0336-531441

```

Sub Main()
    'concatenate two string
    Dim mystring As String
    mystring = "AM"
    Console.WriteLine("I " & mystring & " Legend ")

    Console.ReadKey()|
End Sub

```

Class activity 54

Write a program to concatenate 2 words taken from a user, separated by a space and output it.

(5) CStr function

The CStr function is mainly used to convert a numeric value to a String.

```

Sub Main()
    Dim mydouble, mystring As String
    mydouble = 437.324
    mystring = CStr(mydouble)
    mystring = mystring & " is a decimal number "
    Console.WriteLine(mystring)

    Console.ReadKey()
End Sub

```

Class activity 55

Write a program that takes an integer from a user and prints the integer concatenated with the string "You have input the integer:"

(6) Mid function

The Mid function is used to retrieve a part of a given string. The format of the

Mid Function is **Mid(string, position, n)** where

- String is the whole text line.
- position is the starting position of the part to be retrieved
- n is the number of characters to retrieve.

For example, you can extract "wad" from the string "Mr Fawad Khan" as follows:

```

Sub Main()
    Dim name As String = "Mr Fawad Khan"
    Console.WriteLine(Mid(name, 6, 3)) ' print wad

    Console.ReadKey()
End Sub

```

Class activity 56

Write a program that extracts "Fawad" from the string "Mr Fawad Khan" and displays it to the user.

(7) The Left function

The Left function extracts the left portion of a string. The format is
Left (String, n)

where

- n is the starting position from the left of the string

For example, you can extract "Mr" from the string "Mr Fawad Khan" as follows:

```
Sub Main()  
    Dim name As String = "Mr Fawad Khan"  
    Console.WriteLine(Left(name, 2)) ' print Mr  
  
    Console.ReadKey()  
End Sub
```

Class activity 57

Write a program that extracts "Mr Fawad" from the string "Mr Fawad Khan" and displays it to the user.

(8) The Right function

The Right function extracts the right portion of a string. The format is Right (String, n)
where

- n is the starting position from the right of the string

For example, you can extract "an" from the string "Mr Fawad Khan" as follows:

```
Sub Main()  
    Dim name As String = "Mr Fawad Khan"  
    Console.WriteLine(Right(name, 2)) ' print an  
  
    Console.ReadKey()  
End Sub
```

Class activity 58

Write a program that extracts "Khan" from the string " Mr Fawad Khan " and displays it to the user.

Read & Write Publications
For Books Order: 0236-531441

Unit-8

Arrays

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

- Array - One dimensional
- Sample illustrations + VB practical on 10 arrays
- Arrays – Multidimensional
- Sample illustration of using 2D array:

Unit-8: Arrays

Data stored in a computer is stored at any location in memory. Instead of having similar pieces of data scattered all over memory, the similar data can be grouped together and stored in an array.

8.1 Array - One dimensional

An array is similar to a table, with parallel columns and rows of data. Below shows an array with a list of names:

Name	
1	Ali
2	Asim
3	Hadi
4	Mustafa
5	Naveed

Properties of an array (taking the above figure as an example):

- It has an identifier name, e.g. Name
- It has a size, e.g. 6
- It stores data of same types, e.g. String
- Each element in the array is identified using its subscript or index number, e.g. element Asim is found at index 2
- The largest and smallest index numbers are called the upper bound and lower bound of the array, e.g. Lower bound 1 and upper bound = 5

8.1.1 Initialising an array

When an array is to be created, the computer needs to know the size of the array because it is forced to store all the data in an array together and so it must reserve that amount of space in its memory.

Initialising an array means informing the computer program that an array is going to be used along with

- The identifier name of the array
- The sort of data that is going to be stored in the array, i.e. its data type
- How many items of data are going to be stored, so that it knows how much space to reserve (size of array)

Example: Initialising an array in Visual Basic

Dim name(5) As String

This Dim statement declares:

- the identifier name: Name
- the upper bound: 5 (this means array has reserved space for 6 elements - i.e. array size = 6)
- the data type: String

Further comments:

- The upper bound of 5 specifies that there can be a maximum of 6 data items, since Visual Basic starts with index 0 for arrays.
- The array that has been described so far is really only a list of single data items. It is possible to have an array which can be visualised as a two-dimensional table with rows and columns and a data value in each cell (to be covered later).
- All the data held in an array must be of the same data type in traditional programming languages. For many more recent languages, such as PHP, this restriction does not hold. We could extend our example to store each person's date of birth as well. This is possible as long as the dates of birth are stored as strings. An alternative solution is to use a record with a user-defined data type.

8.1.2 Reading data into an array

To assign data values to the elements of the array, we do this with assignment statements such as:

```
name(2) = "Asim"
```

This places the string "Ashley" at index position 2 in the array.

8.1.3 Writing data into arrays

Consider an application where we know that the program needs to store and process at least 20 surnames.

Method 1: Use a different identifier name for each surname as follows:

```
Dim surname1 As String
.
.
.
Dim surname20 As String

surname1 = "smith"
.
.
Surname20 = "carter"
```

This method is not efficient at all as it takes a large amount of space and effort. If there were 2000 Surnames, would we declare an identifier for each surname? Obviously, it does not make sense.

Read & Write Publications
For Books Order: 0336-531441

Method 2: Using an array

(i) Declaring an array

An array is a data structure that allows, for example, all 20 surnames to use the same identifier (here identifier is array name). Each surname is then referenced with an index number or subscript. The programmer must make clear the size of the array with the declaration statement:

```
Dim Surname(20) As String
```

This declaration says that the highest subscript we shall use (the upper bound of the array) is 20. Note that we are not going to use the array location at index 0, but instead start at index 1. Data values are then assigned with statements such as:

```
Surname(1) = "Williams"
```

```
Surname(6) = "Parker"
```

In summary, declaring the array tells the program:

- the identifier name for the array, e.g. Surname
- the sort of data that are going to be stored (Le. the data type), e.g. String
- the size of the array or, more precisely, its upper bound, e.g. 20.

Note: There are differences across different programming languages when declaring arrays. Visual Basic .NET assumes a lower bound of zero so our example could use subscripts 0 to 20 inclusive.

(ii) Initialising an array

```
For i = 0 To 20
```

```
    Surname(i) = ""
```

```
Next
```

Note: The use of the Index variable - acting as a loop counter - to supply the index value for the array.

Surname is a one-dimensional array. The values can be visualised as a one dimensional table with a cell for each data item and the subscript or index number as a label.

Surname	
1	Ali
2	Asim
3	Hadi
4	Mustafa
5	Naveed
6	Akram
7	William
8	Eric
.	
.	
.	
20	Edwin

8.1.4 Searching in an array

Searching for a particular name in the array involves a simple serial search, looking at each name in turn.

For example, the following pseudocode searches the Surname array for "Edwin":

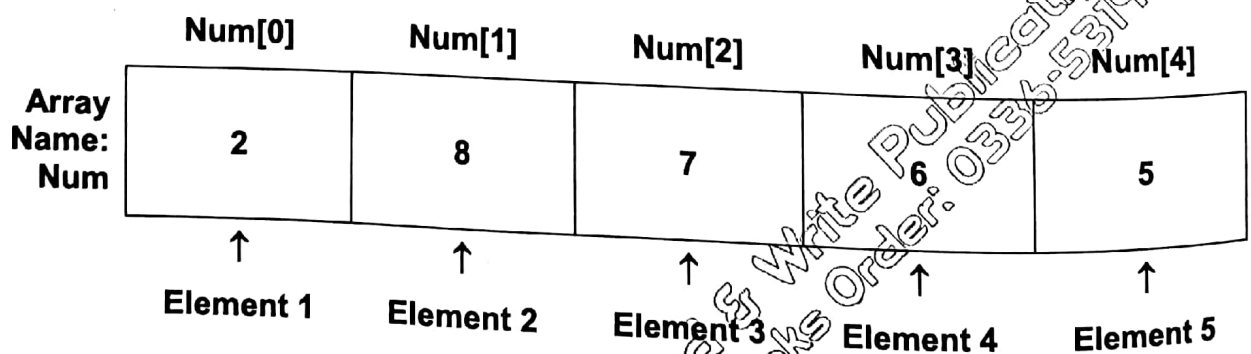
```

Index ← 1
IsFound ← False
While Index <= 20 And IsFound = False
    If Surname[Index] = " Edwin " Then
        IsFound ← True
        Output "Found"
    Else
        Add 1 to index
    End If
End While
If IsFound = False Then
    Output "Name not in array"
End If
  
```

- Note:** – The square brackets (e.g. Surname [index]) is used for array subscripts
- The assignment operator ← means that the value on the right hand side is assigned to the variable on the left hand side.
- Arrays are normally numbered from zero so that the first value is in location Name[0]. But in our case, we are ignoring the location Name[0] to make it easier to interpret the arrays.

