LANDMARK UNIVERSITY
COURSE NOTE

COLLEGE: COLLEGE OF SCIENCE AND ENGINEERING
DEPARTMENT: COMPUTER SCIENCE

COURSE:

COURSE CODE: CSC 122
COURSE TITLE: INTRODUCTION TO PROBLEM SOLVING
COURSE UNIT: 2
COURSE STATUS: COMPULSORY

COURSE DESCRIPTION

THE PURPOSE OF THIS COURSE IS TO INTRODUCE FIRST YEAR COMPUTER SCIENCE STUDENTS TO HIGH LEVEL LANGUAGE USING QBASIC PROGRAMMING LANGUAGE. ATTENTION IS PAID TO UNDERSTANDING OF PROGRAMMING CONCEPTS AND BUILDING STUDENTS' SKILLS IN PROGRAMMING. IT IS ASSUMED THAT STUDENTS POSSESS LITTLE OR NO PRIOR KNOWLEDGE OF PROGRAMMING, THE COURSE THEREFORE BEGINS WITH A COMPREHENSIVE STUDY OF PROGRAMMING TOOLS SUCH AS ALGORITHM (PSEUDOCODE AND FLOWCHART) AND DECISION TABLE. THE KNOWLEDGE OF ALGORITHM IS CENTRAL TO THE LEARNING OF ANY PROGRAMMING LANGUAGE. CONCEPTS SUCH AS CONTROL FLOW, SEQUENCE, ITERATION, DECISION AND PROCESSION WERE COVERED IN DEPTH WITH EXAMPLES. QBASIC PROGRAMMING SYMBOLS, KEYWORDS, DATA TYPES, OPERATORS, FUNCTION AND CONTROL STRUCTURE ARE TAUGHT AND THE KNOWLEDGE USED TO SOLVE PROBLEMS FROM THE VERY SIMPLE TO COMPLEX ONES.

COURSE JUSTIFICATION

QBASIC IS A PROCEDURAL LANGUAGE AND EASY TO UNDERSTAND ESPECIALLY FOR BEGINNERS.

COURSE OBJECTIVES

AT THE END OF THIS COURSE, STUDENTS SHOULD BE ABLE TO:

- DEMONSTRATE GOOD UNDERSTANDING OF ALGORITHM
- UNDERSTAND BASIC PROGRAMMING CONCEPTS AND TECHNIQUES
- PROVIDE ALGORITHMIC SOLUTION TO PROGRAMMING TASKS
- PROVIDE INTERPRETATION TO PROGRAMS WRITTEN BY OTHERS
- DESIGN, DEVELOP DEBUG AND IMPLEMENT PROGRAM IN QBASIC

COURSE CONTENTS

PROGRAMMING TOOLS: ALGORITHM; COMPONENTS OF ALGORITHM, DIFFERENT WAYS OF PRESENTING ALGORITHMS. FLOWCHARTING OF ALGORITHM. DECISION TABLE. PSEUDOCODE; PSEUDOCODE STATEMENT FOR INPUT, OUTPUT, ITERATION, DECISION AND PROCESSION. ARITHMETIC, RELATIONAL AND LOGICAL OPERATIONS IN PSEUDOCODE, USE OF SUB PROCESS IN PSEUDOCODE. INTRODUCTION TO QBASIC PROGRAMMING: SYMBOLS, KEYWORDS, IDENTIFIERS, DATA TYPES, OPERATORS, CONTROL STRUCTURE, FUNCTION, PROCEDURES. ARRAYS: 1-D AND MULTI-DIMENSIONAL ARRAYS.

COURSE REQUIREMENT

IT IS ASSUMED THAT STUDENTS TAKING THIS COURSE POSSESS LITTLE OR NO PRIOR KNOWLEDGE IN PROGRAMMING. IT IS EXPECTED HOWEVER THAT STUDENTS EXHIBIT WILLINGNESS AND APTITUDE FOR LEARNING.

METHOD OF GRADING

S/N	GRADING	SCORE(%)
1.	CONTINUOUS ASSESSMENTS • C.AI • C.AII (MID-SEMESTER TEST)	7% 15% 8%

	• C.AIII	
2.	Assignment	
3.	Practical (Laboratory work)/ Case Studies	
4.	Final Examination	70%
5.	Total	100

Course Delivery Strategies:
Lectures are delivered via electronic media (e-learning platform and power point presentations). Students are also encouraged to work with our programmers and avail themselves of laboratory facilities for practical work. Students are expected to demonstrate their understanding of concepts by completing given tasks in class and submitting assignments as at when due.

Resources used/Reading Materials

- Reuben Aremu. Manuscript on Introduction to Problem Solving (Unpublished)
- Behrouz and Firouz (2008). Foundations of Computer Science. Cengage Learning. ISBN-13:978-1-84480-700-0
- Okeyinka A.E. (1998). Introduction to computer technology. ISBN : 978-31933-5-X
- Peter B. (1982). Further Computer Programming in BASIC. Thomas Nelson ans sons ISBN 0-17-431266-0

1.0. INTRODUCTION

Regardless of the area of study, computer science is all about solving problems with computers. The problems that we want to solve can come from any real-world problem or perhaps even from the abstract world. We need to have a standard systematic approach to solving problems. Since we will be using computers to solve problems, it is important to first understand the computer's information processing model.

Figure 1 below assumes a single CPU (Central Processing Unit). Many computers today have multiple CPUs, so you can imagine the model duplicated multiple times within the computer.

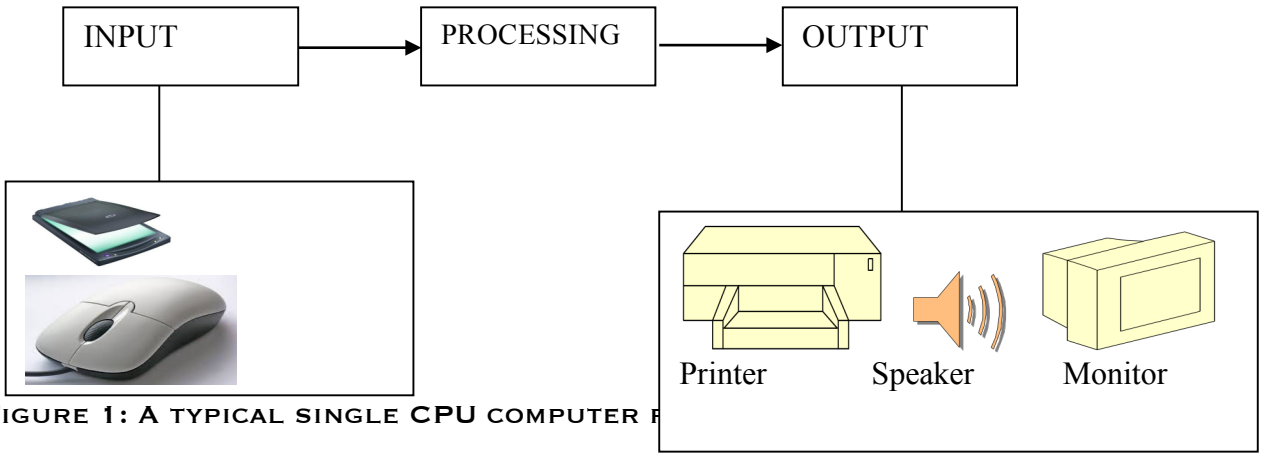


Figure 1: A typical single CPU computer f

Problems are solved when the computer accepts some kind of user input (via keyboard/mouse, game control movements), then process the input and produce some kind of output (e.g., images, test, sound). Sometimes the incoming and outgoing data may be in the form of hard drives or network devices.

In order to apply the above model (Figure 1) to problem solving, it is assumed that some kind of input information are given to work with in order to produce some desired output solution. However, the above model is quite simplified. For larger and more complex problems, there is the need to iterate (i.e., repeat) the input/process/output stages multiple times in sequence, producing intermediate results along the way that solve part of a given problem, but not necessarily the

whole problem. For simple computations, the above model is sufficient. It is the "problem solving" part of the process that is the interesting part, so we'll break this down a little.

2.0. PROBLEM SOLVING

Problem solving can be viewed as a six (6) step activity as stated below:

1. Understand the Problem
2. Formulate a Model
3. Develop an Algorithm
4. Write the Program
5. Test the Program
6. Evaluate the Solution

The problem is easily solved by simply getting the input, computing something and producing the output. Let us now examine the 6 steps to problem solving.

2.1. Understand the Problem:

It sounds strange, but the first step of solving any problem is to make sure that you understand the problem that you are trying to solve.

Defining the problem is the first step towards a problem solution. A systematic approach to problem definition, leads to a good understanding of the problem. Here is a tried and tested method for defining (or specifying) any given problem:

Divide the problem into three (3) separate components:

a. input or source data provided
b. processing - a list of what actions are to be performed in transforming input to output
c. output or end result required

These three components can further be refined as follows:

a. Input or source data provided:
   i. What input data/information is available?
   ii. What does it represent?
   iii. What format is it in?
   iv. Is anything missing?
   v. Do I have everything that I need?

b. Processing
   i. What am I going to have to compute?

c. Output or end result required:
   i. What output information am I trying to produce?
   ii. What do I want the result to look like ... text, a picture, a graph ...?

Consider a simple example of how the input/process/output works on a simple problem:

Example: Calculate the average grade for all students in a class.

1. **Input:** Get all the grades ... perhaps by typing them in via the keyboard or by reading them from a USB flash drive or hard disk.

2. **Process:** Add them all up and compute the average grade.

3. **Output:** Output the answer to either the monitor, to the printer, to the USB flash drive or hard disk ... or a combination of any of these devices.

Finally, we should understand the kind of processing that needs to be performed on the data. This leads to the next step.

2.2. Formulate a Model:

Now there is the need to understand the processing part of the problem. Many problems break down into smaller problems that require some kind of simple mathematical computations in order to process the data. In our example, we are going to compute the average of the incoming grades. So, we need to know the model (or formula) for computing the average of a bunch of numbers. If there is no such "formula", we need to develop one. Often, however, the problem breaks down into simple computations that we well understand. Sometimes, we can look up certain formulas in a book or online if we get stuck.

In order to come up with a model, we need to fully understand the information available to us. Assuming that the input data is a bunch of integers or real numbers $x_1, x_2, \ldots, x_n$ representing a grade percentage, we can use the following computational model:

$$Average1 = \frac{(x_1 + x_2 + \ldots + x_n)}{n}$$

where the result will be a number from 0 to 100. That is very straight forward (assuming that we knew the formula for computing the average of a bunch of numbers). However, this approach will not work if the input data is a set of letter grades like B-, C, A+, F, D-, etc. because we cannot perform addition and division on the letters. This problem solving step must figure out a way to produce an average from such letters. Thinking is required. After some thought, we may decide to assign an integer number to the incoming letters as follows:

A+ = 12
A = 11
B+ = 9
B = 6
C+ = 6
C = 5
D+ = 3
D = 2
F= 0

If we assume that these newly assigned grade numbers are $y_1, y_2, \ldots y_n$ then we can use the following computational model:

$$Average2 = \frac{(y_1 + y_2 + \ldots + y_n)}{n}$$

where the result will be a number from 0 to 12. As for the output, if we want it as a percentage, then we can use either Average1 directly or use (Average2 / 12), depending on the input that we had originally. If we wanted a letter grade as output, then we would have to use (Average1/100*12) or (Average1*0.12) or Average2 and then map that to some kind of "lookup table" that allows us to look up a grade letter according to a number from 0 to 12. Do you understand this step in the problems solving process? It is all about figuring out how you will make use of the available data to compute an answer.

2.3. Develop an Algorithm:

Now that we understand the problem and have formulated a model, it is time to come up with a precise plan of what we want the computer to do by developing an algorithm to solve the problem.

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

An ***algorithm*** is a well-defined computational procedure consisting of a finite set of unambiguous rules (instructions) which specify a finite sequence of operations that takes some values or set of values, as *input*, and produces some values or set of values, as *output*. In other word, an algorithm is a procedure that accepts data, manipulate them following the prescribed steps, so as to eventually fill the required unknown with the desired value(s).

2.3.1. Properties of an Algorithm:

i. **An algorithm must have at least an input data item**

ii. **It must be precise and unambiguous**
An algorithm must be precisely and unambiguously described, so that there remains no uncertainty. An instruction that says "shuffle the deck of card" may make sense to some of us, but the machine will not have a clue on how to execute it, unless the detail steps are described. An instruction that says "lift the restriction" will cause much puzzlement even to the human readers

iii. **It must give the correct solution in all cases**
This means that it must solve every instance of the problem. For example, a program that computes the area of a rectangle should work on all possible dimensions of the rectangle, within the limits of the programming language and the machine

iv. **It must eventually end.**
The ultimate purpose of an algorithm is to solve a problem. If the program does not stop when executed, we will not be able to get any result from it. Therefore, an algorithm must contain a finite number of steps in its execution. Note that an algorithm that merely contains a finite number of steps may not terminate during execution, due to the presence of 'infinite loop".

2.3.2. Algorithms and Humans

- Algorithms are not a natural way of stating a problem's solution, because we do not normally state our plan of action.
- We tend to execute as we think about the problem. Hence, there are inherent difficulties when writing an algorithm.
- We normally tailor our plans of action to the particular problem at hand and not to a general problem (i.e. a near-sighted approach to problem solving)
- We usually do not write out our plan, because we are usually unaware of the basic ideas we use to formulate the plan. We hardly think about it – we just do it.
- Computer programmers need to adopt a scientific approach to problem solving, i.e. writing algorithms that are comprehensive and precise.
- We need to be aware of the assumptions we make and of the initial conditions.
- Be careful not to overlook a step in the procedure just because it seems obvious.
- Remember, machines do not have judgment, intuition or common sense!

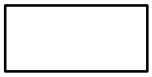
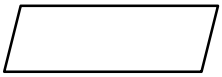
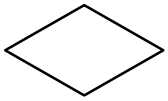
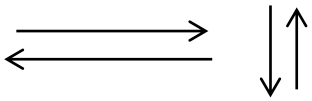
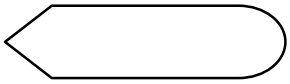
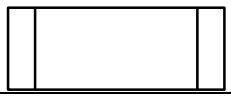
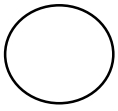
2.3.3. Developing an Algorithm

- Understand the problem (Do problem by hand. Note the steps)
- Devise a plan (look for familiarity and patterns)
- Carry out the plan (trace)
- Review the plan (refinement)

2.3.4. Understanding the Algorithm

Possibly the simplest and easiest method to understand the steps in an algorithm, is by using the **flowchart** method. This algorithm is composed of block symbols to represent each step in the solution process as well as the directed paths of each step. The most common block symbols are as displayed in Table 1:

Table 1 Flowchart Symbol Representation

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

There are many other block symbols, used in flow charting, but we will restrict our usage to the symbols described table 1. They are sufficient to illustrate the steps in developing solutions to the simple problems we will be dealing with.

2.3.5. The Algorithmic Language

During development of an algorithm, the language gradually progresses from English towards a programming language notation. An intermediate notation called pseudo-code is commonly used to express algorithms.

2.3.6. Algorithmic Structure

Every algorithm should have the following sections, in the stated order:

Header: Algorithm's name or title.
Declaration: A brief description of algorithm and variables. i.e. a statement of the purpose.
Body: Sequence of steps
Terminator: End statement

2.3.7. How to write Pseudocode

An algorithm can be written in pseudocode using six (6) basic computer operations:

A computer can receive information.

Typical pseudocode instructions to receive information are:

```
Read name;
Get name;
Read number1, number2;
```

A computer can output (print) information.

Typical pseudocode instructions are:

```
Print name
Write "The average is", ave
```

A computer can perform arithmetic operation

Typical pseudocode instructions:

```
Add number to total, or
Total = Total + Number
Ave = sum/total
```

A computer can assign a value to a piece of data:

e.g. to assign/give data an initial value:

```
Initialize total to zero
Set count to 0
```

To assign a computed value:

```
Total = Price + Tax
```

A computer can compare two (2) pieces of information and select one of two alternative actions.