8.2 Sample illustrations + VB practical on 10 arrays

An array stores a fixed-size sequential collection of elements of the same data type.



- We differentiate each item in the array by using subscript (also known as index)
- The individual values in the array are called the elements of the array.
- They're contiguous memory locations from index 0 (i.e. the lowest or first element) through to the highest index value (i.e. the last element).

Dimension of an Array

An array can be:

(a) One dimensional

- One dimensional array is like a list of items or a table that consists of one row of items or one column of items.

(ii) Multidimensional

E.g. 2D array, 3D array etc.

- A two dimensional array is a table of items that make up of rows and columns.

Normally it is sufficient to use one dimensional and two dimensional array, you only need to use higher dimensional arrays if you need to deal with more complex problems.

8.2.1 Creating arrays in VB 10

To declare an array in VB.Net, you use the Dim statement. For example,

(i) Declaring an array that will store data items of type String

'array identifier is animal

'upper bound =3

'lower bound = 0

'Array size = 4

Dim animals(3) As String

Note: You can also declare arrays of type integer, double, date, Boolean etc.

Class activity 59

Write a program to declare an array with name "Electronics" and size 4.

8.2.2 Inserting data into the array

You can then insert items/data into the array as follows:

Dim animals(3) As String

animals(0) = "Dog"

animals(1) = "Cat"

animals(2) = "Bird"

animals(3) = "Bat"

If you try to insert a 5th item in this array which is of size 4, the program will provide an error message as shown below when executed:

Read & Write Publications
For Books Order: 0336-531441

Sub Main()

```

'array identifier is animal
'upper bound =3
'lower bound = 0
'Array size = 4
Dim animals(3) As String
animals(0) = "Dog"
animals(1) = "Cat"
animals(2) = "Bird"
animals(3) = "Bat"
animals(4) = "Moskito"

```

End Sub

IndexOutOfRangeException was unhandled

Index was outside the bounds of the array.

Troubleshooting tips:

Class activity 60

Insert the following elements in the array Electronics, starting at index 0: Computer, TV, Watch.

8.2.3 Printing the data items in the array

- (i) One at a time

Console.WriteLine(animals(0)) 'console output Dog

Class activity 61

Print the 2nd element in the array Electronics.

- (ii) All at once

'print each data items

For index = 0 To 3

Console.WriteLine(animals(index))

Next

Class activity 62

Print all elements in the array Electronics.

Class activity 63

Create an array that stores 5 numbers taken from a user. Then, find the average of these 5 numbers by iterating through the array.

8.2.4 Initialising an array with empty string or integer values

Normally, you should initialise your array first before using it. In some programming language such as JAVA, it is mandatory to initialise your array first. In VB it is not mandatory but as a principle of good programming practice, you should do it.

- (i) **Initialising an array of type String**

```
Dim animals(3) As String
'initialising the array with empty string
For index = 0 To 3
    animals(index) = ""
Next
```

Class activity 64

Write a program to initialise the array Electronics discussed before.

(ii) Initialising an array of type Integer

```
Dim number(5) As Integer
'initialising the array with 0's
For index = 0 To 5
    number(index) = 0
Next
```

Class Activity 65

Create an array "Digits" of data type integer and size 5. Initialise the elements in the array with values 1.

8.2.5 Initialising an array with default data

You can also initialise arrays while declaring the arrays. For example,

```
Dim intData() As Integer = {12, 44, 55, 23, 46, 76}
Dim name() As String = {"Ali", "hadi", "Mustafa"}
```

Note: You can use underscore to continue your program on the next line

Class Activity 66

Write a program that creates an array "Furniture" with initial values chair, table and bed.

(6) Searching through arrays

```
'searching for an item in array
For i = 0 To 3
    If animals(i) = "Bat" Then
        Console.WriteLine("Item Found ")
    End If
Next
```

Class activity 67

Write a program that ask a user to input a furniture name and the program searches the array Furniture previously created to see if there is match. If so, the program displays the position of the furniture in the array.

Read & Write Publications
For Books Order: 0336-531441

Class activity 68

- (i) Write a program that takes 10 computer marks and store them in an array.
- (ii) Write a program to find the smallest mark in the array.
- (iii) Write a program to find the smallest mark in the array.

8.3 Arrays – Multidimensional

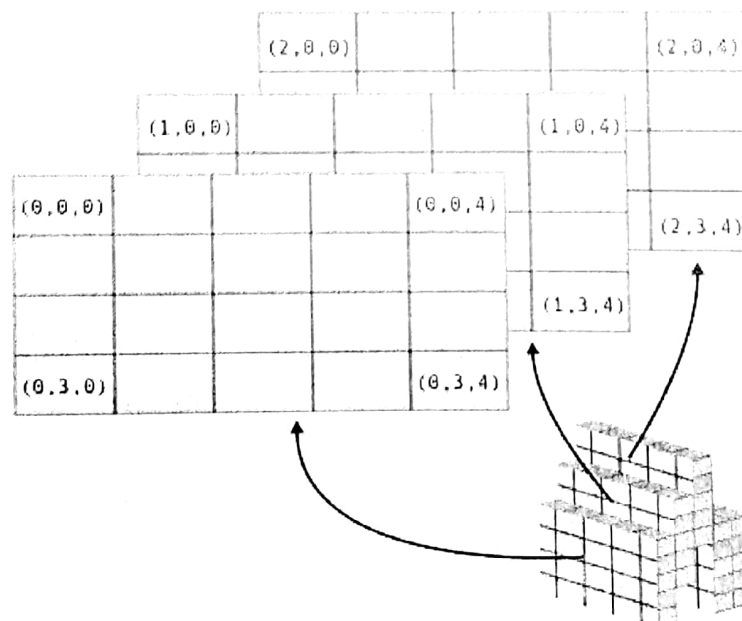
So far, you have used 1D array:

(0)	(1)	(2)	(3)	(4)
-----	-----	-----	-----	-----

There are also multi-dimensional arrays such as 2D, 3D arrays etc:

Two- dimensional array

(0,0)				(0,4)
(1,0)				
(3,0)				(3,4)

Three-dimensional array

It is possible to have an array which can be visualised as a two-dimensional table with rows and columns and a data value in each cell.

8.3.1 Declaring/Creating an empty 2D array:

The following statement declares a 12 array with 3 rows and 4 columns:

Dim emptyArray(2, 3) As Integer ' consists of 3 rows and 4 columns

This can be viewed as:

Class activity 69

Write a program to create a 2D array with name "array20" and having 2 rows and 3 columns.

8.3.2 Declaring a 2D array with initial values:

The following statement declares an array with initial values (1,2) which is row 1 and (3,4) which is row two.

```
Dim testarray(,) As Integer = {{1, 2}, {3, 4}}
```

Class activity 70

Write a program to create a 2D array with name "arrayTest2D" and having initial values (10, 10), (20, 20), (30, 30)

8.3.3 Printing a specific element in an array:

The following statements print the values in the array at position (0, 1) and (1, 1) respectively:

```
Dim testarray(,) As Integer = {{1, 2}, {3, 4}}
Console.WriteLine(testarray(0, 1)) ' prints 2
Console.WriteLine(testarray(1, 1)) ' prints 4
```

Class activity 71

Print the element at position (1, 2) in the arrayTest2D in the previous class activity.

8.3.4 Print all elements in an array:

The following code prints all elements in the array:

```
Dim testarray(,) As Integer = {{1, 2}, {3, 4}}
For row = 0 To 1
    For column = 0 To 1
        Console.WriteLine(testarray(row, column))
    Next
Next
```

Read & Write Publications
For Books Order: 0336-531441

Class activity 72

Print all the elements in the arrayTest2D used in class activity 72.

8.3.5 Entering elements in a 2D array and printing them:

The following code fills the array with values 1 everywhere and prints the elements at each array position after they have been inserted into the array:

```
Dim array2D(2, 3)
For row = 0 To 2
    For column = 0 To 3
        array2D(row, column) = 1
    Next
Next

For row = 0 To 2
    For column = 0 To 3
        Console.WriteLine(array2D(row, column))
    Next
Next
```

Class activity 73

Write a program that creates a 2D array with 3 rows and 4 columns and filled each row with values 2.