Typical pseudocode e.g.

```
IF number < 0 then
        add 1 to neg_number
ELSE
        add one to positive number
        end-if
```

A computer can repeat a group of actions.

Typical pseudocode e.g.

```
Repeat until total = 50
        read number
        write number
        add 1 to total
end-repeat
                        OR

while total < = 50 do:
        read number
        write number
end-while
```

Now, let's review the plan and write out algorithm for the average problem in the specified format:

Algorithm **Average**

This algorithm reads a list of numbers and computes their average.

Let: **SUM** be the total of the numbers read
NUM be the number of items in the list
AVE be the average of all the numbers
Set **SUM** to 0,
Set **COUNTER** to 0. (i.e. initialize variables)

```
While (COUNTER < NUM)
        do:
                Read number;
```

```
                // (i.e. add number to SUM, storing result in SUM)
                SUM = SUM + number

                // (i.e. add 1 to COUNTER, storing result in COUNTER)
                COUNTER = COUNTER + 1;
    end-while

    if COUNTER = 0 then
        AVE = 0
    else
        AVE = SUM/ NUM
    Stop.
```

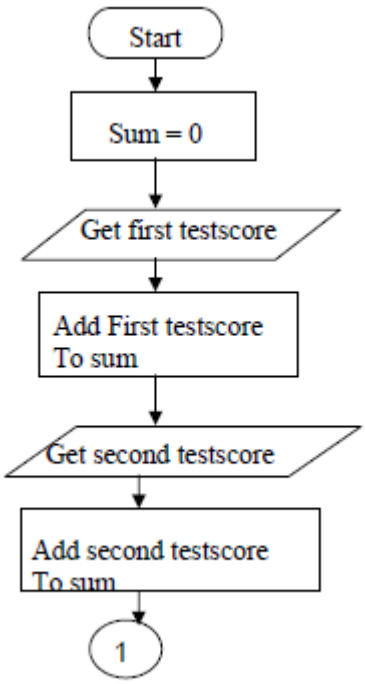
EXAMPLES OF FLOWCHARTS, ALGORITHM AND PSEUDOCODES

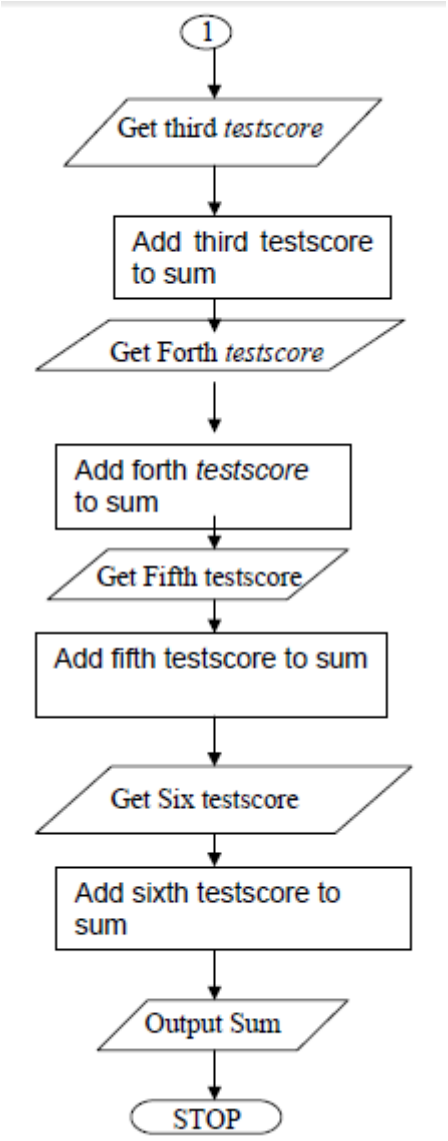
Design an algorithm and the corresponding flowchart for adding the test scores as given below:
26, 49, 98, 87, 62, 75

Algorithm
addingTestScores

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

b) The corresponding flowchart is as follows:





The algorithm and the flowchart above illustrate the steps for solving the problem of adding six testscores. Where one testscore is added to sum at a time. Both the algorithm and flowchart should always have a **Start** step at the beginning of the algorithm or flowchart and at least one **stop** step at the end, or anywhere in the algorithm or flowchart. Since we want the sum of six testscore, then we should have a container for the resulting sum. In this example, the container is called **sum** and we make sure that sum should start with a zero value by step 2.

Example 2

Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

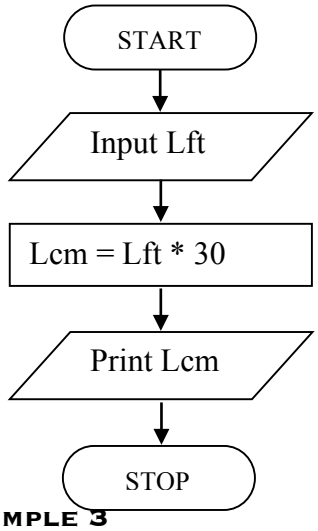
Pseudocode:

- Input the length in feet (Lft)
- Calculate the length in cm (Lcm) by multiplying LFT with
- Print length in cm (LCM)

Algorithm:

Step 1: Input Lft
Step 2: Lcm = Lft x 30
Step 3: Print Lcm

Flowchart



Example 3

Write an algorithm to determine a student's final grade and indicate whether it is pass or fail. The final grade is calculated as the average of four marks.

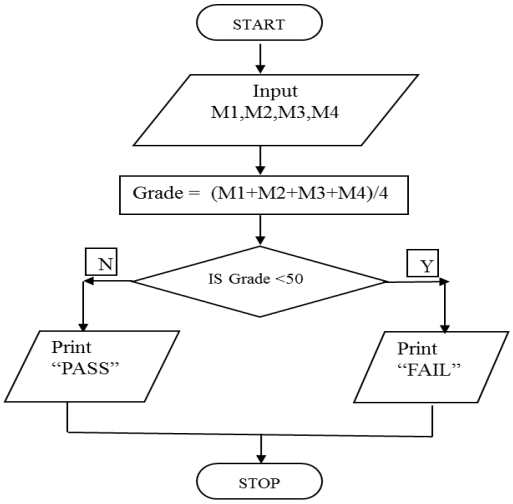
Pseudocode:

- **Input a set of 4 marks**
- **Calculate their average by summing and dividing by 4**
- **IF average is below 50**
  Print "FAIL"
  ELSE
  Print "PASS"

Algorithm

Step 1: Input M1,M2,M3,M4
Step 2: GRADE (M1+M2+M3+M4)/4
Step 3: IF (GRADE < 50) then
Print "FAIL"
ELSE
Print "PASS"
ENDIF

Flowchart



Example 4

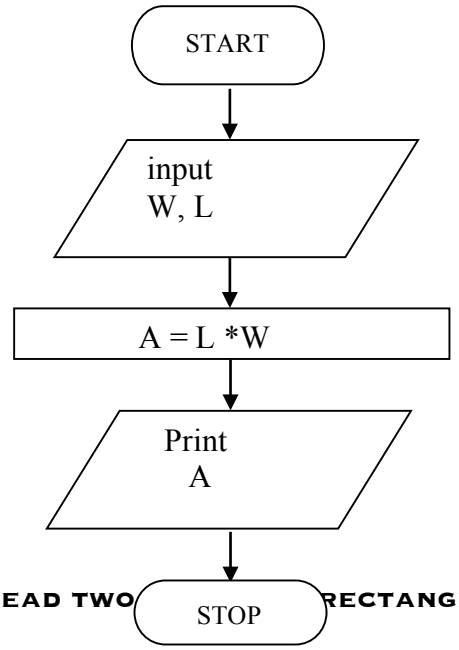
Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

Pseudocode to read two sides of a rectangle and calculate its area:

- **Input the width (W) and Length (L) of a rectangle**

- Calculate the area (A) by multiplying L with W
- Print A

Flowchart to read two sides of a rectangle and calculate its area:



Algorithm to read two sides of a rectangle and calculate its area:

- Step 1: Input W,L
- Step 2: A = L x W
- Step 3: Print A

Example 5

Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation
$AX2 + BX + C = 0$

- Hint: D = SQRT ($B2 - 4AC$), and the roots are: X1 = (–B + D)/2A and X2 = (–B – D)/2A

Pseudocode to calculate the roots of a quadratic equation:

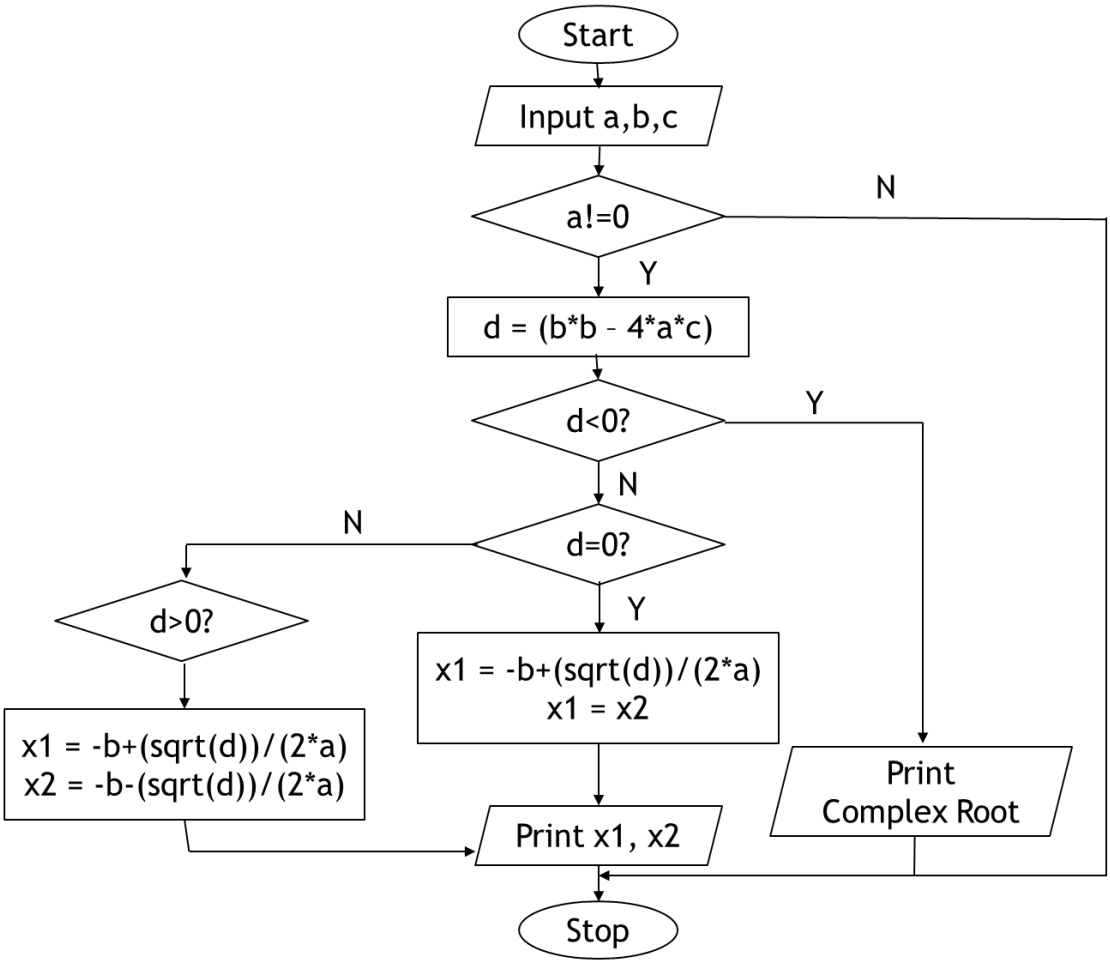
- Input the coefficients (A, B, C) of the quadratic equation
- While (A != 0)
  Calculate D
- Calculate **X1**
- Calculate **X2**
- Print X1 and X2

Algorithm to calculate the roots of a Quadratic Equation:

- Step 1: input A, B, C
- Step 2: WHILE (A != 0)
  {
Step 3: D = B*B – 4*A*C
Step 4: IF (D > 0)
  {
  X1 = -B + (SQRT(D))/(2*A)
  X2 = -B - (SQRT(D))/(2*A)
  PRINT X1, X2
  }
Step 5: ELSEIF (D = = 0)
  {
  X1 = -B + (SQRT(D))/(2*A)
  Print “Equal Roots “

```
                        x2 = x1
                        print x1, x2
                    }
Step 6                  ELSE
                        print " Complex Root "
                    ENDIF
                ENDWHILE
```

Flowchart to calculate the roots of a Quadratic Equation:



Algorithm to calculate the average of a list of n numbers

Input the value of n

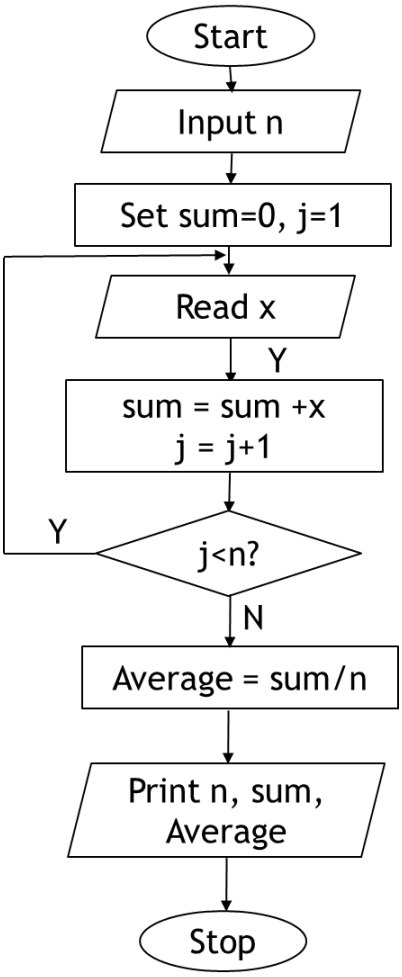
//Initialize variable sum to zero
Sum←0

//Initialize a counter j to 1
j←1
Input the next number on the list, call it x
Add x to sum i.e. sum←sum+x
Increment j by 1 i.e. j←j+1

While j≤n
    Input the next number on the list, call it x
    Add x to sum i.e. sum←sum+x
    Increment j by 1 i.e. j←j+1
End While
average ← sum/n
Print average

STOP

Flowchart to calculate the average of a list of n numbers



Algorithm to divide two integers A and B by method of successive subtraction

Let: A be the dividend
B be the divisor

```
Input A, B
Result←0
While A≥B
        A = A-B
        Result ←Result+1
End while
Print Result
STOP
```

Algorithm to multiply two integers A and B by method of successive addition

```
Input A and B

//Initialize variables
Result←0
Counter←0

While counter ≤ B
        Result ←Result+A
```

```
        Counter←counter+1
End while
Print Result
STOP
```

Algorithm to swap the content of two variables A and B for example, if A = 40 and B = 60, then your algorithm should output the following results A = 60, B = 40

```
Input A,B
Create an extra variable, C
        C←A
        A←B
        B←C
Print A,B
STOP
```

Arrays

An array is a set of data items all of the same type and stored together in the memory; all the data items in an array can therefore be referred to by a single identifier. Array elements can be referenced by use of a subscripted variable. The number of data items in an array is fixed. An array can be one-dimensional or multi-dimensional. E.g.

X(1), X(2), X(3), ... ,X(n)

X(1)	X(2)	X(3)	...	X(4)

A two-dimensional array however is made up of rows and columns of data. A two-dimensional array of four rows and three columns can be depicted as shown below

TABLE (1,1)	TABLE (1,2)	TABLE (1,3)
TABLE (2,1)	TABLE (2,2)	TABLE (2,3)
TABLE (3,1)	TABLE (3,2)	TABLE (3,3)
TABLE (4,1)	TABLE (4,2)	TABLE (4,3)

A multi-dimensional array is specified by giving more than a pair of dimensional limits in the description of the array

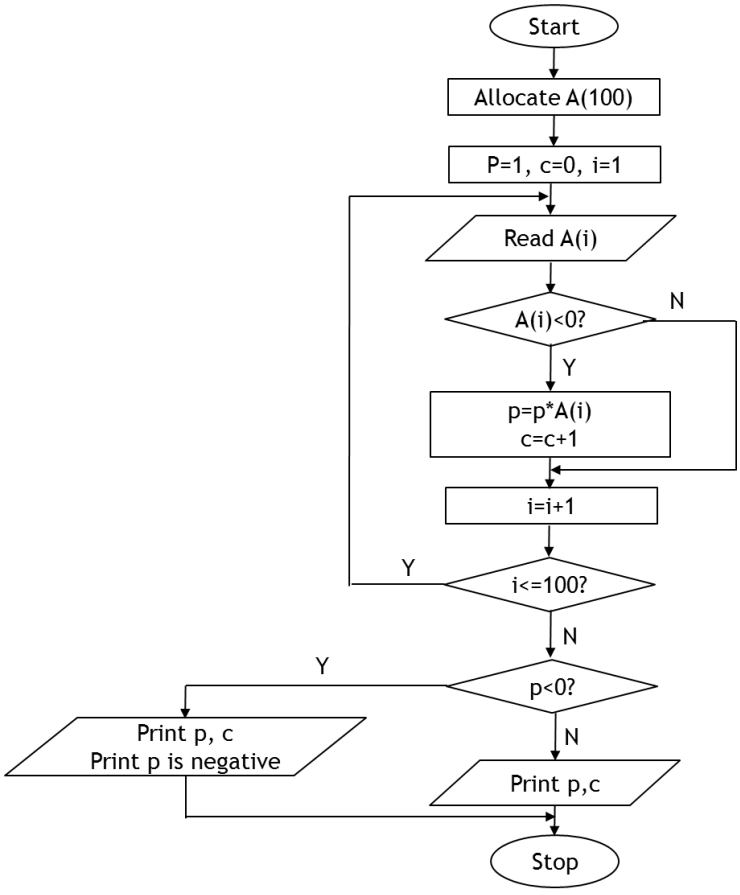
Examples

1. Algorithm to determine the smallest number in a list of n numbers

```
Input n, the number of elements in the list
Input all the numbers into array x
Set smallest to first element in the array i.e. Smallest ← x[1]
Set counter j to 2 i.e j ← 2
While j≤n
   If x[j] < smallest,
         mallest ← x[j]
   End If
   j←j+1
End While
Print smallest
STOP
```

2. Write an algorithm to scan a linear array of 100 elements for negative number, print the number of all such elements found and compute the product of all such number, print it if positive, or print an appropriate message if negative

```
Let:   P be product of all negative numbers
       C be number of negative number in the array
Create A(100)
I = 1, P = 1, C = 0
WHILE I <= 100
   Read A(I)
   IF A(I) < 0 THEN
      P = P * A(I)
      C = C + 1
   END IF
   I = I + 1
End While
IF P > 0
   PRINT P, C
ELSE
       print P, C
       PRINT "P is negative"
END IF
END
```



3. The grades obtained by students in eight courses are to be supplied as input to the computer in addition to the credit unit of the courses. Design an algorithm to compute the Grade Point Average (GPA) of the student. Assume the following: A=5points, B=4points, C=3points, D=2points and F=0point.

```
Set Totalpoint←0, TotalUnit←0
```

```
Set counter j to 1 i.e. j←1
While j≤8
        Input grade[j] and unit[j]
        TotalUnit←TotalUnit+Unit[j]
        If grade[j] = 'A' then
                Totalpoint←TotalUnit+5*unit[j]
                Else if grade[j] = 'B' then Totalpoint←TotalUnit+4*unit[j]
                Else if grade[j] = 'C' then Totalpoint←TotalUnit+3*unit[j]
                Else if grade[j] = 'D' then Totalpoint←TotalUnit+2*unit[j]
                Else if grade[j] = 'F' then Totalpoint←TotalUnit+0*unit[j]
        End If
        j←j+1
End while
GPA←Totalpoint/TotalUnit
Print GPA
STOP
```

2.3.8. DECISION TABLE

A decision table, just like flowchart is a programming and system analysis tool. It can be used to define complex programming logic.

Decision Table format

Table Heading	Decision rules
Condition stub	Condition
Action stub	Action entries

The table is divided into four major parts:

i. Condition stub
ii. Condition entries
iii. Action stub
iv. Action entries

Apart from these, there is also a portion reserved for table heading and another for decision rules. The condition stub will contain conditions to be tested while the action stub will contain actions to be taken after the examination of each rule. The rule itself is a combination of answers to questions asked in the condition stub. If three conditions are entered in the condition stub, then we will expect 8 rules i.e. 2^3. In general, if there are n conditions, there will be 2^N rules. However, some of the rules may be irrelevant or redundant and hence such rules are simply ignored. Realistically speaking then, there will always be less than or equal to 2^N rules for n conditions.

The condition entries are responses to the questions asked under the condition stub and they are usually answered as a 'YES' or a 'NO'.

The action entries contain column by column, the action actually taken in response to the rule in the column. X is put against each action in response to rule.

Examples

1. Candidates are accepted for employment if their qualifications and references are satisfactory and they pass the interview, when a candidate's references or interview (but not both) is unsatisfactory, but the qualifications are satisfactory, he is offered a job for a probationary period.

In all other circumstances, his application is rejected. Construct an appropriate decision table for this policy.

Employment policy	1	2	3	4	5	6	7	8
Qualification OK?	Y	Y	Y	Y	N	N	N	N
References OK?	Y	Y	N	N	Y	N	Y	N
Interview OK?	Y	N	Y	N	Y	Y	N	N
Offer a job	X							
Probationary offer		X	X					
Reject				X	X	X	X	X

2. A finance company applies the following rules in granting loan to its employees: An application for loan should be submitted. The income level of the loan seeker is then checked. If the income is okay and the loan seeker has no further debt with the company, then the application is accepted, otherwise further investigation is carried out. However, if the income is not okay, but the loan seeker has a guarantor, then his application is accepted, otherwise, it is rejected. Draw a decision table representing this policy.

	RULES			
	1	2	3	4
Income OK?	Y	Y	N	N
Any Debt?	N	Y	-	-
Any Guearantor?	-	-	Y	N
Accept Application	X		X	
Reject Application				X
Investigate Further				

2.4. PROGRAMMING WITH QBASIC

INTRODUCTION

BASIC stands for Beginner's All Purpose Symbolic Instruction Code. Invented in 1963, at Dartmouth College, by mathematicians, John George Kemeny and Tom Kurtzas.

BASIC is an interpreter which means it reads every line, translates it and lets the computer execute it before reading another. Each instruction starts with a line number.

Microsoft's Visual Basic language was introduced in the early 1990s to simplify the development of Microsoft Windows applications and is one of the world's most popular programming languages.

Microsoft's latest development tools are part of its corporate-wide strategy for integrating the Internet and the web into computer applications. This strategy is implemented in Microsoft's .NET platform, which provides developers with the capabilities they need to create and run computer applications that can execute on computers distributed across the Internet. Microsoft's three primary programming languages are Visual Basic .NET (based on the original BASIC), Visual C++ .NET (based on C++) and C# (based on C++ and Java, and developed expressly for the .NET platform). Developers using .NET can write software components in the language they are most familiar with then form applications by combining those components with components written in any .NET language.

CONSTANTS AND VARIABLES

Data in QBasic are stored in Constants and Variables.

Constant is an identifier (named memory location) whose associated value cannot typically be altered by the program during its execution (though in some cases this

can be circumvented, e.g. using self-modifying code). Many programming languages make an explicit syntactic distinction between constant and variable symbols

Constants in QBASIC is divided into two types:

1. Numeric Constants: there are 3 types of numeric constants:
   - Real: the numbers used may be written in decimal form such as (6.9, -52.76, 0.095, -3269.0)
   - Integer: Whole numbers may be written without the decimal point such as (89, -132, 7698)
   - Exponential Form: this form requires a number fallowed by the letter E, such as (2.8E05, 0.57E-03, 0.07E-9 and 29.8E7).
2. String Constant: a string consists of a sequence of characters enclosed in double quote marks. strings usually consist of names or address or cities such as "Computer", "Baghdad".

BASIC is not a strongly typed language so constants and variables can be declared dynamically i.e. they don't have to be described with data types before use.
The example below describes a typical constant declaration statement in BASIC.

```
Pie = 3.14
```

Variable is an identifier (named memory location) whose associated value can be altered by the program during its execution. E.g. if you wanted to count the number of times a certain function was called you would use a variable, because it may be called a different number of times each time you run the program. You may not always run that function twice or three times, you may run it as many times as needed. Therefore, you would need to store the value in a variable.

QBASIC keeps track of variables by their names referred to as identifier. If you have a variable representing number of times a crtain function was called, and you called it 'count', QBASIC would not recognize it if you wanted to print the value of it and you referred to it as 'counts'.

Variables Naming Convention

- Variable names cannot begin with a number ('1num' is not a valid variable name while 'num1' is).
- Variable names can only include letters and periods (special characters used in identifying the TYPE of the variable are allowed, but only at the end of the name).
- Variable names CANNOT include spaces.
- Variable names CANNOT be the name of a QBasic command such as PRINT, CLS, END, etc.

SOME DEFINITION

Statement/Command: An instruction passed to the computer to perform a task usually represented in a line of code. These words can be used interchangeably most of the time;

Keyword: A word that is part of the QBasic language. When you type a keyword, QBasic will automatically capitalize it for you. Keywords are used to identify commands and parts of commands. Note that you cannot make a named constant or variable with the same name as a QBasic keyword.

Operator: They are built-in to perform maths operations. They range from

- arithmetic operators (+, -, etc.),
- relational operators (=, >, etc.), and
- Boolean/binary operators (AND, OR).

Block: Generic term for a group of lines inside a structure.
Loop: Generic term for a group of lines executed a series of times.

Our First Program

```
10 CLS
20 ' Hello World program
30 PRINT "Hello, world!"
40 END
```

EXPLAINING THE FIRST PROGRAM

Line 10: CLS – Clear Screen.

CLS Command

Syntax: CLS

The simple **CLS** command clears everything off of the screen and puts the cursor at the top left corner of the screen.

Line 20: is a comment. All comments in **QBasic** begin with an ' (apostrophe) or the keyword **REM** followed by a space.

Comment

Syntax: {**REM** | '} comment

The **REM** command lets you add a comment to your code. As the syntax definition shows, you can use an apostrophe (') in place of the word **REM**. It is good programming convention to add comment while programming

Example:

```
' My first QBASIC program
REM My first QBASIC program
```

Line 30: PRINT "Hello, world!"

PRINT displays text on the screen at the current cursor position. Following the **PRINT** keyword is a literal constant, the text to display. You can **PRINT** just about anything.

PRINT Command

Syntax: PRINT [expression {; | ,} expression {; | ,} ...] [{; | ,}]

The **PRINT** command is used to put text on the screen at the current cursor position. The syntax
will take some explaining. expressioncan be any string or number expression.

Examples: PRINT
PRINT " Name", "**SSN**" **PRINT** "My name is. . . . "; myname$;

Line 40: END

END Command

Syntax: END

The **END** command quits the program and returns to the **QBasic** editor. **Example: END**

Variable Data Types

Every variable used in the program has data type. A variable is created the first time it is referenced in the program.
There are five types of variables.
Each one has its own associated suffix to identify its type.

Data type	Suffix	Description
String	$	String variables are the only variables that hold text
Integer	%	Integer variables are 2 bytes long and hold integers (numbers with no fractional part).
Long Integer	&	Long Integer variables are 4 bytes long and also hold integers.
Single	!	Single-Precision variables are 2 bytes long (usually called Single) can handle numbers with a decimal point.
Double	#	Double-Precision variables are 4 bytes long (usually called Double) can

		also handle numbers with a decimal point.

Library functions

QBasic caters for computational processes that require multiple steps to obtain their desired result and are to be used again and again in the course of the program or by different programmers e.g. computing the square root of a given number, determining the absolute value of an expression, finding the largest value from a set of numbers etc. QBASIC provides a number of built-in functions to cater for such. These built-in functions are called intrinsic functions. Below are some library functions in QBASIC:

Function	Description	Syntax	Examples
ABS	Returns the absolute value of a number.	ABS (numeric expression)	PRINT ABS(45.5–100.0) 'Output is: 54.5
CINT	Rounds a numeric expression to an integer	CINT (numeric-expression)	PRINT CINT(12.49), CINT(12.51) 'Output is: 12 13
CLNG	Rounds a numeric expression to a long (4byte) integer	CLNG(numeric-expression)	PRINT CLNG(338457.8) 'Output is: 338458
CSNG	converts a numeric expression to a single-precision value	CSNG(numeric-expression)	CSNG(975.3421515) 'Output is: 975.3422
CDBL	converts a numeric expression to a double-precision value	CDBL(numeric-expression)	CDBL(1 / 3) 'Output is: .333333333333333
FIX	truncates a floating-point expression to its integer portion	FIX(numeric-expression)	PRINT FIX(12.49), FIX(12.54) 'Output is: 12 12
INT	returns the largest integer less than or equal to a numeric expression	INT(numeric-expression)	PRINT INT(12.54), INT(-99.4) 'Output is: 12 -100
ATN	returns the arctangent of a specified numeric expression	ATN(numeric-expression)	CONST PI=3.141592654 PRINT ATN(TAN(PI/4.0)), PI/4.0 'Output is: .7853981635 .7853981635
SIN	return the sine of a specified angle in radian	SIN(angle)	
COS	return the cosine of a specified angle in radian	COS(angle)	
TAN	return the tangent of a specified angle in radian	TAN(angle)	
EXP	returns e raised to a specified power, where e is the base of natural	EXP(numeric-expression) N.B. For EXP, the numeric expression is a number less than or equal to 88.02969.	PRINT EXP(0), EXP(1) 'Output is: 1 2.718282
LOG	returns the natural logarithm of a numeric expression	LOG(numeric-expression) N.B. For LOG, any positive numeric expression.	PRINT LOG(1), LOG(EXP(1)) 'Output is: 0 1
MOD	Divides one number by another and returns the remainder. numeric-expression1	MOD numeric-expression2 numeric-expression1, numeric-expression2 – Any numeric expressions. Real numbers are rounded to integers.	PRINT 19 MOD 6.7 'QBasic rounds 6.7 to 7, then divides. 'Output is: 5
SQR	Returns the square root of a numeric expression.	SQR(numeric-expression) numeric-expression – A value greater than	PRINT SQR(25), SQR(2) 'Output is: 5 1.414214

		or equal to zero.	
STRING PROCESSING			
INSTR	Returns the position of the first occurrence of a string in another string	INSTR([start%,]string expression1$, stringexpression2$) N.B. start% – Sets the character position where the search begins. If start% is omitted, INSTR starts at position 1. stringexpression1$ – The string to search stringexpression2$ – The string to look for.	a$ = "Microsoft QBasic" PRINT INSTR(1, a$, "QBasic") 'Output is 11
LEFT$ RIGHT$	Return a specified number of leftmost or rightmost characters in a string.	LEFT$(stringexpression$,n%) RIGHT$(stringexpression$,n%) N.B. stringexpression$ – Any string expression. n% – The number of characters to return, beginning with the leftmost or rightmost string character.	a$ = "Microsoft QBasic" PRINT LEFT$(a$, 5) 'Output is: Micro PRINT RIGHT$(a$, 5) 'Output is: Basic
MID$	The MID$ function returns part of a string (a substring). The MID$ statement replaces part of a string variable with another string	MID$ (stringexpression$, start% [,length%]) MID$ (stringvariable$, start% [,length%])=stringexpression$ N.B. stringexpression$ – The string from which the MID$ function returns substring, or the replacement string used by the MID$ statement. It can be any string expression. start% – The position of the first character in the substring being returned or replaced. length% – The number of characters in the substring. If the length is omitted, MID$ returns or replaces all characters to the right of the start position. stringvariable$ – The string variable being modified by the MID$ statement.	a$ = "Where is Paris?" PRINT MID$(a$, 10, 5) 'Output is: Paris Text$ = "Paris, France" PRINT Text$ 'Output is: Paris, France MID$(Text$, 8) = "Texas " PRINT Text$ 'Output is: Paris, Texas
LEN	Returns the number of characters in a string or the number of bytes	LEN(stringexpression$) LEN(variable)	a$ = "Microsoft QBasic" PRINT LEN(a$) 'Ouput is 16

	required to store a variable.	N.B. stringexpression$ – Any string expression. Variable – Any nonstring variable.	