Search for a specific element and print its indexes:

The following codes search for the value 3 in an array already filled with initial values and prints the row and column index of 3 if it is found:

```
Dim testArray(,) As Integer = {{1, 2}, {3, 4}}
'searching for element 3
For row = 0 To 1
    For column = 0 To 1
        If testArray(row, column) = 3 Then
            'prints row index: 1 and column index: 0 if found
            Console.WriteLine("row index: " & row & " column index: " & column)
        End If
    Next
Next
```

Class activity 74

Write a program that creates a 2D array with initial values (1, 5), (2, 10), (3, 15) and searches for the value 10 in the array, returns its index if found.

8.4 Sample illustration of using 2D array:

School visits are made by five staff inspectors over a two-week period.

Two arrays are needed:

- Staff Name: a 1-D array to store the names of the five staff

- HomeVisit: a 2-D array to store the number of visits made by each staff member over the 14-day period.

The arrays need to be declared and initialise:

- The following code initialises the Staff Name array to the empty string

```
Dim staffName(4) As String 'stores 5 elements
```

```
'initialise staffname array to empty string
```

```
For i = 0 To 4
```

```
    staffName(i) = ""
```

```
Next
```

- The following code initialises all of the entries of the HomeVisit array to zero (note the use of nested loops):

```
Dim HomeVisit(4, 13) As Integer '5 rows(staff) by 14 columns(days)
```

```
Dim staffCount As Integer
```

```
Dim dayCount As Integer
```

```
Dim staffName(4) As String ' stores 5 elements
```

```
'initialise staffname array to empty string
```

```
For i = 0 To 4
```

```
    staffName(i) = ""
```

```
Next
```

```
'initialise the array homevisit to 0
```

```
For staffCount = 0 To 4
```

```
    For dayCount = 0 To 13
```

```
        HomeVisit(staffCount, dayCount) = 0
```

```
    Next
```

```
Next
```

Class activity 75

Create a 20 array to represent the multiplication tables from 0 to 10.

Hint: Use row index * column index

Read & Write Publications
For Books Order: 0336-531414

Unit-9

Topics

- Bubble sort

Sorting

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Unit-9: Sorting

Sorting means to arrange items in a particular order (usually either ascending or descending)

9.1 Bubble sort

Bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.

Bubble sort can be:

- Bubble-Up the Largest Element
- Bubble-down the smallest Element

Disadvantage of bubble sort:

- It is a very slow way of sorting data

Faster sorting algorithms are:

- Insertion sort
- Quick sort

Bubble sort works as follows:

1. Compare adjacent elements. If the first is greater than the second, swap them.
2. Repeat this for each pair of adjacent elements, starting with the first two and ending with last two. At this point the last element should be the greatest.
3. Repeat the steps for all elements except the last one.
4. Keep the repeating for one fewer element each time until there are no more pairs to compare.

9.1.1 Visual illustration of bubbling up the largest element

Traverse a collection of elements

- Move from the front to the end
- "Bubble" the largest value to the end using pair-wise comparisons and swapping

9.1.2 Step by step example of bubble sort

The initial values stored in the array are as follows:

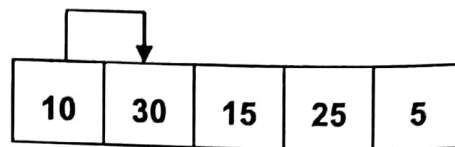
10	30	15	25	5
----	----	----	----	---

In the above example, nested loops are used to sort the array. The outer loop moves from 0 to 4 and with each iteration of outer loop, inner loop moves from 0 to $e - (i+1)$.

First of all, the value of i is 0 so the focus of outer loop is on the first element of the array. The sorting process will work as follows:

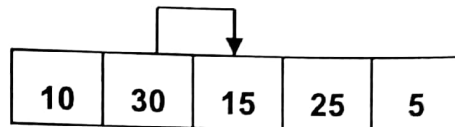
Pass 1

Iteration 1



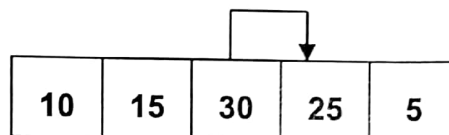
$j = 0$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 10 with 30. As 10 is not greater than 30, there will be no change in the array.

Iteration 2



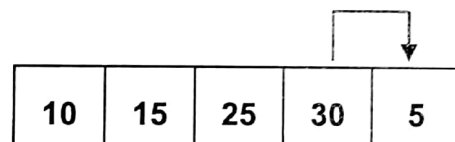
$j = 1$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 30 with 15. As 30 is greater than 15, both values will be interchanged and the array will be as follows:

Iteration 3

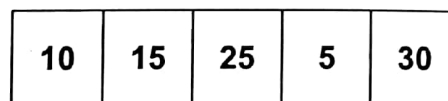


$j = 2$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 30 with 25. As 30 is greater than 25, both values will be interchanged and the array will be as follows:

Iteration 4



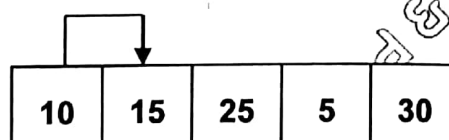
$j=3$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 30 with 5. As 30 is greater than 5, both values will be interchanged and the array will be as follows:



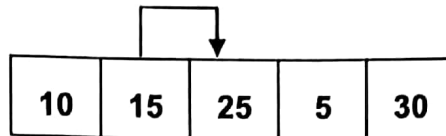
At this point, the inner loop is completed and the largest value in array has moved in the last element. It means that the position of largest value is now finalized. Now the control moves to the beginning of outer loop and the value of i becomes 1.

Pass 2

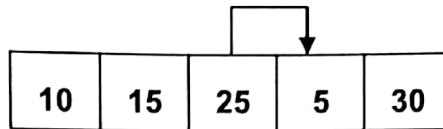
Iteration 1



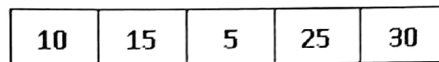
$j=0$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 10 with 15. As 10 is not greater than 15, there will be no change in the array.

Iteration 2

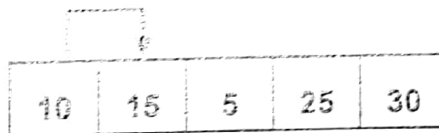
$j = 1$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 15 with 25. As 15 is not greater than 25, there will be no change in the array.

Iteration 3

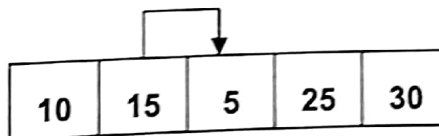
$j = 2$ so the statement will $\text{arr}(j) > \text{arr}(j+1)$ compares 25 with 5. As 25 is greater than 5, both values will be interchanged and the array will be as follows:



At this point, the inner loop is completed and the second largest value in array has moved in the second last element. It means that the position of second largest values is now finalized. Now the control moves to the beginning of outer loop and the value of i becomes 2.

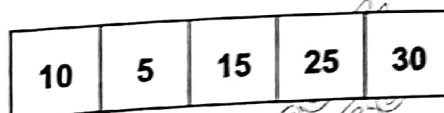
Pass 3**Iteration 1**

$j = 0$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 10 with 15. As 10 is not greater than 15, there will be no change in the array.

Iteration 2

$j = 1$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 15 with 5. As 15 is greater than 5, both values will be interchanged.

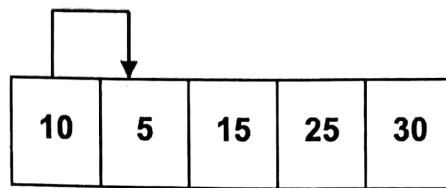
The array will be as follows:



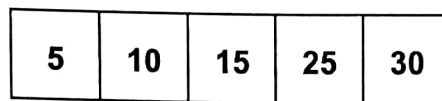
At this point, the inner loop is completed and the third largest value in the array has moved in the third last element. It means that the position of third largest value is now finalized. Now the control moves to the beginning of outer loop and the value of i becomes 3.

Pass 4

Iteration1



$j = 0$ so the statement $\text{arr}(j) > \text{arr}(j+1)$ compares 10 with 5. As 10 is greater than 5, both values will be interchanged and the array will be as follows:



At this point, the inner loop is completed and the fourth largest value in the array has moved in the fourth last element. It means that the position of fourth smallest value is now finalized. The position of smallest number is also automatically finalized. Now the outer loop also terminates and the array is sorted in ascending order.