ASSIGNMENT STATEMENT

Variables should be assigned a value, to use it in the program. There are two ways to do this.

LET Command

The LET command assigns a variable a value.

Syntax: [LET] variable = expression

E.g. mystring$ = "This is a test."
LET result% = var1% + var2%

INPUT Command

INPUT lets the user input the value of a variable or variables.

Syntax: INPUT [;] [literalstring$ {; | ,}] var[, var, ...]

E.g. INPUT "What's your name:", n$
INPUT "Enter your phone number:", p$
PRINT n$; ", your number "; p$; " has been nominated for the monthly draw, congratulation!"
END

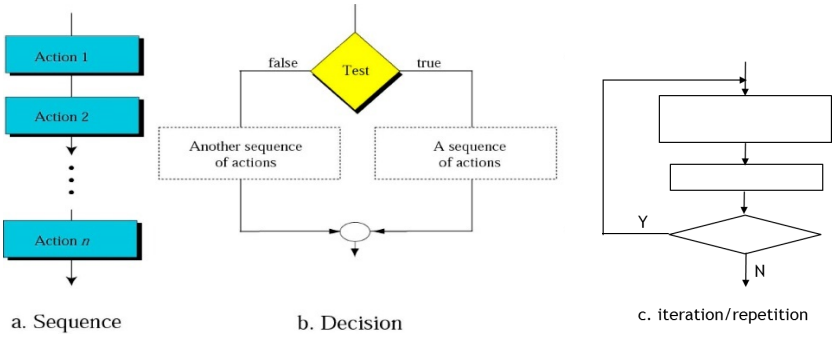
INPUT; var1!

CONTROL STRUCTURE

Control structure is a block of code that dictates the flow of control in a programming language i.e. it chooses a direction in which to go based on given parameters. The term flow control details the direction the program takes.

The three basic control structure are:

- Sequence
- Condition testing/Selection
- Iteration/repetition



a. Sequence b. Decision c. iteration/repetition

Control structure

Condition Testing/Selection

The core to any conditional/selection statement is a Boolean, or true/false, value. If the value is true, it does one thing, if the value is False, it does another thing. We can get T/F values by using the other two kinds of operators: relational and logical (Boolean)/binary. Relational operators are tests between two number values. Basically, you can find how two numbers relate to each other.

NAME	SYMBOL(S)	DESCRIPTION
Equal to	=	Returns true if the two values are equal, and false if not.
Not Equal to	<>, ><	Returns true if the two values are not equal, and false if they are.
Greater than	>	Returns true if the first number is greater than the second, and false if not.
Less than	<	Returns true if the first number is less than the second, and false if not.
Greater than orEqual to	>=, =>	Returns true if the first number is greater than or equal to the second, and false if not.
Less than or Equal to	<=, =<	Returns true if the first number is less than or equal to the second, and false if not.

There are two main commands you can use with conditionals:

- **IF - THEN – ELSE** statement and
- **SELECT CASE.**

Both are similar, but are better suited to some things over others. **IF** can be used wherever **SELECT CASE** can, but not the other way.

IF Command (if-then-else --single-line form)

Syntax:

```
IF condition THEN
        statement
[ELSE ...
        ... statement]
END IF
```

The **IF** command in this form lets you execute a line depending on a condition. Condition is a True/False value. If condition is True (not 0), then the statement following **THEN** is executed. If condition is False (0) and an **ELSE** clause is included, the statement following **ELSE** is executed; or else, flow continues to the next line. This command is small and good if you only need to do one thing based on a condition.

Example:

```
INPUT "Enter a number between 1 and 10:", guess
IF guess > 10 THEN
        PRINT "error"
ELSE
        PRINT "okay"
END IF
END
```

IF Command (If-Elseif-else -- block form)

Syntax:

```
IF condition THEN
        <statements>
[ELSEIF condition THEN
        <statements>
] ...
[ELSE
        <statements>
```

```
]
END IF
```

This form of 'IF' statement is designed for both complex situations and large chunks of code. First, the top condition is tested. If true, the code between it and the next **ELSEIF**, **ELSE**, or **END IF** is run. If it's false, the next **ELSEIF** clause is tested, and so on. If none of the clauses are true, the **ELSE**'s code is run. After going through any chunk of code, flow returns to after the **END IF** line. The **ELSE** clause is optional, and you can have as many **ELSEIF** clauses as you wish.

Note: In large programs we might have a number of blocks inside each other. It's easy to forget the closing statements, and QBasic gives cryptic, confusing errors when you leave a block or loop open. For example, you might get a "Block IF with no **END IF**" error when in fact you forgot to close one of your loops. Even QBasic admits it.

LABELS AND THE GOTO AND GOSUB COMMANDS

The **GOTO** and **GOSUB** commands enable you to jump to certain positions in your program. Labels are used to specify what point in the program to continue execution.

The **GOTO** syntax is as below

```
GOTO <label>
```

e.g.:

```
PRINT "1"
GOTO TheLabel
PRINT "2"
TheLabel:
PRINT "3"
```

Output (notice how **PRINT "2"** is skipped):

1
3

N.B: 'TheLabel' can be placed on the same line as **PRINT "3"**

```
TheLabel: PRINT "3"
```

GOSUB

The **GOSUB** command is the same as **GOTO**, except when it encounters a **RETURN** statement, the program "returns" back to the **GOSUB** command i.e. **RETURN** continues program execution immediately after the previous **GOSUB** statement.

```
PRINT "1"
GOSUB TheLabel
PRINT "2"
END

TheLabel: PRINT "3"
RETURN
```

(Note: The **END** command exits the program.)

Output:

1
3
2

Line numbers: Line numbers can also be used as labels.

```
PRINT "1"
GOTO 10
PRINT "2"
10 PRINT "3" (Notice the line number)
```

You can also write the program like this:

```
10 PRINT "1"
20 GOTO 40
30 PRINT "2"
40 PRINT "3"

10 PRINT "1"
20 GOSUB 40
30 PRINT "2"
END
40 PRINT "3"
RETURN
```

The line numbers don't even have to be in sequence.

```
17 PRINT "1"
2 GOTO 160
701 PRINT "2"
160 PRINT "3"
```

Each of these programs output:

```
1
3
```

SELECT CASE command.

This is preferred when we are only examining the value of one variable throughout the tests.

Syntax:

```
SELECT CASE expression
        CASE {expression1 [, expression2, ...] | IS relational_operator expression | Expression1 TO
        expression2}
        <statements>
[
        CASE (see choices above)
        <statements>
] ...
[
        CASE ELSE
        statements
]
END SELECT
```

This block statement looks at the value of the beginning expression. It checks the first **CASE**. The expression1 [, expression2, ...] form checks to see if the value equals a certain value. The **IS** relational_operator expression form checks to see how it relates to another value. The expression1 **TO** expression2 checks to see if it is between (inclusively) two other values. If it is found to be True, the following block of code is run. If none are found True, the **CASE ELSE** block (if it exists) is run. After a block is run, the program continues after the **END SELECT** keyword.

SELECT CASE is recommended when checking the value of one number, and IF when using multiple variables in your tests. It's all personal preference, though. The next program show how SELECT CASE can be used.

Example:

```
'SELECT CASE demonstration
INPUT "Please enter your marks: ", marks
SELECT CASE marks
CASE IS > 100
        PRINT “Score is invalid”
CASE 70 TO 100
        PRINT "You have an A- grade."
CASE 60 TO 69
        PRINT "You have a B- grade"
CASE 50 TO 59
        PRINT "You got a C - grade."
CASE 45 TO 49
        PRINT "You got a D - grade"
CASE ELSE
        PRINT "You failed."
END SELECT
END
```

Iteration -- Loops

Loops are used when you want your program to do something repeatedly. All of the loop constructs in QBasic execute a block of commands repeatedly 0 or more times.

FOR/NEXT Loop

Syntax:

```
FOR counter = start TO end [STEP increment]
NEXT [counter]
```

Example:

```
FOR I = 1 TO 10 STEP 2
        'Write my name 5 times
        PRINT “Kuldeep”;
NEXT I

CLS
PRINT "PROGRAM: 12 X Tables Square"
PRINT
FOR TIME = 1 TO 12
FOR TABLE = 1 TO 12
        PRINT TIME * TABLE; " ";
NEXT
PRINT
NEXT
END
```

Write a program which uses a FOR/NEXT loop to count from 100 to 20 in steps of -2 and print the number out on each loop.

DO/LOOP

Syntax:

```
DO {WHILE|UNTIL} condition
LOOP
```

OR

```
DO
LOOP {WHILE|UNTIL} condition
```

Note the two different ways of using it. If you put the condition at the beginning, it is evaluated before each loop execution. If you put it with **LOOP** at the end, however, it evaluates it after each execution! This guarantees that, no matter what, the code inside runs at least once. If you use the **WHILE** keyword before the condition, the loop runs as long as condition is true. If you use **UNTIL**, the loop runs as long as condition is false!

Example:

```
DO
        INPUT "Enter the first number: ", A
        INPUT "Enter the second number: ", B
        PRINT "The answer is: "; A * B
        INPUT "Would you like to do it again (y/n)? ", Answer$
LOOP WHILE Answer$="y"

'This example shows one of its best uses: verifying input!

DO 'run the code at least once
        INPUT "Enter a number between 1 and 10. ",num
LOOP UNTIL num> 0 AND num< 11 'wait until it's valid
```

WHILE ... WEND command

Syntax:

```
WHILE (condition)
        Statement
        Statement 2
        .
        .
        .
        Statement n
WEND
```

Program to calculate factorial for a given number n is as follow:

```
LET nfac = 1
INPUT "Enter number", n
IF n = 0 THEN
   nfac = 1
END IF
WHILE n > 0
   nfac = nfac * n
   n = n - 1
WEND
PRINT nfac
```

EXIT Command

Syntax:

EXIT {FOR | DO}

The EXIT command lets you break out of a FOR/NEXT or DO/LOOP construct in the middle of
the code. This would be used when you must stop the loop prematurely. For example, if you
have a FOR/NEXT going from 1 to 10 and you need to end it early because another condition is
true, you can use EXIT FOR to stop the loop.

Example:

```
DO
    INPUT "Please enter a number between 1 and 10. ",num
    IF num> 0 AND num< 11 THEN EXIT DO
    PRINT "Hey! Can't you read?"
LOOP

'note that the DO/LOOP can be used without a condition at all,'resulting in an infinite loop.
EXIT is the only way to break such a loop.
```

Other examples

Divide two integers A and B by method of successive subtraction

```
INPUT "Value of dividend: ", A
INPUT "enter value of divisor: ", B
PRINT A, "divided by", B, "is:";
WHILE A >= B
  A = A - B
  Division = Division + 1
WEND
IF A > 0 THEN
  fraction = A / B
  Answer = Division + fraction
  PRINT Answer
ELSE
  PRINT Division
END IF
```

Program to calculate the average of a list of n numbers

```
INPUT "How many numbers are in the List: ", n
CLS
FOR j = 1 TO n
  INPUT "Enter the Number: ", x
  CLS
  sum = sum + x
NEXT j
average = sum / n
PRINT "The average of the", n, "numbers is:", average
END
```

Design an algorithm that will receive a positive integer as input and find the sum of the digit that make up the integer e.g. if the integer is 324, then sum = 3+2+4 = 9

```
INPUT "number;", n
```

```
sum = 0
WHILE n > 0
   c = n MOD 10
   PRINT c
   sum = sum + c
   n = FIX(n / 10)
WEND
PRINT sum
END
```

Qbasic program to scan a string, check if it is a palindrome and print appropriate message

```
DIM Wrd AS STRING
DIM RevWrd AS STRING
DIM x AS INTEGER

CLS

INPUT "Enter Word: ", Wrd

FOR x = LEN(Wrd) TO 1 STEP -1
   RevWrd = RevWrd + MID$(Wrd, x, 1)
NEXT x

PRINT
PRINT "Original Word: "; LCASE$(Wrd)
PRINT "Reverse Word: "; LCASE$(RevWrd)

IF LCASE$(Wrd) = LCASE$(RevWrd) THEN
   PRINT "The Word Is A Palindrome"
ELSE
   PRINT "The Word Is Not A Palindrome"
END IF
```

Arrays

Arrays hold lists of variables of the same data type and reference them as subscripted variables. When there are large lists of variables and data, it is easier to contain the data in an array than have large amounts of separate variables to hold the data.

In QBASIC, an array should be declared before use. An array may either be declared implicitly or explicitly.

The syntax for array declaration is:

```
DIM array_name(number_of_items) [AS datatype]
```

Examples

Program to scan a linear array of 100 elements for negative number, print the number of all such elements found and compute the product of all such number, print it if positive, or print an appropriate message if negative

```
DIM A(10)
LET i = 1
p = 1
c = 0
WHILE i <= 10
   INPUT "enter the values: ", A(i)
   IF A(i) < 0 THEN
      p = p * A(i)
      c = c + 1
   END IF
   i = i + 1
WEND
IF p > 0 THEN
   PRINT "The product of the negative numbers is: ", p
   PRINT "The total number of negative numbers entered is ", c
ELSE
   PRINT "The product of the negative numbers is: ", p
   PRINT "The total number of negative numbers entered is ", c
   PRINT "the value is negative"
END IF
END
```

Program to determine the smallest number in a list of n numbers

```
INPUT "How many numbers are in the list: ", n
FOR j = 1 TO n
   INPUT "Enter Number: ", Numbers(j)
NEXT j
largest = Numbers(1)
Smallest = Numbers(1)
FOR j = 2 TO n
   IF Numbers(j) > largest THEN
      largest = Numbers(j)
   ELSEIF Numbers(j) < Smallest THEN
      Smallest = Numbers(j)
   END IF
NEXT j
CLS
PRINT "Array Numbers:"
FOR j = 1 TO n
   PRINT Numbers(j); " ";
NEXT j
PRINT "The Highest Number: "; largest
PRINT "The Lowest Number: "; Smallest
```

The grades obtained by students in eight courses are to be supplied as input to the computer in addition to the credit unit of the courses. Design an algorithm to compute the Grade Point Average (GPA) of the student. Assume the following: A=5points, B=4points, C=3points, D=2points and F=0 point

```
totalpoint = 0 totalunit = 0, j = 1
WHILE j <= 5
   INPUT "enter course: ", course$(j)
   INPUT "enter unit: ", unit(j)
```

```
   INPUT "enter grade scored: ", grade$(j)
   totalunit = totalunit + unit(j)
   IF grade$ = "A" THEN
      totalpoint = totalunit + 5 * unit(j)
   ELSEIF grade$ = "B" THEN
         totalpoint = totalunit + 4 * unit(j)
      ELSEIF grade$ = "C" THEN
            totalpoint = totalunit + 3 * unit(j)
         ELSEIF grade$ = "D" THEN
               totalpoint = totalunit + 2 * unit(j)
            ELSEIF grade$ = "F" THEN
                  totalpoint = totalunit + 0 * unit(j)
      END IF
j = j + 1
WEND
PRINT totalunit
PRINT totalpoint
GPA = totalpoint / totalunit
PRINT "GPA =", GPA
```

Consider a sequence of real numbers X_i, 1, 2,,..., m. The mean is defined as $\bar{X}=((X1+X2+...+Xm))/m$