9.1.3 Bubble sort algorithm implementation (un-optimised)

Pseudocode:

```

For count = 0 to numberOfElementsInArray-1
    For pass = 0 to numberOfElementsInArray-2
        If Array[pass] > Array[pass+1]
            Temp = Array[pass]
            Array[pass] = Array[pass+1]
            Array[pass+1] = Temp
        End If
    End For
End For
  
```

Program code:

We can optimised it by using a Boolean variable to check when the array is already sorted and stop the program, thus not wasting time and resources. The section which follows illustrate an optimised code which sorts in the array in the minimum number of pass possible.

Real Time Write Publications
 For Order: 0336-5314141

```

Sub Main()
    Dim arr(4) As Integer, temp As Integer
    Dim s As Integer, e, i As Integer
    s = LBound(arr)
    e = UBound(arr)
    arr(0) = 10
    arr(1) = 30
    arr(2) = 15
    arr(3) = 25
    arr(4) = 5
    Console.WriteLine("Original array is :")
    For i = s To e
        Console.WriteLine(arr(i))
    Next
    For i = s To e
        For j = s To e - 1
            If arr(j) > arr(j + 1) Then
                temp = arr(j)
                arr(j) = arr(j + 1)
                arr(j + 1) = temp
            End If
        Next
    Next
    Console.WriteLine("Sorted Array is = ")
    For i = s To e
        Console.WriteLine(arr(i))
    Next
End Sub

```

Class activity 76

Write the program code to sort an array with initial values 77, 42, 35, 12, 101, 5

9.1.5 Bubble sort algorithm implementation (optimised)

There may be cases where only a few elements were out of place and after a couple of "bubble ups", the collection was sorted. We want to be able to detect this and "stop early".

Improving the previous bubble sort algorithm

- We can use a boolean variable to determine if any swapping occurred during the "bubble up."
- If no swapping occurred, then we know that the collection is already sorted!
- This boolean "flag" needs to be reset after each "bubble up."

Pseudocode solution:

For count = 0 to numberOfElementsInArray-1

Swapped = False

For pass = 0 to numberOfElementsInArray-2

If Array[pass] > Array[pass+1] Then

Read & Write Publications
For Books Order: 0336 531441

```

        Temp = Array[count]
        Array[count] = Array[pass+1]
        Array[pass+1] = Temp
        Swapped = True

    End If

End For

If swapped = False Then
    Exit program
End if

End For

```

Program code:

```

Dim arrs() As Integer = {2, 7, 4, 1, 5, 3}
Dim i, j, temp, arrayLength, count As Integer
Dim swapped As Boolean
For i = 0 To (arrs.Length() - 1)
    swapped = False
    For j = 0 To (arrs.Length() - 2)
        If arrs(j) > arrs(j + 1) Then
            temp = arrs(j + 1)
            arrs(j + 1) = arrs(j)
            arrs(j) = temp
            swapped = True
        End If
    Next
    count = count + 1
    If swapped = False Then
        Exit For ' exit the loop
    End If
Next
Console.WriteLine("count is " & count) ' print number of passes
For i = 0 To (arrs.Length - 1)
    Console.WriteLine(arrs(i))
Next

```

Class activity 77

Write the program code that sorts an array with initial values 77, 42, 35, 12, 101, 5 and ensuring this sorting is done in the minimum number of pass possible.

Class activity 78

Construct a trace table for the following code and dry run it using the values in the array arrs:

```
Dim arrs() As Integer = {2, 7, 4, 1, 5, 3}
Dim i, j, temp, arrLength, count As Integer
Dim swapped As Boolean
For i = 0 To (arrs.Length() - 1)
    swapped = False
    For j = 0 To (arrs.Length() - 2)
        If arrs(j) > arrs(j + 1) Then
            temp = arrs(j + 1)
            arrs(j + 1) = arrs(j)
            arrs(j) = temp
            swapped = True
        End If
    Next
    count = count + 1
    If swapped = False Then
        Exit For ' exit the loop
    End If
Next
Console.WriteLine("count is " & count) ' print number of passes
For i = 0 To (arrs.Length - 1)
    Console.WriteLine(arrs(i))
Next
```

Read & Write Publications
For Books Order: 0336-531414

Unit-10

Files

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

- Keys terms associate with files
- Types of files
- File manipulation
- File extensions
- Pseudocode structures for file handling
- Sample illustration of manipulating serial files
- Sample illustration of manipulating sequential files

Unit-10: Files

Data stored on a computer system are organised as files. When data are stored, they are normally connected in some way. For example, data stored about the students in a class is connected because it all refers to the same set of people. Each student has his or her own data stored, for example, their name, address, telephone number and exam grades.

The data for a number of students is shown below:

Name	Address	Telephone	Exam grades
Imsha Dar	113a New Town	323525666	A,CC,D
Nooshun Bilal	17b Old Town	788585858	A,A,B,A
Alison Smith	15 Old Town	745865859	B,B,A,C

10.1 Keys terms associate with files

- A file consists of a large quantity of similar data. E.g. a student file is shown above
- A record is the information referring to a particular student. Each record contains exactly the same fields: name, address, etc. E.g. 3 records shown above
- Each type of information is called a field. A number of fields make up a record and all records from the same file must contain the same fields.

10.2 Types of files

There are different types of files that are used including:

- Serial file
- Sequential file
- Indexed sequential file
- Random file

For paper 2, you need to be aware of serial and sequential file only.

10.2.1 Serial files

Serial files have no order, no aids to searching and no complicated methods for adding new data.

Data storage/Insertion in a serial file:

- In a serial file, data are stored in the file in the order in which it arrives. The new data are simply placed at the end of the existing file (i.e. "appended"). This is illustrated below where some names are inserted into a serial file:

Read & Write Publications
For Book Order: 0336-531441

1. Insert Ahsan <div>Ahsan</div>	2. Insert Jack <div>Ahsan Jack</div>
3. Insert Tom <div>Ahsan Jack Tom</div>	4. Insert Mary <div>Ahsan Jack Tom Mary</div>

Searching a serial file:

- Searching a data require the whole file to be read, starting with the first record and ending either when the requested record is found or the end of the file is read without finding the data. This is illustrated below where the name being searched for is Tom:

1. Is equal to Tom? No → <div>Ahsan Jack Tom Mary</div>	2. Is equal to Tom? No → <div>Ahsan Jack Tom Mary</div>
3. Is equal to Tom? Yes Found → <div>Ahsan Jack Tom Mary</div>	

Read & Write Pub
For Books Order: 0336

Example of serial file:

The book that you are reading now.

- The words were all typed in, in order, and that is how they should be read. Reading this book would be impossible if all the words were in alphabetic order.

Advantage of serial file:

- Simplest form of storage

Disadvantage of serial file:

- Finding an item may be very difficult as data are in an unstructured way

Use of serial files:

- Serial files are used when it is unlikely that the data will be needed again
- Serial files are used when the order of the data should be determined by when it is input

10.2.2 Sequential files

In a sequential file, data records are stored in sequence using a key field. The data in such a file can only be read from beginning to end.

Data storage/Insertion in a sequential file:

- Adding a new record is more complex because it has to be placed in the correct position in the file. To do this, all the records that come after it have to be moved in order to make space for the new one. This is illustrated below:

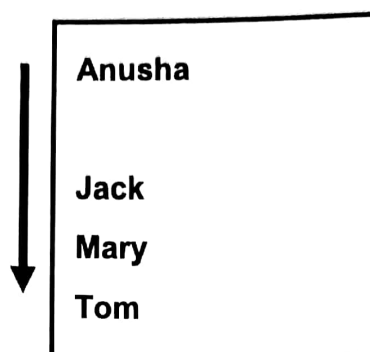
A section of a school student file might look like this:

Anusha
Jack
Mary
Tom

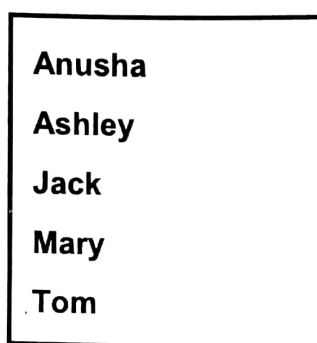
Anusha , Jack, Mary, Tom were inserted into the sequential file in this order. However, since this is a sequential file and it is sorted in order of Name, the names were inserted in their correct position each time.

If a new student arrives whose name is **Ashley**, storage space must be found between **Anusha** and Jack. To do this all the other records have to be moved down one place, starting with Tom, then Mary, and then Jack.

Read & Write Publications
For Book Order: 0236-531411



This leaves a space into which Ashley's record can be inserted and the order of the records in the file can be maintained.



Note: It is necessary to move Tom first and not Jack as this leaves a space into which Ashley's record can be inserted and the order of the records in the file can be maintained.

Searching in a sequential file:

- It is similar to searching in a serial file which requires the whole file to be read, starting with the first record and ending either when the requested record is found or the end of the file is read without finding the data. This is illustrated below where the name being searched for

Since the sequential file is in order of key field name, Anusha is found at the start itself.

Example of sequential:

The data of a set of students can be stored in a computer in:

- In alphabetic order of their name or
- In the order that they performed in a Computing exam or
- By date of birth with the oldest first

If the data are in alphabetic order of name and the computer is asked for Zaid's record, it must start looking from the beginning of the file.

Advantage of sequential file:

- Easier to find a particular record as the data has been arranged

Drawback of sequential file:

- Insertion is time consuming

Use of sequential file:

- This type of file structure is only used for files that have a small number of records or that change infrequently. It is not suitable for large files.

Note: In paper 2, you are going to manipulate sequential file only. In paper 4, you are going to learn about random files and how to manipulate them.

10.3 File manipulation

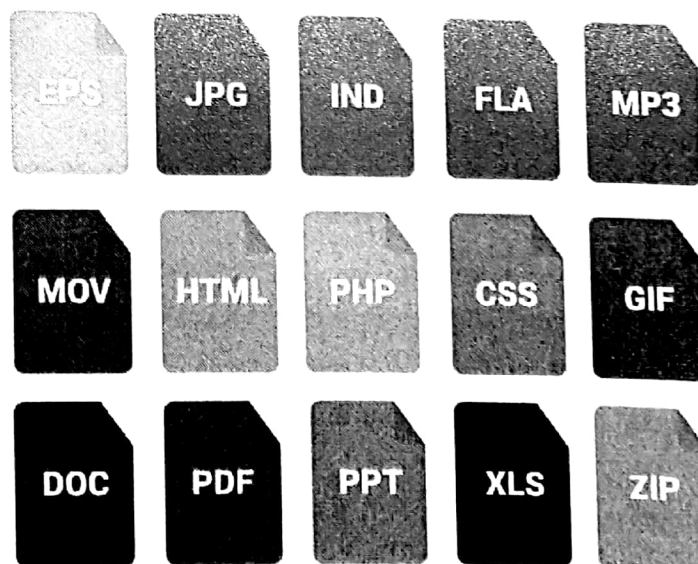
Files can be manipulated in terms of:

- Write/Save new data into files (e.g. saving a game)
- Read data in a file (e.g. opening an existing spread sheet)
- Delete data from files (deleting unwanted records in a database or text file)
- Update data in files (e.g. saving modification made to a word documents)
- Append data in files (e.g. adding a new record to an existing files of records)
- Search for and retrieve data in files (e.g. get records from database or text file)

10.4 File extensions

File extensions such as .txt, .CSV, .xls, .pptx, .exe are only present to tell the operating system or program how to handle them. They don't define what's inside the file.

Different files have different extensions such as:



ations
36-531441

10.5 Pseudocode structures for file handling**(i) Opening a file**

OPENFILE <filename> FOR READ/WRITE/APPEND

(ii) Reading a line of text from the file

READFILE <filename>, <string>

- (iii) **Write a line of text to the file**
WRITE FILE <filename>, <string>
- (iv) **Close file**
CLOSEFILE
- (v) **Function to test for end of file**
EOF()

Example 1: Creating a student file and writing one line in it

```
OPEN FILE student.txt FOR WRITE
WRITEFILE student.txt, "This is a new file"
CLOSEFILE student.txt
```

Example 2: Creating a student file and writing multiple lines in it

```
OPEN FILE student.txt FOR WRITE
Repeat
    Input StudentName
    WRITEFILE student.txt, StudentName
Until no more data to input
CLOSEFILE student.txt
```

Example 3: Reading a whole student file line by line

```
OPEN FILE student.txt FOR READ
WHILE NOT EOF (student.txt)
    READFILE student.txt, NameRetrieved
    Output NameRetrieved
ENDWHILE
CLOSEFILE student.txt
```

Example 4: Reading data from one student file line by line and writing to another student file

```
OPENFILE student.txt FOR READ
OPENFILE studentNew.txt FOR WRITE
    WHILE NOT EOF (student.txt)
        READFILE student.txt, NameRetrieved
        WRITEFILE studentNew.txt, NameRetrieved
    ENDWHILE
CLOSEFILE student.txt
CLOSEFILE studentNew.txt
```


Example 5: Appending data to a student file

```
OPENFILE student.txt FOR APPEND
    WRITEFILE student.txt, "fawad"
CLOSEFILE student.txt
```

Example 6: Searching for a specific name in a student file

```
Found = false
OPENFILE student.txt FOR READ
Input NameToFind
WHILE NOT EOF (student.txt) and Found = false
    READFILE student.txt, NameRetrieved
    If NameToFind = NameRetrieved Then
        Found = True
        Output "Name Found"
    End If
End While
If Found = false
    Output "Name not Found"
End If
CLOSEFILE student.txt
```

10.6 Sample illustration of manipulating serial files

At all times when manipulating files in VB, you will have to import the System.IO file class. This class contains methods (i.e. functions) for many common file operations such as creating, opening, copying, deleting etc.

```
Imports System.IO ' namespace
Module Module1
    Sub Main()
    End Sub
```

ed & Write Publications
or Books Order: 0336-531111

(i) Creating a file and writing one line

```
Imports System.IO ' namespace
Module Module1

    Sub Main()
        fileWriter() ' Call procedure
    End Sub

    Sub fileWriter()
        'create an object of the file class to use its predefine functions
        Dim filewriter As StreamWriter
        'the object FileWriter uses the method CreateText to create a file
        filewriter = File.CreateText("F:/Myfile.txt")
        'the object FileWriter uses the method writeline to write a line
        filewriter.WriteLine("Hello, i have created you ")
        'close file else won't see data since still in RAM
        filewriter.Close()
    End Sub

End Module
```

The file will appear in partition F (if you have, else use partition C) in your computer:

<

It will have the following contents:

Myfile - Notepad
File Edit Format View Help
Hello, i have created you

(ii) Creating a file and writing multiple lines

```
Imports System.IO ' namespace
Module Module1

    Sub Main()
        fileWriter() ' Call procedure
    End Sub

    Sub fileWriter()
        'create an object of the file class to use its predefine functions
        Dim filewriter As StreamWriter
        'the object FileWriter uses the method CreateText to create a file
        filewriter = File.CreateText("F:/Myfile.txt")
        For i = 1 To 10
            'the object FileWriter uses the method writeline to write a line
            filewriter.WriteLine("Hello, i have created you ")
        Next
        'close file else won't see data since still in RAM
        filewriter.Close()
    End Sub

End Module
```

The content of the file will be as follows:

```
Myfile - Notepad
File Edit Format View Help
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
```

(iii) Reading a whole file line by line

```
Sub FileRead()
    Dim dataRetrieved As String
    'create an object of the file class to use its predefined functions
    Dim filereader As StreamReader
    'the object FileReader uses the method OpenText to open a file for reading
    filereader = File.OpenText("F:/Myfile.txt")
    'read file till end of file
    Do While filereader.Peek <> -1
        'output what is being read
        dataRetrieved = filereader.ReadLine
        Console.WriteLine(dataRetrieved)
    Loop
End Sub
```

(iv) Appending data to a file

```
Sub FileAppend()
    'create an object of the file class to use its predefined functions
    Dim FileAppend As StreamWriter
    'the object FileAppend uses the method AppendText to
    'add a line at the end of a file
    FileAppend = File.AppendText("F:/Myfile.txt")
    FileAppend.WriteLine("Hello, i have added this line to your text ")
    FileAppend.Close()
End Sub
```

The file will look as follows:

Read & Write Public
For Books Order: 0336-5

```

Myfile - Notepad
File Edit Format View Help
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have created you
Hello, i have added this line to your text

```

(v) Searching for a specific data in a file

Suppose the student file is as follows:

```

Myfile - Notepad
File Edit Format View Help
Ali
Ahmed
Akram
Sohail
Hadi
Mustafa
Asim
Basit

```

Searching for the name "Hadi":

```

Sub Filesearch()
    Dim found As Boolean = False
    'variable to store data retrived
    Dim dataRetrived As String
    'variable to keep track of line number
    Dim count As Integer = 1
    'variable to get input
    Dim NameToFind As String
    'create an object of the file class to use its functions
    Dim Filereader As StreamReader
    'the object FileReader uses the method OpenText to open a file for reading
    Filereader = File.OpenText("F:/Myfile.txt")
    'get input from user
    NameToFind = Console.ReadLine
    Do While Filereader.Peek <> -1
        'output what is being read
        dataRetrived = Filereader.ReadLine
        If dataRetrived = NameToFind Then
            Console.WriteLine("Found at line " & count)
            found = True
        End If
        count = count + 1
    Loop
    If found = False Then
        Console.WriteLine("Data Not found ")
    End If

```

The output will be:

file:///C:/Users/Fawad Khan/AppData/Local/Tem
 Hadi
 Found at line 5

Note: Input is case sensitive. Make sure the name is typed as it is.