The deviation about the mean is:

$$d\_i = (X\_i - \bar{X}),\ i=1, 2,3,..., m$$

And the standard deviation is

$$S= \sqrt{(((d\_1^2+d\_2^2+...+d\_m^2))/m)}$$

Write a program to calculate the standard deviation of the numbers

```
INPUT "How many sequence real numbers are to be considered: ", m
FOR j = 1 TO m
   INPUT "Enter the numbers: ", X(j)
   sum = sum + X(j)
NEXT j
Mean = sum / m
CLS
FOR j = 1 TO m
   dev(j) = X(j) - Mean
   PRINT "The deviation around the mean for", X(j), "is", dev(j)
   devsum = devsum + (dev(j) * dev(j))
NEXT j
stddev = SQR(devsum / m)
PRINT "The standard Deviation for the", m, "numbers is:", stddev
END
```

Functions and Subroutines

A subroutine is a functional unit of code which may be called and implemented anywhere in an program and takes zero or more values as input, act upon that data to give output to the calling routine. Invoked with the CALL statement

A function is similar to a subroutine but must return at least one value to the calling routine. Invoked by placing the function name and its associated argument in an expression

```
FOR x = 1 TO 3
   CALL GetText
```

```
NEXT x

SUB GetText
PRINT "Enter some text:";
INPUT text$
PRINT "The text you entered was: "; text$
END SUB
```

Functions have to return a value,
In QBASIC, set a variable with the same name as the function.

```
PRINT Add(10, 7)
FUNCTION Add (num1, num2)
        Add = num1 + num2
END FUNCTION


PRINT Add$("Hello", "World")
FUNCTION Add$ (str1$, str2$)
        Add$ = str1$ + str2$
END FUNCTION
```

Implementing factorial using a function

```
FOR i = 0 TO 3:
   INPUT "Enter number: ", num
   PRINT num; "! ="; nfac(num) || PRINT STR$(num) + "! =" + STR$(nfac(num))
NEXT i
END

FUNCTION nfac (n)
nfac = 1
IF n = 0 THEN
   nfac = 1
END IF
WHILE n > 0
   nfac = nfac * n
   n = n - 1
WEND
END FUNCTION
```

Implementing factorial using a subroutine

```
WHILE i <= 2
   INPUT "Enter number", num
   PRINT num; "! =";
   CALL fact(num)
   i = i + 1
WEND

SUB fact (n)
nfac = 1
IF n = 0 THEN
   nfac = 1
```

```
END IF
WHILE n > 0
   nfac = nfac * n
   n = n - 1
WEND
PRINT nfac
END SUB
```

The fibonacci sequence is a series of numbers in which each number (Fibonacci number) is the sum of the two preceding numbers. Mathematically, the sequence n of Fibonacci numbers is defined by the recurrence relation $Fn=Fn-1+Fn-2$,

```
a = 1
b = 2
PRINT a, b,
FOR i = 1 TO 8
   c = a + b
   PRINT c,
   a = b
   b = c
NEXT i
END
```

Implementing Fibonacci in a Subroutine

```
INPUT "Enter the number of Sequence: ", num
CALL fibonacci(num)

SUB fibonacci (n)
INPUT "Enter the first term", a
INPUT "Enter the second term", b
PRINT a; b;
IF n > 0 THEN
   FOR i = 1 TO n
      c = a + b
      PRINT c;
      a = b
      b = c
   NEXT i
END IF
END SUB
```

Problem Solving Using Recursion

Recursion is the process of solving a problem by reducing it to small version of itself. Recursion is a powerful way to solve certain problems for which the solution would otherwise be very complicated.

Recursive definition

A recursive definition is a definition in which something is defined in terms of a smaller version of itself. In mathematics for example, the factorial of an integer is defined as follows:

0! = 1	2.0
n! = n x (n – 1)! If n > 0	2.1

In this definition:
0! is defined to be 1, and if n is an integer greater than 0, first we find (n – 1)! And then multiply it by n. To find (n – 1)!, we apply the definition again. If (n – 1) > 0, then we use Equation 2.1., otherwise, we use Equation 2.0. Thus, for an integer n greater than 0, n! is obtained by first finding (n – 1)! (that is n! is reduced to a smaller version of itself) and then multiplying (n – 1)! by n.

Note that the solution in Equation 2.0 is direct – that is, the right side of the equation contains no factorial notation. The solution in Equation 2.1 is given in terms of a smaller version of itself. The definition of the factorial as given in Equations 2.0 and 2.1 is called a recursive definition. Equation 2.0. is called the base case, the case for which the solution is obtained directly. Equation 2.1 is called the general case or recursive case.

It is clear from this example that:

(i) every recursive definition must have one (or more) base cases
(ii) the general case must eventually be reduced to a base case.
(iii) the base case stops the recursion.

The concept of recursion in computer science works similarly. In computer science, we talk about recursive algorithm and recursive methods. An algorithm that finds the solution to a given problem by reducing the problem to smaller versions of itself is called a recursive algorithm. The recursive algorithm must have one or more base cases, and the general solution must eventually be reduced to base case.

A method that calls itself is called a recursive method. That is, the body of the recursive method contains a statement that causes the same method to execute before completing the current call. Recursive algorithms are implemented using recursive methods.

In what follows, we write the recursive method that implements the factorial definition.

```
FUNCTION factorial (n)
IF n = 0 THEN
   factorial = 1
ELSE
   factorial = n * factorial(n - 1)
END IF
END FUNCTION
```

Direct and Indirect Recursion

A method is called directly recursive if it calls itself. A method that calls another method and eventually results in the original method call is called is called indirectly recursive. For example, If method A calls method B and method B calls method A, then method A is indirectly recursive. Indirect recursion could be several layers deep. For example, If method A calls method B, method B calls method C, method C calls method D, and method D calls method A, then method A is indirectly recursive.

Indirect recursion requires the same careful analysis as direct recursion. The base case must be identified and appropriate solutions must be provided to them. However, tracing through indirect recursion can be a tedious process. Therefore, extra care must be exercise when designing indirect recursion methods.

A recursive method in which the last statement executed is the recursive call is called a tail recursive method. The method factorial is an example of a tail recursive method.

Infinite Recursion

A finite recursion occurs when recursive call reaches a call that make no further recursive calls, that is the sequence of recursive calls eventually reach a base case. However, if every recursive call results in another recursive call, then the recursive method (algorithm) is said to have infinite recursion. In theory, infinite recursion executes forever. Every call to a recursive method requires the system to allocate memory for the local variables and formal parameters. In addition, the system also saves the information so that after completing a call, control can be transferred back to the right caller. Therefore, because computer memory is finite, if you execute an infinite recursive method on a computer, the method will execute until the system runs out of memory, which results in an abnormal termination of the program.

Recursive methods (algorithms) must be carefully designed and analyzed. You must make sure that every recursive call eventually reduces to a base case. The following sections give various examples illustrating how to design and implement recursive algorithms.

To design a recursive method, you must:

1. Understand the problem requirements.
2. Determine the limiting conditions. For example, for a list, the limiting condition is determined by the number of elements in the list.
3. Identify the base cases and provide a direct solution to each base case.
4. Identify the general cases and provide a solution to each general case in terms of a smaller version of itself.

Recursion: some examples

This section presents examples on how recursive algorithms are developed and implemented in qbasic using recursive methods.

Fibonacci Number

In this example, we write a recursive method, rFibNum to determine the desired Fibonacci number. The method rFibNum takes as parameters three numbers representing the first two numbers of the Fibonacci sequence and a number n, the desire n^{TH} Fibonacci number. The method rFibNum returns the n^{TH} Fibonacci number in the sequence.
Recall that the third Fibonacci number is the sum of the of the first two Fibonacci numbers. The fourth Fibonacci number in a sequence is the sum of the second and third Fibonacci numbers. Therefore, to calculate the fourth Fibonacci number, we add the second Fibonacci number and the third Fibonacci number (which itself is the sum of the first two Fibonacci numbers). The following recursive algorithm calculates the n^{TH} Fibonacci number, where a denotes the first Fibonacci number, b the second Fibonacci number, and n the n^{TH}
Fibonacci number:

$$\text{rFibNum}(a, b, n) = \begin{cases} a & \text{If } n = 1 \\ b & \text{If } n = 2 \\ \text{rFibNum}(a,b,n-1) + \text{rFibNum}(a,b,n-2) & \text{if } n > 2 \end{cases}$$

The following recursive method implements the Fibonacci algorithm:

```
REM Nth term = (N-1)th term + (N-2)th term;
INPUT "How many terms do you want to calculate: ", n
DECLARE FUNCTION Fibonacci (n)
FOR counter = 1 TO n
```

```
   PRINT fibonacci(counter);
NEXT counter

REM Function to calculate Nth Fibonacci number
REM Fibonacci(N) = fibonacci(N - 1) + fibonacci(N - 2);

FUNCTION fibonacci (n)
IF (n = 0 OR n = 1) THEN
   fibonacci = 1
ELSE
   fibonacci = (fibonacci(n - 1) + fibonacci(n - 2))
END IF
END FUNCTION
```

Review Exercise

1. Mark the following statements as true or false.
   a. Every recursive definition must have one or more base cases.
   b. Every recursive method must have one or more base cases.
   c. The general case stops the recursion.
   d. In the general case, the solutions to the problem is obtained directly.
   e. A recursive method always returns a value.
2. What is a base case?
3. What is a recursive case/
4. What is direct recursion?
5. What is indirect recursion?
6. What is tail recursion?
7. Consider the following recursive method:

```
FUNCTION myMethod(number)
If (number = 0)
        myMethod = number;
else
        myMethod = number + myMethod(number - 1));
```

   a. Identify the base case.
   b. Identify the general case.
   c. What valid values can be passed as parameters to the method myMethod?
   d. If myMethod(0) is a valid call, what is its value? If not, explain why?
   e. If myMethod(5) is a valid call, what is its value? If not, explain why?
   f. If myMethod(-3) is a valid call, what is its value? If not, explain why?

8. What is problem solving?
9. List and discuss six(6) steps you should follow in order to solve a problem
10. What is an algorithm?
11. State the properties of an algorithm.
12. Comment on each of the following:
    (a) Problem solving called recursion.
    (b) Direct and Indirect Recursion.
    (c) Infinite Recursion.

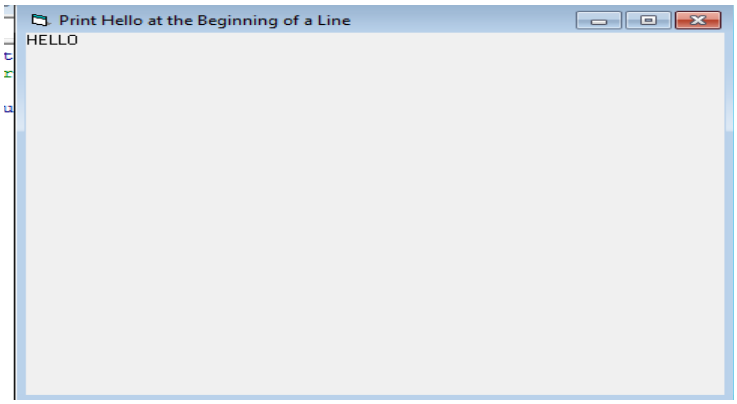
Laboratory Problems

A Program to print HELLO at the Beginning of a Line

Source Code

```
Private Sub form_Activate()
'A Program to print Hello at the Beginning of a Line
Print "HELLO"
End Sub
```

Object Screen

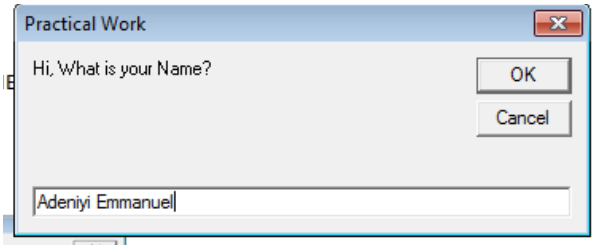


A Program to accept the name of the User and print Welcome (name) LET'S BE FRIENDS

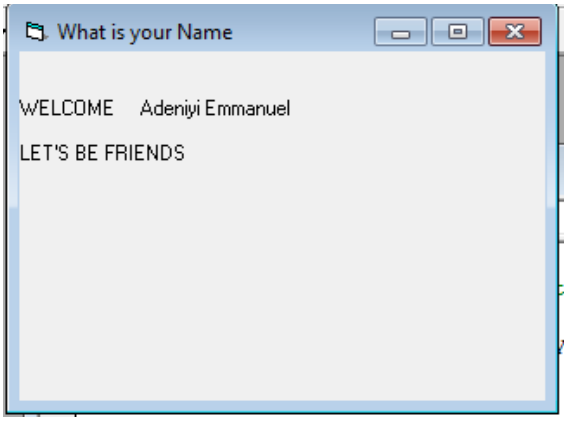
Source Code

```
Private Sub Form_Activate()
' A program to print the name of the User
    Dim Name As String
    Name = InputBox("Hi, What is your Name? ", "Practical Work")

    Print vbNewLine
    Print "WELCOME ", Name
    Print
    Print "LET'S BE FRIENDS"
End Sub
```

Object Screen



A

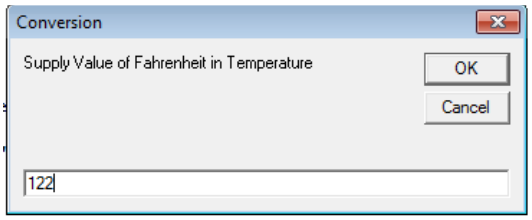


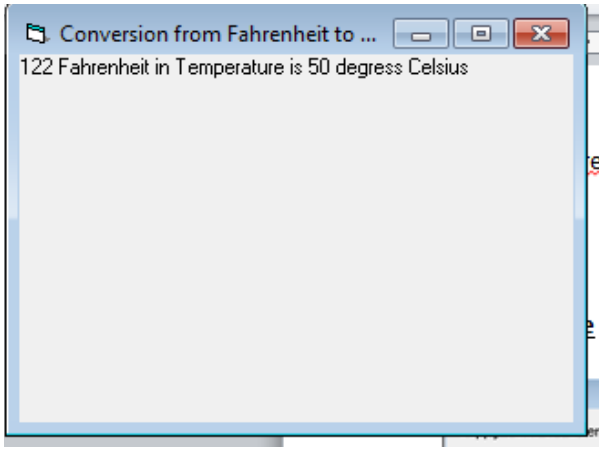
program to Convert Temperature reading in Fahrenheit

Source Code

```
Private Sub Form_Activate()
' A Program to Convert Fahrenheit to Celsius
Dim Celsius, Fehre As Single
   Fahre = InputBox("Supply Value of Fahrenheit in Temperature", "Conversion")
      Celsius = 0.05556 * (Fahre - 32)
   Print Fahre & " Fahrenheit in Temperature is " & Celsius & " degress Celsius"
   End Sub
```