(vi) Updating a specific data in a file

Suppose the student file is as follows:

Myfile - Notepad
 File Edit Format View Hel
 Tom
 Jack
 Anusha
 Harry

Changing the name "Anusha" to "Ashley"

Sub FileAmend()

```
'Variable to check if name to be changed found
Dim found As Boolean = False
'variable to store retrieved name from file
Dim RetrievedName As String = ""
Dim NameToChange, NewName As String
'create objects of the file class to use its functions
Dim OlderFileReader As StreamReader
Dim NewFileWriter As StreamWriter
OlderFileReader = File.OpenText("F:/Myfile.txt")
NewFileWriter = File.CreateText("F:/MyfileNew.txt")
'get name to change in file from user
NameToChange = Console.ReadLine
'loop through file contents
Do While OlderFileReader.Peek <> -1
    RetrievedName = OlderFileReader.ReadLine
    'check if there is a match
    If RetrievedName <> NameToChange Then
        'write in new file other names
        NewFileWriter.WriteLine(RetrievedName)
    Else
        'write in new file the new name
        NewFileWriter.WriteLine(NewName)
        found = True
    End If
Loop
```

Read & Write
 For Books Only

```

'close the file
OlderFileReader.Close()
NewFileWriter.Close()
'inform user if name to be changed not found
If found = False Then
    Console.Write("Name not found ")
End If
'ensure any previous backup file are deleted
File.Delete("F:/Myfile.bak")
'rename old file to backup
File.Move("F:/Myfile.txt", "F:/Myfile.bak")
'rename new file created to old name
File.Move("F:/Myfilenew.txt", "F:/Myfile.txt")
End Sub

```

10.7 Sample illustration of manipulating sequential files

At all times when manipulating files in VB, you will have to import the System.IO file class. This class contains methods (i.e. functions) for many common file operations such as creating, opening, copying, deleting etc.

```

Imports System.IO ' namespace
Module Module1
    Sub Main()
    End Sub

```

(i) Creating a file and writing one line

Similar to serial file (See previous section)

(ii) Creating a file and writing multiple lines

If you want to write more than one records in order of Names, you will have to ensure that the name is placed in the correct position each time. See part

(iii) in this section.

(iv) Reading a whole file line by line

Similar to serial file (See previous section)

(v) Appending data to a file

Suppose the file contains the following data:

Myfile - Notepad
File Edit Format View Help
Ashley
Harry
Jack
Tom

Read & Write
For Books ©

Suppose we want to insert the name "Lavind":

```

Sub FileInsert() ' for sequential file
'variable to check if name to be changed found
Dim found As Boolean = False
'variable to store retrieved name from file
Dim RetrievedName As String = ""
'variable to check if new name has been inserted
Dim checkinserted As Boolean = False
'variable to get new name from user
Dim Newname As String
'create objects of the file class to use its functions
Dim OldFileReader As StreamReader
Dim NewFileWriter As StreamWriter
OldFileReader = File.OpenText("F:\Myfile.txt")
NewFileWriter = File.CreateText("F:\MyfileNew.txt")
'get name to change in file from user
Newname = Console.ReadLine
'loop through file contents
Do While OldFileReader.Peek <> -1
    RetrievedName = OldFileReader.ReadLine
    'ensure that new value placed in file only once
    If checkinserted = False Then
        If Newname < RetrievedName Then
            NewFileWriter.WriteLine(Newname)
            checkinserted = True
        End If
    End If
    NewFileWriter.WriteLine(RetrievedName)
Loop
If checkinserted = False Then 'if name largest than all other names
    NewFileWriter.WriteLine(Newname)
End If
OldFileReader.Close() : NewFileWriter.Close()
End Sub

```

Output:

```

Myfile - Notepad
File Edit Format View Help
Ashley
Harry
Jack
Lavind
Tom

```

Note: We have created a new file StudentNew this time instead of deleting the old studentFile.

(vi) Searching for a specific data in a file

Similar to serial file (See previous section)

(vii) Updating a specific data in a file

Similar to serial file (See previous section)

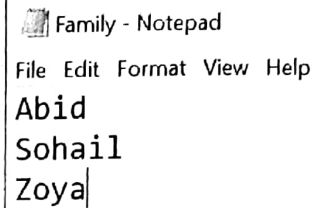
Read & Write Publications
For Books Order 0336-531441

Class activity 79

Below is a sequential file with name Family.txt containing the following records?

Write a program that displays a menu where a user can choose and perform the following actions:

- read a chosen record
- update a chosen record
- insert a new record



```
Family - Notepad
File Edit Format View Help
Abid
Sohail
Zoya
```

Read & Write Publications
For Books Order: 0336-531441

Unit-11

Topics

Identifier Table

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Unit-11: Identifier Table

An identifier table represents the different identifiers in a program code. It usually consists of 3 fields namely:

- the Identifiers
- its data type
- the purpose/explanation/description of the identifier

Example 1

Consider the following pseudocode:

```

For Count = 1 to 50
  Input name, hours_wrkd, rate_pay
    If name <> "END" Then
      Totalpay = rate_pay * hours_wrkd
      Print Name, Totalpay
    End If
  Next Count
End For

```

The identifier table will be as follows:

Identifier	Data type	Description
Count	INTEGER	Loop counter
name	STRING	Stores the name of the employee
hours_wrkd	DOUBLE	Stores the number of hours worked by the employee
rate_pay	DECIMAL	The amount paid per hour of working

Draw the identifier table for the following pseudocode:

```

Pie = 3.142
Input radius
While radius > 0
  Area = Pie * radius * radius
  Output area
  Input radius
End While

```

Read & Write Publications
For Books Order: 0336-25

Example 2:

Consider the following program code:

```
'searching for an item in array
For i = 0 To 3
    If animals(i) = "Bat" Then
        Console.WriteLine("Bat is found at index " & i & " in the array ")
    End If
Next
```

The identifier table will be as follows:

Identifier	Data type	Description
i	INTEGER	Loop counter which is used as the subscript/index for the array
animals	ARRAY [3] : String	Array to store names of animals

Class activity 81

Draw the identifier table for the following program code:

```
Dim testArray(,) As Integer = {{1, 2}, {3, 4}}
```

```
For row = 0 To 1
```

```
    For column = 0 To 1
```

```
        Console.WriteLine(testArray(row, column))
```

```
    Next
```

```
Next
```

Read & Write Publications
For Books Order: 0336-531441

Unit-12

Software Development

A Level

Computer Science

P-2 NOTES

Fawad Khan

Cell: 0321-6386013

fawad.khan11@hotmail.com



GREEN HALL
Resource Center

| Gulberg | Johar Town | Wapda Town |
| DHA Phase-1 | DHA Phase-4 | Saddar Cantt |
For Books Order: 0336-5314141

Topics

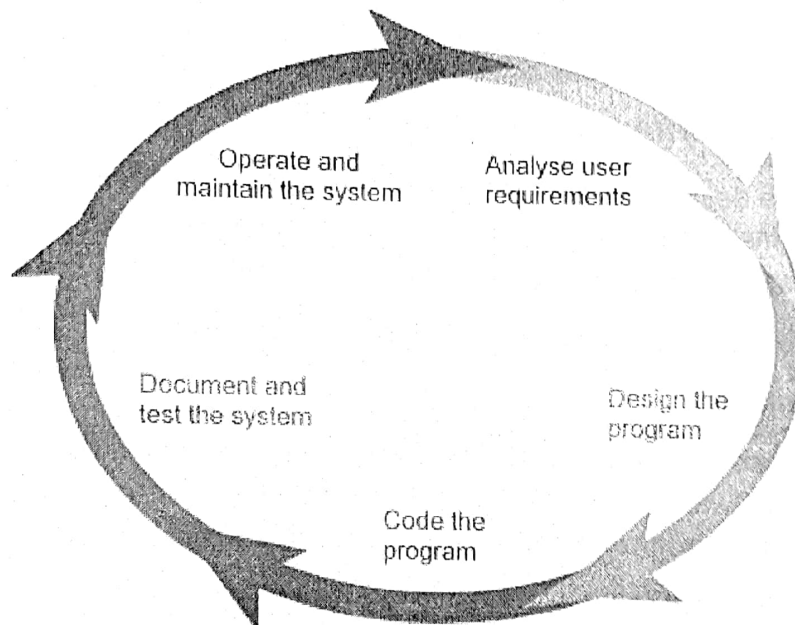
- Stages of program development life cycle (PDLC)
- System maintenance
- Program testing
- Testing strategies
- Features found in a typical Integrated Development Environment (IDE)

Unit-12: Software Development

When software applications are built, programmers do not rush and start writing code. Instead, they follow an organised plan/methodology which breaks the software development into a series of tasks. There are different methodologies but they tend to be variations of what is called the program development life cycle (PDLC).

The **program development life cycle (PDLC)** is an outline of each of the steps used to build software applications. Similarly to the way the system development life cycle (SDLC) guides the systems analyst through development of an information system, the program development life cycle is a tool used to guide computer programmers through the development of an application (software).

12.1 Stages of program development life cycle (PDLC)



Program/Software development consists of a number of stages:

- Requirement identification
- Design
- Coding
- Testing
- Documentation
- Maintenance

(i) Requirement Identification

Define the problem to be solved, and write program specifications including descriptions of the program's inputs, processing, outputs, and user interface.

(ii) Design

Develop a detailed logic plan using a tool such as pseudocode, flowcharts or structure diagrams to group the program's activities into modules and devise a method of solution or algorithm for each module

(iii) Coding

Translate the design into an application using a programming language by creating the user interface and writing code with comments to explain the purpose of code statements.

(iv) Testing

Test the program, finding and correcting errors (debugging) until it is error free and contains enough safeguards to ensure the desired results.

(v) Documentation

Formalise and complete end-user (external) documentation

(vi) Maintenance

Provide support to end users, correcting any unanticipated errors that emerge and identify user-requested modifications (known as enhancements). Once errors or enhancements are identified, the program development life cycle begins again.

Note: Stages may overlap

12.2 System maintenance

Systems/software are designed for a well-defined purpose and should realize that purpose; if they do, they are considered successful. During use it may become necessary to alter the system for some reason - this is known as maintenance.

There are three different types of maintenance to be considered:

(i) Corrective maintenance

Corrective maintenance is necessary when a fault or bug is found in the operation of the new system. These are software bugs which were not picked up at the formal testing stage. A technician or the original programmers will be needed to correct the error.

(ii) Adaptive maintenance

Adaptive maintenance is necessary when conditions change from those that existed when the original system was created. This may be because of a change in the law (tax rates may change, for example) or the hardware may be changed, so that changes need to be made to the software for it to remain functional.

(iii) Perfective maintenance

Perfective maintenance is required to "tweak" the system so that it performs better. For example, searching for a particular stock item may be quite slow. The technician decides that supplier details should be stored in another file rather than with the details of the stock, the size of the stock file is reduced and it is far quicker to search. This has not changed how the system operates as far as the user is concerned but the performance improves.

In sum, a system/software is maintained after going live (installation):

- Corrective maintenance fixes software bugs which were not picked up at the formal testing stage.
- Adaptive maintenance changes the system to cater for changes in the law, or hardware or new business procedures.
- Perfective maintenance makes the system perform better.

Note: The need for continual maintenance for a system/software

Computing and computer applications change regularly through advances in technology, new ideas, different legal frameworks and different business practices. A system should never be considered to be finished. Rather than being a linear process with a beginning, a middle and an end, it should be thought of as a circular process, continually returning to previous stages to fine tune them and take advantage of changing circumstances.

In addition to the need for continual maintenance, all systems have a natural lifespan and eventually need to be replaced. The limited lifespan of a system is known as obsolescence. Even the most up-to-date and most expensive system eventually becomes out-of-date. This may be because a piece of hardware needs to be replaced and it is not possible to find anything that is compatible with the software; because the competitor for a business updates all its systems and customers find their service more impressive; or because customers expect more to be done for them.

12.3 Program testing

Programs are tested after they are written to ensure that they are error free and perform what they are intended to.

Program errors may be of the following types:

- (1) Errors in syntax (incorrect use of the programming language)
- (2) Errors in logic (the program does not do what was intended)
- (3) Run-time errors (ones that are only discovered when the program runs).

12.3.1 Syntax errors

Syntax errors are errors in the grammar of the program language, i.e., the rules of the language have been broken.

Common errors are:

- typing errors, such as Selct for Select
- an If statement without a matching End If statement
- For statement without a matching Next statement
- An opening parenthesis without a closing parenthesis: (a + b.

How syntax errors detected?

- Most syntax errors are spotted by the programmer before an attempt to run the code.
- Some program editors help to identify syntax errors. For example, the Visual Basic.NET editor underlines any variable in the code which has not been declared. Similarly, it highlights declared variables which have not been used in the code.

12.3.2 Logic errors

A logic error occurs when the programmer makes a mistake in their logic for some part of the program.

Common errors are:

(1) Suppose a programmer wants the first 10 positive integers to be output and creates the following algorithm:

For Count = 0 To 10

Output Count

Next Count

This algorithm has no grammatical errors but it does not produce the correct result. It produces the integers 0 to 10 not 1 to 10. This type of error can only be found by thorough testing. The programmer can carefully check the code by reading it or can attempt to run the program.

(2) Another common error is to use the wrong arithmetic symbol, e.g. $a+b$ instead of $a-b$, or to put parentheses in the wrong place, such as $(a+b-c)*d$ instead of $(a+b)-c*d$. These errors should be identified during testing and are all a mistake in the logic of the programmer.

12.3.3 Run-time errors

Run-time errors occur during the execution (running) of the program.

Example of a runtime error

A typical error is to have an expression such as $(a+b)/(c-d)$, used in the program but c is equal to d , resulting in an attempted division by zero. The problem is that this error may be undetected for a considerable time because the equality of c and d rarely happens. However, it could be disastrous when it does.

It is a good idea to test the denominator before division to ensure that this error doesn't crash the program. Instead, the programmer can "trap" the error and displays an error message.

12.4 Testing strategies

Software creation is a human activity and, as a result, is subject to errors. All software has to be tested. Only the simplest of programs work first time and most contain errors (or bugs) that have to be found. If a program has been designed and written in modules, it is much easier to debug because the programmer should be able to test each module independently of the others. There are several strategies the programmer can adopt for testing a program.

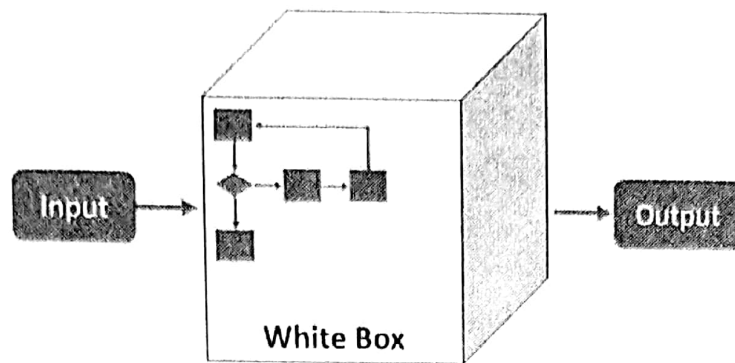
12.4.1 White box testing

White box testing is where testers examine each line of code for the correct logic and accuracy.

This may be done manually by:

- (1) Drawing up a table to record the values of the variables after each instruction is executed. Examples could include selection statements (If and Case Select), where every possible condition must be tested.
- (2) Debugging software can be used to run the program step by step and then display the values of the variables.





Trace table

It is a technique used to test algorithms to make sure that no logical errors occur. Working through an algorithm using test data values is called a dry run or trace. The Hand tracing or desk checking or 'dry running' allows you to use a **trace table** to:

- see what code will do before you have to run it
- find where errors in your code are

Note: A trace table need to keep track (trace) all the variables and outputs when drawn.