Object Screen





A Program to Compute Volume and Area of a Sphere

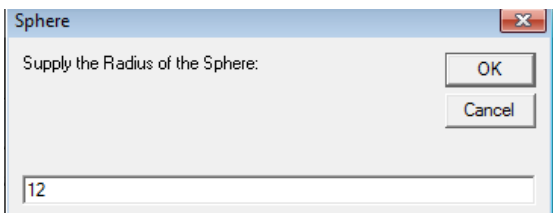
Source Code

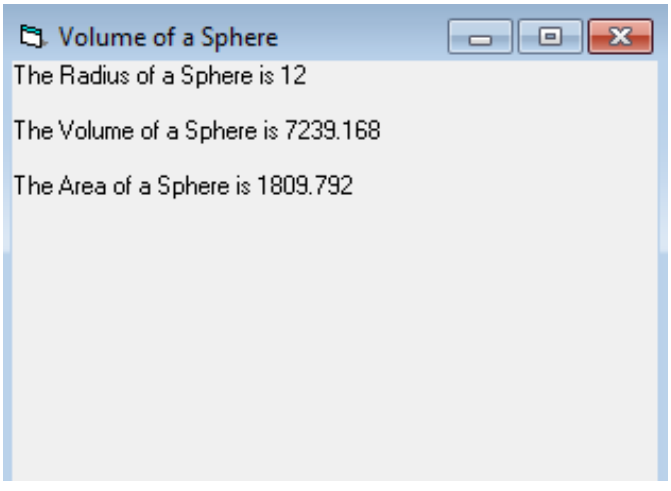
```
Private Sub Form_Activate()
' Program to calculate the volume and area of a sphere using the formular V=(4*pie*r^3)/3
   Dim Pie, Volume, Area, Radius As Single
      ' Assign Value to Pie
      Let Pie = 3.142
      ' Accept Value for Radius
            Radius = InputBox("Supply the Radius of the Sphere: ", "Sphere")
             ' Compute Volume and Area
               Volume = (4 * Pie * Radius ^ 3) / 3
               Area = 4 * Pie * Radius ^ 2
      'Output computed Values
   Print "The Radius of a Sphere is " & Radius & vbNewLine
   Print "The Volume of a Sphere is " & Volume & vbNewLine
   Print "The Area of a Sphere is " &
```

Area

End Sub

Object Screen



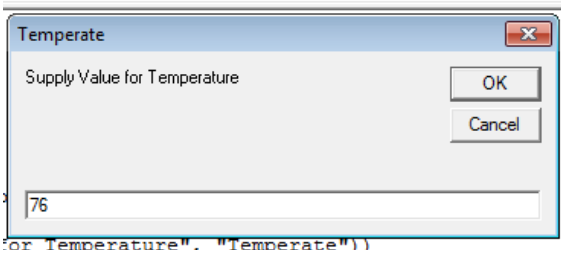


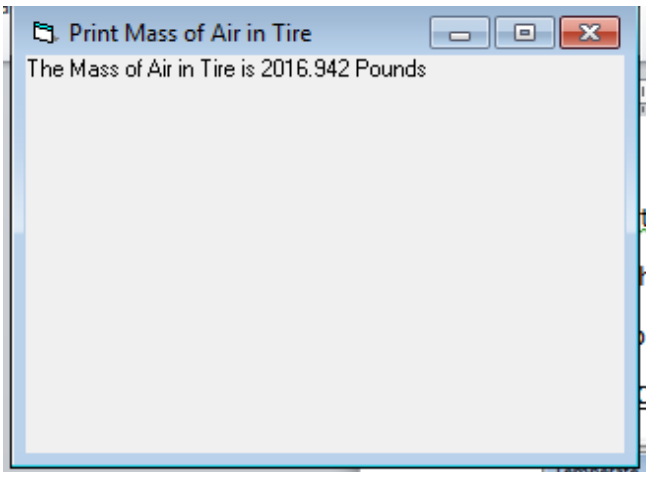
A program to calculate the Mass of Air in an automobile tire

Source Code

```
Private Sub Form_Activate()
' A Program to find the Mass of Air in Tire

Dim Temperature, Pressure, Volume, Mass As Single
   ' Accept value for Pressure
      Pressure = InputBox("Supply Value for Pressure", "Pressure")
   'Accept value for Volume
      Volume = InputBox("Supply value for Volume", "Volume")
   'Accept Value for Temperature
      Temperature = InputBox("Supply Value for Temperature", "Temperate")
         'Compute Mass of Air in Tire
      Mass = (Pressure * Volume) / (0.37 * (Temperature + 460))
' Output the mass in pounds
Print "The Mass of Air in Tire is " & Mass & " Pounds"
End Sub
```

Object Screen





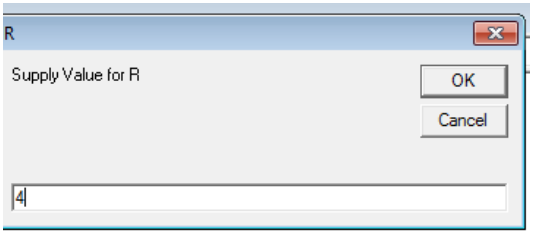
A program to Calculate $\frac{n!}{(n-r)!r!}$ where n and r are positive integers such that n >= r

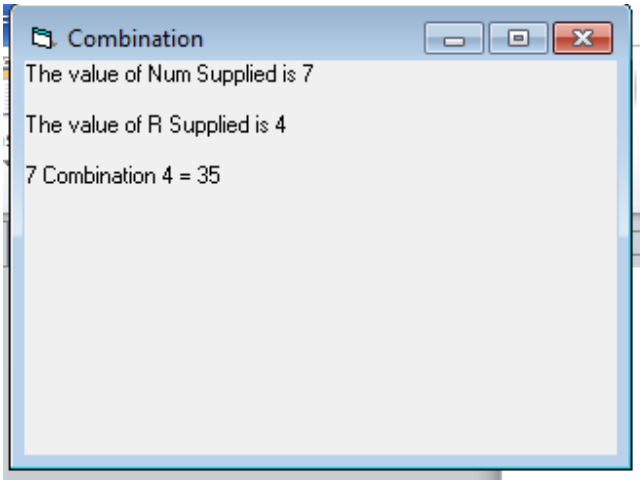
Source Code

```
Private Sub Form_Activate()
' A program to calculate N factorial
Dim Num, R, Kounter, Factorial As Integer
Dim I, J, Comb, Nfact, Pfact As Integer
   Let Factorial = 1
   Let Nfact = 1
   Let Pfact = 1
   Num = InputBox("Supply Value for Num", "Factorail")
   R = InputBox("Supply Value for R ", "R")
  ' Test the value supply for Num and R
   If Num < R Then
      Print "Invalid Number Supply"
   Exit Sub
```

```
    End If
    ' Substract R from Num
       Temp = Num - R
   ' Calculate Num Factorial
       For Kounter = 1 To Num
          Factorial = Factorial * Kounter
       Next
    'Print Num & "! = " & Factorial
    ' Calculate Temp factorial
       For I = 1 To Temp
          Nfact = Nfact * I
       Next
    'Calculate R Factorial
       For J = 1 To R
          Pfact = Pfact * J
       Next
    'Compute the Combination
       Comb = Factorial / (Nfact * Pfact)
    'Output the Answer
       Print "The value of Num Supplied is " & Num & vbCrLf
       Print "The value of R Supplied is " & R & vbCrLf
       Print Num & " Combination " & R & " = " & Comb
End Sub
```

Object Screen





A program to expand binomial expression with the assumed p=0.3333, q=0.6667 and n=7.

Source Code

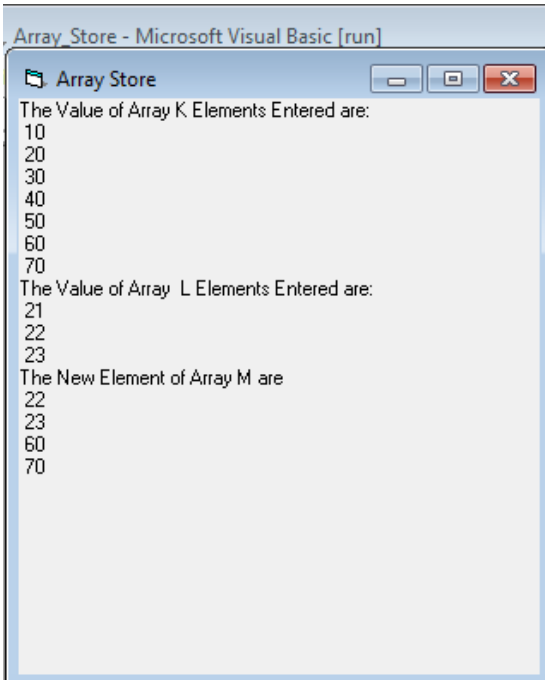
A program to Store Array K, store array L in locations occupied by the k elements 30, 40, 50 and write out the last four elements of k as a one-dimension array M

Source Code

```
Private Sub Form_Activate()
'Array Declaration
Dim K(7) As Integer
Dim L(3) As Integer
Dim M(4) As Integer

    For I = 1 To 7
    K(I) = InputBox("Supply Value of Array K " & I, "Array Input")
Next
    'Display Headling
Print "The Value of Array K Elements Entered are: "
For J = 1 To 7
```

```
    Print K(J)
Next
    'Print vbCrLf
For I = 1 To 3
    L(I) = InputBox("Supply Value of Array L " & I, "Array Input")
Next

'Display Headling
Print "The Value of Array  L Elements Entered are: "
For J = 1 To 3
    Print L(J)
Next
K(3) = 21
K(4) = 22
K(5) = 23
    J = 1
          Print "The New Element of Array M are "
For F = 4 To 7
   M(J) = K(F)
   Print M(J)
    J = J + 1
Next
End Sub
```

Object Screen



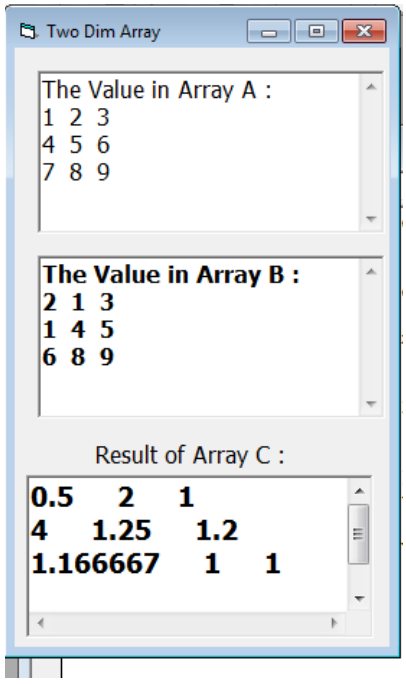
A Program to Store different sets of array a and b respectively.

Source Code

```
Private Sub Form_Activate()
'Array Declaration
Dim A(23, 23) As Integer
Dim B(23, 23) As Integer
Dim C(23, 23) As Single

    Dim Row, Col As Integer

        'Enter Value of Row and Columns
```

```
    Row = Val(InputBox("Enter Number of Row ", "Row"))
    Col = Val(InputBox("Enter Number of Columns ", "Columns"))

     'Supply Values into Elements of Array Row and Columns
      For I = 1 To Row
         For K = 1 To Col
            A(I, K) = Val(InputBox("Supply Values in Array A " & K, "Array Input"))
         Next K
      Next I

       For I = 1 To Row
         For K = 1 To Col
            B(I, K) = Val(InputBox("Supply Values in Array B " & K, "Array Input"))
         Next K
      Next I

      'Display Array A
       txtA.Text = "The Value in Array A : " & vbCrLf
      For M = 1 To Row
         For N = 1 To Col
         txtA.Text = txtA.Text & A(M, N) & Space(2)
         Next N
        txtA.Text = txtA.Text & vbCrLf
      Next M
               'Display Array B
         txtDisplay.Text = "The Value in Array B : " & vbCrLf
      For M = 1 To Row
         For N = 1 To Col
         txtDisplay.Text = txtDisplay.Text & B(M, N) & Space(2)
         Next N
        txtDisplay.Text = txtDisplay.Text & vbCrLf
      Next M
    'Compute and Output Result of Elements C
  For M = 1 To Row
    For N = 1 To Col
       C(M, N) = A(M, N) / B(M, N)
    txtC.Text = txtC.Text & C(M, N) & Space(5)
    Next N
  txtC.Text = txtC.Text & vbCrLf
  Next M
End Sub
```

Object Screen

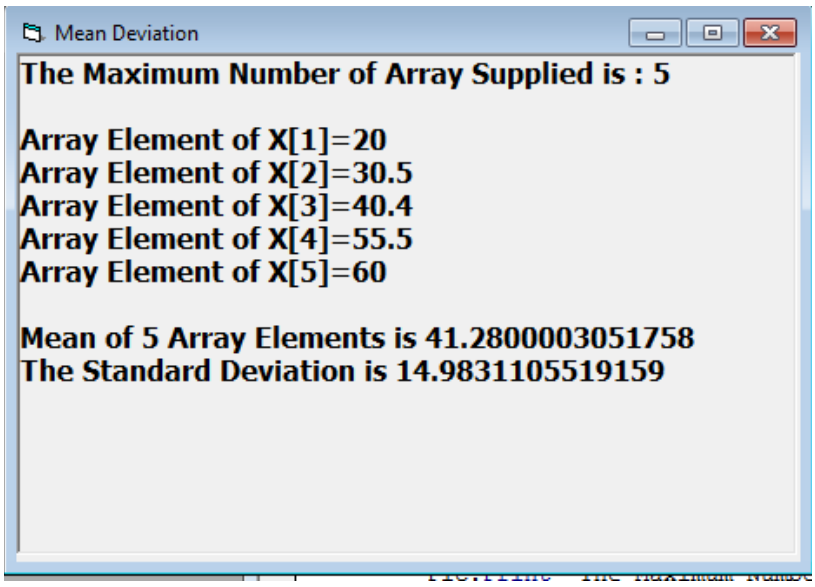


A Program to calculate deviation about the mean.

Source Code

```
Private Sub Form_Activate()
'Calculate the Standard Deviation and Mean Deviation

Dim Mean, SD, Sum, DevSum, Dev As Single
Dim X(100) As Single
Dim M As Integer
   Sum = 0
   DevSum = 0
      M = InputBox("Enter the Maximum Number of Elements of Array : ", "Standard Deviation")
         For I = 1 To M
            X(I) = InputBox("Values of Linear Array : ", "Element of Array")
         Sum = Sum + X(I)
         Next I
      'Compute the Mean
      Mean = Sum / M
            'Compute the Deviation
         For I = 1 To M
            Dev = X(I) - Mean
            DevSum = DevSum + (Dev * Dev)
         Next
      'Compute Standard Deviation
      SD = Sqr(DevSum / M)
            'Display the Computed Results
      Pic.Print "The Maximum Number of Array Supplied is : " & M & vbNewLine
      For I = 1 To M
         Pic.Print "Array Element of X[" & I & "]=" & X(I)
      Next
      Pic.Print
      Pic.Print "Mean of " & M & " Array Elements is " & Mean
      Pic.Print "The Standard Deviation is " & SD

End Sub
```

Screen Product

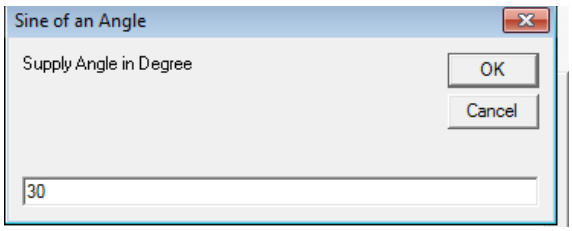


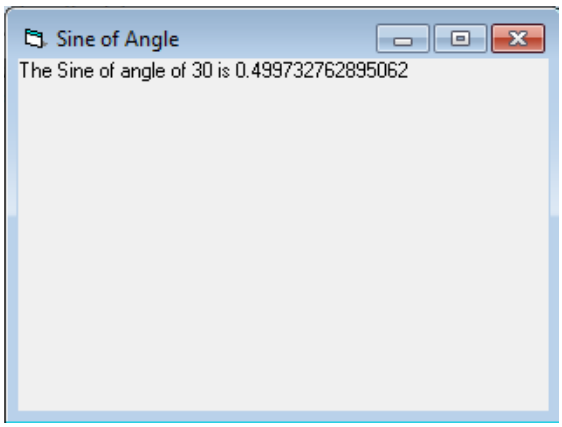
A program to compute Sine of x by summing the first n term of the infinite series sin(x) = x-x^3/3!+x^5/5!-x^7/7!+….

Source Code

```
Private Sub Form_Activate()
' A Program to Calculate the Sine of X

   Let Pie = 3.142
   Let ErrorLimit = 0.00001
      Dim Term, Sum, X, Angle As Single
         Dim Denum As Integer
            Angle = Val(InputBox("Supply Angle in Degree ", "Sine of an Angle"))
               X = (Angle * Pie) / 180
                  Term = X
                     Denum = 1
                     Sum = Term
                     'For I = 1 To 499
                        'If I < 499 Then
         Do While (Term >= ErroLimit)
               Denum = Denum + 2
            Term = Term * (-(X * X)) / (Denum * (Denum - 1))
         Sum = Sum + Term
         Loop
         'End If
         'Next
   Print "The Sine of angle of " & Angle & " is " & Sum
End Sub
```

Screen Result





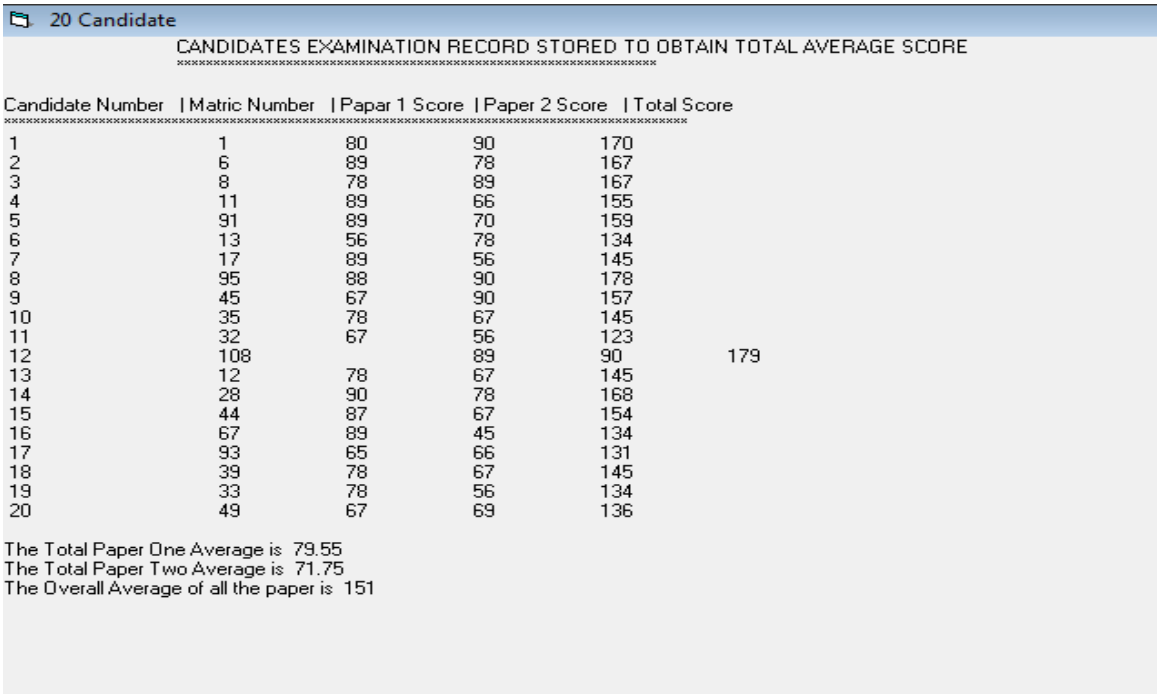
A program which print the total marks obtained by each candidate over 2 exams paper. The program should also print the overall average mark for paper1, paper2 and overall total average mark.

Source Code

```
Private Sub Form_Activate()
'A Program to Store Records of 20 Candidate

   Dim Candidate(20), Paper1(20), Paper2(20) As Integer
   Dim Paper1_Avg, Paper2_Avg, Overall_Avg As Integer
   Dim TotalMark_Paper1, TotalMark_Paper2, Kounter, Total(20), Overall_Total, MatricNo(20) As Integer
   TotalMark_Paper1 = 0
   TotalMark_Paper2 = 0
   Overall_Total = 0
      'Start the Computation
```

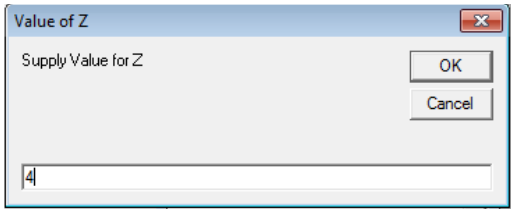
```
    For Kounter = 1 To 20
       MatricNo(Kounter) = (InputBox("Supply Student Matric Number for Candidate " & Kounter,
"Candidate Record"))
       Paper1(Kounter) = (InputBox("Enter Mark Obtained in Paper One", "Paper One"))
       Paper2(Kounter) = (InputBox("Enter Mark Obtained in Paper Two", "Paper Two"))
       Total(Kounter) = Paper1(Kounter) + Paper2(Kounter)
       Overall_Total = Overall_Total + Total(Kounter)
       TotalMark_Paper1 = TotalMark_Paper1 + Paper1(Kounter)
       TotalMark_Paper2 = TotalMark_Paper2 + Paper2(Kounter)

    Next

    'Display the Output




    Print Tab(20); "CANDIDATES EXAMINATION RECORD STORED TO OBTAIN TOTAL
AVERAGE SCORE"
    Print Tab(20); "*************************************************************"
    Print
    Print "Candidate Number   | Matric Number   | Papar 1 Score  | Paper 2 Score   | Total Score"
    Print
"*************************************************************************************
********"

    For Kounter = 1 To 20
       Print Kounter, Space(15); MatricNo(Kounter), Space(15); Paper1(Kounter), Space(15);
Paper2(Kounter), Space(15); Total(Kounter)
       'Print MatricNo(Kounter)
       'Print Paper1(Kounter)
       'Print Paper2(Kounter)
       'Print Total(Kounter)
    Next
    Paper1_Avg = TotalMark_Paper1 / 20
    Paper2_Avg = TotalMark_Paper2 / 20
    Overall_Avg = Overall_Total / 20

    Print
    Print "The Total Paper One Average is "; Paper1_Avg
    Print "The Total Paper Two Average is "; Paper2_Avg
    Print "The Overall Average of all the paper is "; Overall_Avg
End Sub
```

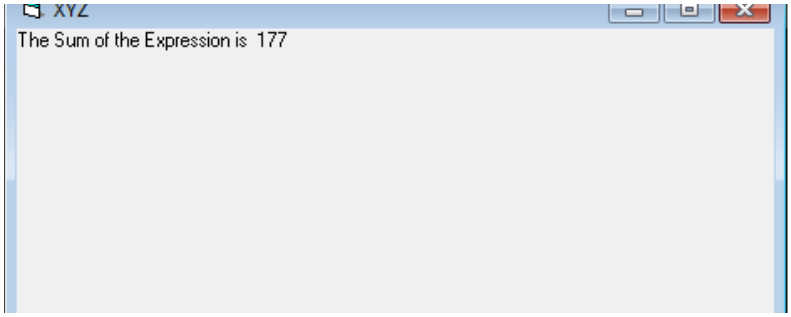
Object Screen



A Program to evaluate the expression $X^5 + Y^4 + Z^3$

Source Code





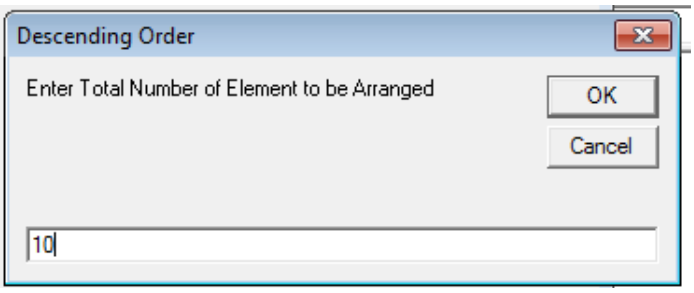
A program to read in 3 numbers and print them in descending order.

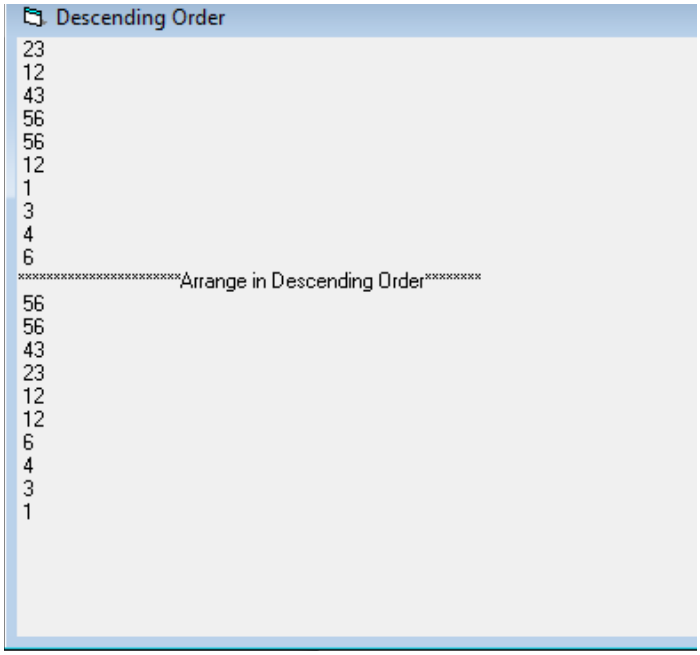
Source Code

```
Private Sub Form_Activate()
' A program to Arrange Number of Array  in Descending order
   Dim Total, K, I, J, Swap As Integer

   Total = Val(InputBox("Enter Total Number of Element to be Arranged ", "Descending Order"))
   Dim A(1000) As Integer
   'Enter List of Array to Rearrange
   For K = 1 To Total
      A(K) = Val(InputBox("Enter the List of Array to Rearrange " & K, "Array Elements"))
   Next
   For K = 1 To Total
      Print A(K)
   Next
      Print "**********************Arrange in Descending Order********"

      'Code that rearrange the list of Array in descending order
      For K = 1 To Total
         For J = 1 To K
            If (A(J) < A(K)) Then
               Swap = A(K)
               A(K) = A(J)
            A(J) = Swap
            End If
```

```
    Next
  Next

  'Print the Descending Order Arrangement
  For I = 1 To Total
    Print A(I)
  Next
End Sub
```

Object Screen





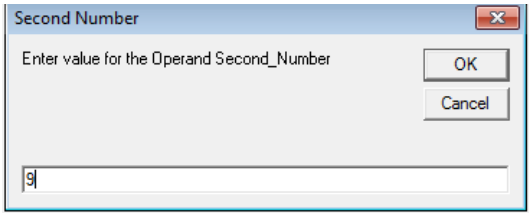
A Program to enable a computer to be used as a simple calculator

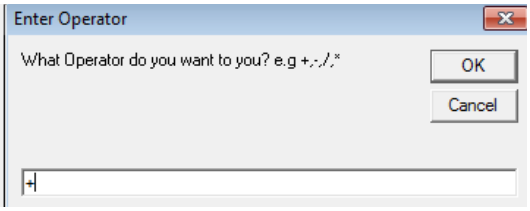
Source Code

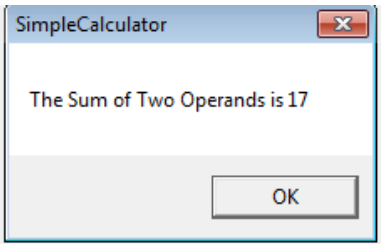
```
Private Sub Form_Activate()
Dim First_Num, Second_Num As Integer
Dim Answer As Single
Dim Operator, Response As String
  'Supply Values for the Operand
upcome:
  First_Num = Val(InputBox("Enter value for the Operand First_Number ", "First Number"))
  Second_Num = Val(InputBox("Enter value for the Operand Second_Number ", "Second Number"))

  Operator = InputBox("What Operator do you want to you? e.g +,-,/,*", "Enter Operator")

    If Operator = "+" Then
      Answer = First_Num + Second_Num
      MsgBox "The Sum of Two Operands is " & Answer
    ElseIf Operator = "-" Then
      Answer = First_Num - Second_Num
      MsgBox "The Sum of Two Operands is " & Answer
    ElseIf Operator = "/" Then
      Answer = First_Num / Second_Num
      MsgBox "The Sum of Two Operands is " & Answer
    ElseIf Operator = "*" Then
      Answer = First_Num * Second_Num
      MsgBox "The Sum of Two Operands is " & Answer
    Else
```

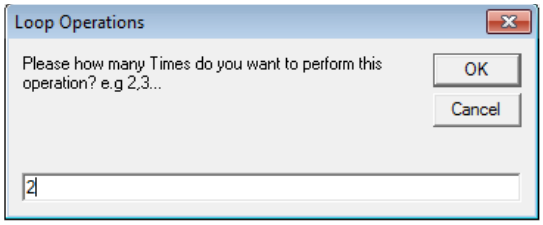
```
        MsgBox "Invalid Operator Entered, Try Again", vbCritical
    End If

    Times = InputBox("Please how many Times do you want to perform this operation? e.g 2,3...", "Loop
Operations")
        For I = 1 To Times
            Response = InputBox("Do you want to perform another Operation? Y or N", "Decision Making")
                        If Response = "Y" Then
                GoTo upcome
            Else
                MsgBox "Thank you for using this Simple Calculator", vbInformation, "Thank you God
Bless You"
            Exit Sub
            End If
    Next
End Sub
```

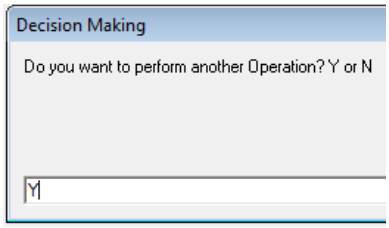
Object Screen











A

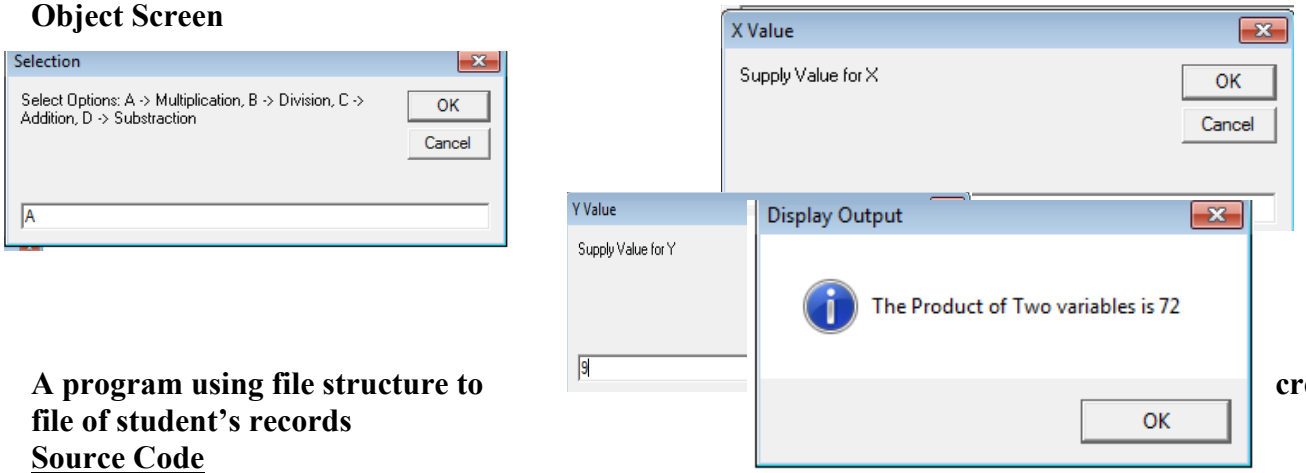
Program to perform positive integer multiplication and division.