To make a trace table, the first step is to note the table headings, this involves the following:

1. Variables: note all the variables in the piece of code you are looking at (this includes arrays). Note each variable as a heading
2. Output: note if there is an output and put this as a heading

Example: Dry run the following codes

```
Dim y As Integer = 3
For x = 1 To 4
    y = y + x
Next
Console.WriteLine(y)
```

Solution:

To do this, we create a trace table:

X	Y	Output
	3	
1	4	
2	6	
3	9	
4	13	13

Class Activity 82

Dry run the following algorithm using the sequence of input data -2, 0, 3, 5, 12, 16:

$n=0$

WHILE $n \leq 0$

INPUT n

ENDWHILE

Total $\leftarrow 0$

Count $\leftarrow 0$

REPEAT

INPUT Number

Total \leftarrow **Total** + **Number**

Count \leftarrow **Count** + 1

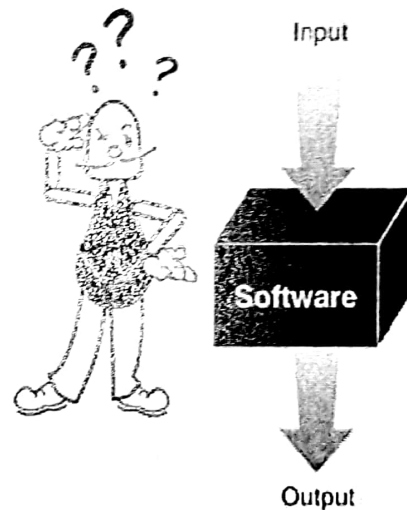
UNTIL **Count** = n

OUTPUT **Total/Count**

12.4.2 Black box testing

Black box testing is one of the earliest forms of test where the programmer uses test data, for which the results have already been calculated, and compares the results from the program with those that are expected.

- The programmer does not look at the individual code to see what is happening - the code is viewed as being inside a "black box". (no knowledge of the internal code and structure required)
- The testing only considers the inputs and the outputs they produce.



Black-Box Testing

Black-box test design is usually described as focusing on

- testing functional requirements
- external specifications or interface
- specifications of the program or module.

Black box testing makes use of test plan:

A test plan is used to test the correctness of a program by using test data which includes:

- Normal data is data that the system is expected to process.
- Borderline/extreme/boundary data is at the boundary of data that the system is expected to process.
- Invalid/abnormal data is data that the system is expected to reject.

Consider the pseudocode below verifies if an input value lies between 5.5 and 9 inclusive:

Input n

While n <= 5.5 OR n >= 9

Input n

End While

In this case:

- Normal data: any value between 5.5 and 9 (excluding 5.5 and 9)
- Borderline data: 5.5 and 9
- Invalid data: they could be 4, 10 or even 2000.

12.4.3 Stub testing

A stub is a dummy (fake) program or component used when the actual program code is not ready for testing. It acts as a temporary replacement for a called module and give the same output as the actual product or software.

For example, if there are 3 modules and last module is yet to be completed and there is no time, then we will use dummy program to complete that 3rd module and we will run whole 3 modules also.

Illustration of stub testing

Consider a program for calculating tax on a product that requires 3 modules. The 1st module is the main module which gets the product's price and tax rate from a user and the 2nd one calculate the tax on the product and the 3rd module output the tax on the product.

The 2nd module to calculate the tax need to be dynamic, that is it will determine the tax price based on tax rate provided by the user.

However, due to lack of time and the need to deliver the program to the client, the programmer hasn't been able to code the module dynamically. So he created a stub module to calculate the tax price by taking the tax rate as 15% and price as Rs100 instead of taking it from the user.

This thus enables him to test the whole program as shown below.

Expected program:

```
Sub Main()
    Dim price, taxPrice, TaxRate As Decimal
    Console.WriteLine("Please enter the price of the product ")
    price = Console.ReadLine
    Console.WriteLine("Please enter the tax rate ")
    TaxRate = Console.ReadLine
    taxPrice = funCalcTaxPrice(TaxRate, price)
    proutTaxPrice(taxPrice)
End Sub

Function funCalcTaxPrice(MyVal paraTaxRate As Decimal, MyVal paraPrice As Decimal) As Decimal
    Return paraTaxRate * paraPrice
End Function

Sub proutTaxPrice(MyVal paraTaxPrice As Decimal)
    Console.WriteLine("the tax price is " & paraTaxPrice)
End Sub
```

Actual program with stub:

```

Sub Main()
    Dim price, taxPrice, TaxRate As Decimal
    Console.WriteLine("please enter the price of the product ")
    price = Console.ReadLine
    Console.WriteLine("Please enter the tax rate ")
    TaxRate = Console.ReadLine
    taxPrice = funCaltaxPrice()
    proOutTaxPrice(taxPrice)
End Sub

Function funCaltaxPrice() As Decimal ' stub module
    Return 0.15 * 100 ' decides to use Rs 100 as price and 15% for tax
End Function

Sub proOutTaxPrice(ByVal paraTaxPrice As Decimal)
    Console.WriteLine("the tax price is " & paraTaxPrice)
End Sub

```

12.5 Features found in a typical Integrated Development Environment (IDE)

Procedural programming languages tend to have an integrated development environment (IDE) that provides for all stages in the program's development: creation, compilation, testing and execution. An IDE enables the programmer to use some very sophisticated techniques such as:

- Suggesting programming elements
- Reporting syntax errors
- Debugging
- Watch values of elements during run time

These are discussed in the section below:

(i) For coding

The IDE suggest programming elements defined by the user (e.g. variables, constants, procedure, function identifiers) and programming elements already in VB (e.g. reserved words)) as we type and we just need to select them to display them instead of typing them fully as well as no need to remember them by heart, saving time and allowing easy programming.

(ii) For initial error detection

During translation of a high-level language into machine code (or intermediate code), syntax errors are spotted and reported by the programmer. Some text editors report a syntax error as the code is typed in. For example, if you write the instruction `Mass = 56` and the variable `Mass` has not been declared, some editors report it immediately. In any case, it is reported during translation by the interpreter or compiler.

Module Module1

Sub Name 'Mass' is not declared.

Mass = 56

End Sub

correction suggestions)

(iii) For debugging

Debugging is the process of finding and reducing the number of bugs/defects/errors in computer programs.

— Single stepping

This technique is used by the programmer to see what happens when each line in a program is executed.

```

Sub Main()
    prosplit()
End Sub

Sub prosplit()
    Dim name As String = "Mr fawad Khan"
    Dim arraySplit() As String
    arraySplit = name.Split(" ") ' split the string by space
    Console.WriteLine("the elements in the arraySplit are ")
    Console.WriteLine(arraySplit(0))
    Console.WriteLine(arraySplit(1))
    Console.WriteLine(arraySplit(2))
End Sub

```

— Breakpoints

This technique is used to arrange for a program to stop at a given instruction and display the values of the variables at this point, known as breakpoints. The program can then be continued or stopped by the programmer.

A programmer may step through a program either by setting break points or by stopping after the execution of each instruction. When a programmer sets a breakpoint, the program can then be run to this point. The programmer can then step through the following instructions one at a time, run the program to the next break point, or run to the end of the program.

```

Sub prosplit()
    Dim name As String = "Mr fawad Khan"
    Dim arraySplit() As String
    arraySplit = name.Split(" ") ' split the string by space
    Console.WriteLine("the elements in the arraySplit are ")
    Console.WriteLine(arraySplit(0))
    Console.WriteLine(arraySplit(1))
    Console.WriteLine(arraySplit(2))
End Sub

```

– Variables/expressions report window

A watch window (Visual Basic .NET calls this the "Immediate Window") can be used to display the values of variables and expressions. The window displays to the programmer the current value (and maybe the data type) of each variable. To obtain these values, the programmer simply types the name of the variable or the expression into the window and the information is displayed.

```
Sub prosplit()  
    Dim name As String = "Mr fawad Khan"  
    Dim arraySplit() As String  
    arraySplit = name.Split(" ") ' split the string by space  
    Console.WriteLine("the elements in the arraySplit are ")  
    Console.WriteLine(arraySplit(0))  
    Console.WriteLine(arraySplit(1))  
    Console.WriteLine(arraySplit(2))  
End Sub
```

Read & Write Publications
For Books Order: 0336-531441