Source Code

```
Private Sub Form_Activate()
' A Program to Display list of operation to perform on two operand
    Dim options As String
    Dim X, Y As Integer
    Dim Ans As Single
    Ans = 0

    options = InputBox("Select Options: A -> Multiplication, B -> Division, C -> Addition, D ->
Substraction", "Selection")
        If options = "A" Then
            MsgBox "You are just Selected to Perform Multiplication Operation", vbInformation, "Information"
            X = Val(InputBox("Supply Value for X ", "X Value"))
            Y = Val(InputBox("Supply Value for Y ", "Y Value"))
            Ans = X * Y
             MsgBox "The Product of Two variables is " & Ans, vbInformation, "Display Output"
        ElseIf options = "B" Then

            MsgBox "You are just Selected to Perform Division Operation", vbInformation, "Information"
            X = Val(InputBox("Supply Value for X ", "X Value"))
            Y = Val(InputBox("Supply Value for Y ", "Y Value"))
                If X <> 0 And X <= Y Then
                    X = X - Y
                    Ans = Ans + 1
                Else
```

```
            Ans = X / Y
        End If
    MsgBox "The Division of Two variables is " & Ans, vbInformation, "Display Output"
ElseIf options = "C" Then
    MsgBox "You are just Selected to Perform Addition Operation", vbInformation, "Information"
    X = Val(InputBox("Supply Value for X ", "X Value"))
    Y = Val(InputBox("Supply Value for Y ", "Y Value"))
    Ans = X + Y
    MsgBox "The Sum of Two variables is " & Ans, vbInformation, "Display Output"
ElseIf options = "C" Then
    MsgBox "You are just Selected to Perform Substraction Operation", vbInformation, "Information"
    X = Val(InputBox("Supply Value for X ", "X Value"))
    Y = Val(InputBox("Supply Value for Y ", "Y Value"))
    Ans = X - Y
    MsgBox "The Differences of Two variables is " & Ans, vbInformation, "Display Output"
Else
    MsgBox "Invalid Entry, Please Try Again", vbCritical, "Error"
End If
End Sub
```

Object Screen



A program using file structure to create the file of student's records

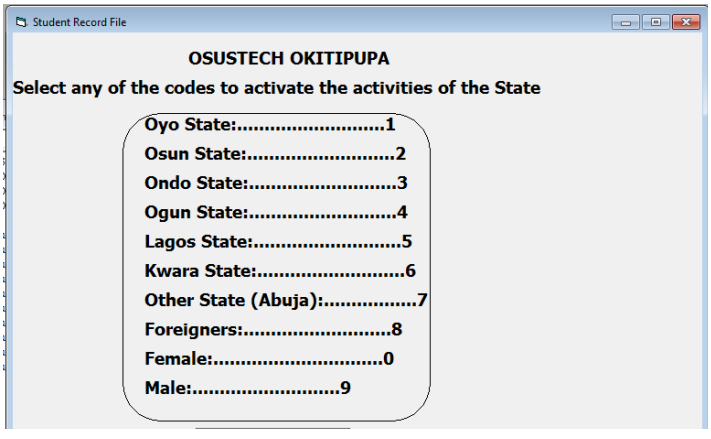
Source Code

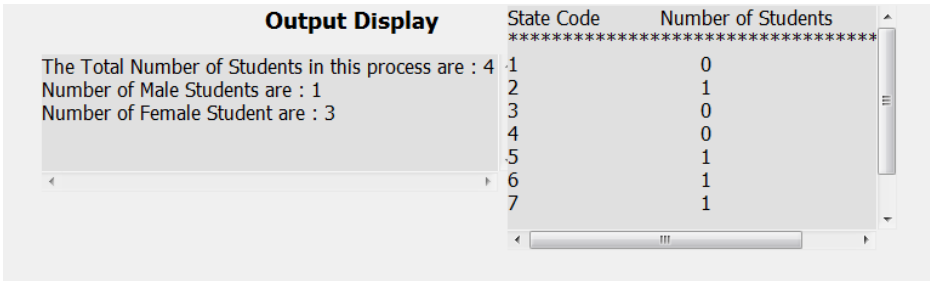
```
Dim Lv As ListItem
Private Sub CmdCode_Click()
Dim N, No_Std, C, D, Q, W As Integer
Dim A, B, E, F, G, H As Integer
Dim Student_Name As String
Dim Sexcode, Matric_No, Statecode As Integer
   A = 0
   B = 0
   C = 0
   D = 0
   E = 0
   F = 0
   G = 0
   H = 0
   Q = 0
   W = 0

   No_Std = Val(InputBox("Please Enter the Number of Student in Osustech", "Number of Student"))
     For N = 1 To No_Std
        Student_Name = InputBox("Please Enter Student Name", "Student Name of 30 Character")
        Matric_No = Val(InputBox("Please Enter the Student Matric Number", "Matric Number"))
           If Matric_No >= 0 Then
           Statecode = Val(InputBox("Enter State code from range 1 to 8", "State Code"))
```

```
            If Statecode = 1 Then
               A = A + 1
            ElseIf Statecode = 2 Then
               B = B + 1
            ElseIf Statecode = 3 Then
               C = C + 1
            ElseIf Statecode = 4 Then
               D = D + 1
            ElseIf Statecode = 5 Then
               E = E + 1
            ElseIf Statecode = 6 Then
               F = F + 1
            ElseIf Statecode = 7 Then
               G = G + 1
            ElseIf Statecode = 8 Then
               H = H + 1
            Else
                MsgBox "No Such Code in the File", vbCritical, "Error Input"
            End If
               Sexcode = (InputBox("Enter Sex code 0 for Female, 9 for Male", "Sex Code"))
               If Sexcode = 0 Then
                  Q = Q + 1
               ElseIf Sexcode = 9 Then
                  W = W + 1
               Else
                  MsgBox "No Such Code in the File", vbCritical, "Error Input"
               End If
            End If
        Next

      frmStd_Rec.Height = 9840
      txtDisplay.Text = txtDisplay.Text & "The Total Number of Students in this process are : " & No_Std &
vbCrLf
      txtDisplay.Text = txtDisplay.Text & "Number of Male Students are : " & Q & vbCrLf
      txtDisplay.Text = txtDisplay.Text & "Number of Female Student are : " & W & vbCrLf

     txtOutput.Text = txtOutput.Text & "State Code" & Space(10) & "Number of Students" & vbCrLf
     txtOutput.Text = txtOutput.Text & "**************************************" & vbCrLf
     txtOutput.Text = txtOutput.Text & "1" & Space(30) & A & vbCrLf
     txtOutput.Text = txtOutput.Text & "2" & Space(30) & B & vbCrLf
     txtOutput.Text = txtOutput.Text & "3" & Space(30) & C & vbCrLf
     txtOutput.Text = txtOutput.Text & "4" & Space(30) & D & vbCrLf
     txtOutput.Text = txtOutput.Text & "5" & Space(30) & E & vbCrLf
     txtOutput.Text = txtOutput.Text & "6" & Space(30) & F & vbCrLf
     txtOutput.Text = txtOutput.Text & "7" & Space(30) & G & vbCrLf
     txtOutput.Text = txtOutput.Text & "8" & Space(30) & H & vbCrLf
```

End Sub

Object **Screen**



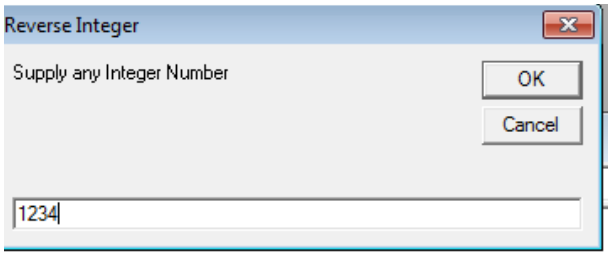


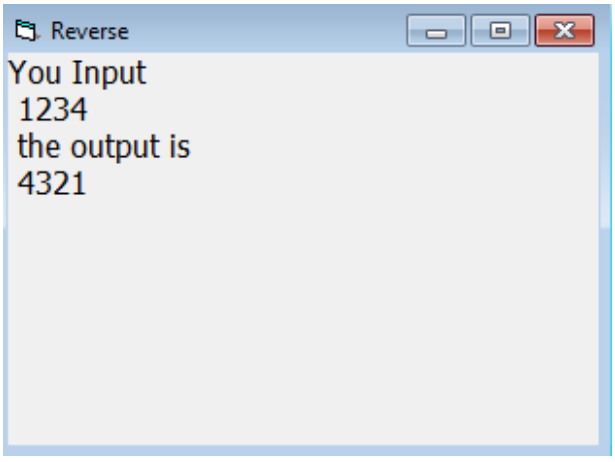
A program that accepts integer value from right to left and prints it from left to right

Source Code

```
Private Sub Form_Activate()
Dim R, G, N, V As Integer
   N = Val(InputBox("Supply any Five Integer Number", "Reverse Integer"))
   G = N
   Print "You Input "
   Print G
   V = 0

   Do While (G > 0)
      R = Int(G Mod 10)
      V = Int(V * 10) + R
      G = Int(G / 10)
   Loop
   Print " the output is "
   Print V
End Sub
```

Object Screen





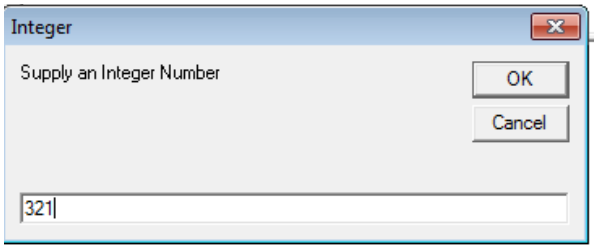
A program that accepts positive integer as input and finds the sum of the digits that make up the integer.

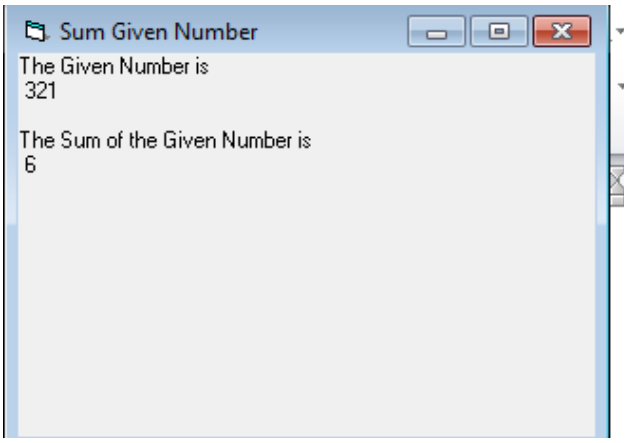
Source Code

```
Private Sub Form_Activate()
'Add up any Given Integer Number

   Dim R, N, Sum, GV As Integer
   Sum = 0

   N = Val(InputBox("Supply an Integer Number", "Integer"))
      GV = N
      Print "The Given Number is "
      Print GV
      Do While (GV <> 0)
         R = GV Mod 10
         Sum = Sum + R
         GV = GV / 10
```

```
    Loop
    Print
        Print "The Sum of the Given Number is "
  Print Sum
End Sub
```

Object Screen

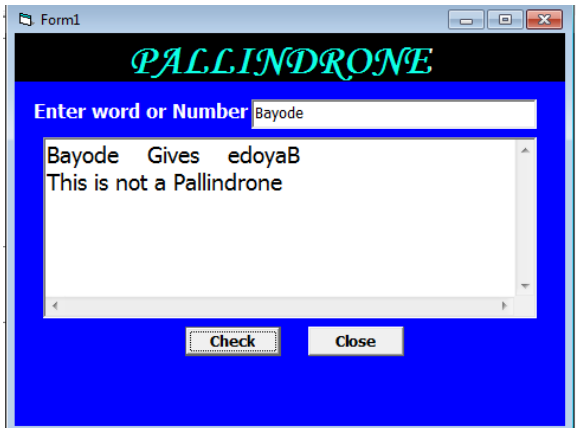




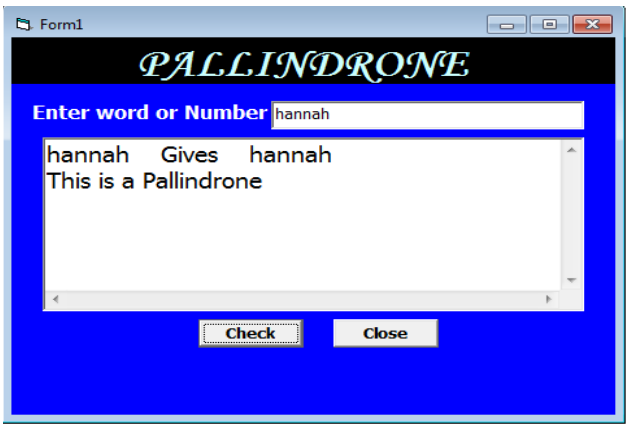
A Program to test for palindrome

Source Code

```
Dim A, B As String
Private Sub Command1_Click()
A = (Text1.Text)
B = StrReverse(A)
If (B = A) Then
Text2.Text = A & "    " & "Gives" & "    " & B & vbNewLine & "This is a Pallindrone"
Else
Text2.Text = A & "    " & "Gives" & "    " & B & vbNewLine & "This is not a Pallindrone"
End If

End Sub

Private Sub Command2_Click()
End
End Sub
```



A



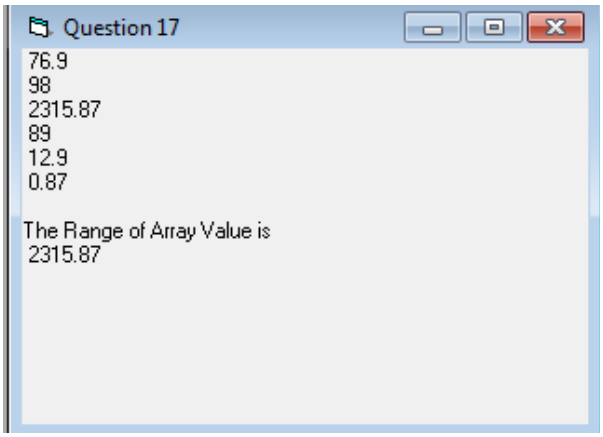
Program to find the range of a set of numbers, where range is defined as the difference between the biggest and the smallest numbers in a set.

Source Code

```
Private Sub Form_Activate()
Dim Largest, Smallest, Range As Single
Dim Count, Num As Integer
```

```
    Num = (InputBox("Enter the Number of Array to Input", "Largest and Smallest"))
Dim X(1000) As Single
    'Input Value of an Array
    For Count = 1 To Num
       X(Count) = (InputBox("Supply Value into Array Element " & Count, "Array Element"))
    Next
    'Output Value of an Array
    For Count = 1 To Num
      Print X(Count)
    Next
    Largest = X(0)
    For Count = 1 To Num
       If Largest < X(Count) Then
          Largest = X(Count)
       End If
    Next
    Smallest = X(0)
    For Count = 1 To Num
       If Smallest > X(Count) Then
          Smallest = X(Count)
       End If
    Next
Range = Largest - Smallest
Print
Print "The Range of Array Value is "
Print Range
End Sub
```

Object Screen



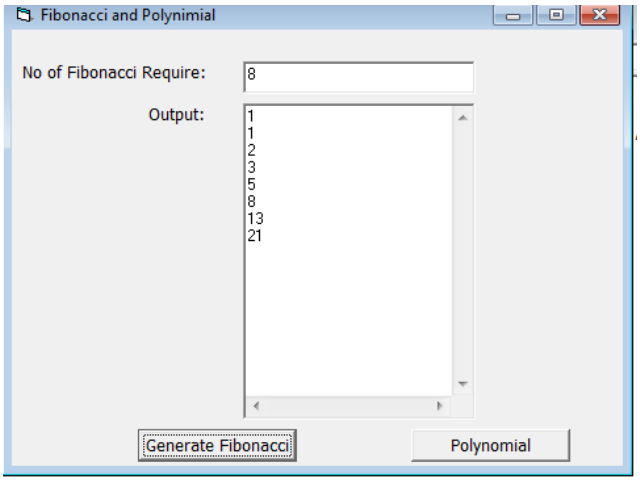
A program to determine the nth Fibonacci number.

Source Code

```
Private Sub Command1_Click()
    If Text1.Text = Empty Then
       MsgBox "Please Supply the Number of Fibonacci Required", vbCritical, "Fibonacci"
    Exit Sub
    End If
txt1.Text = Empty
    A = Text1.Text
    Sum = 0
    j = 0
    k = 1
          txt1.Text = txt1.Text & k & vbCrLf
```

```
For i = 2 To A
   Sum = j + k
      txt1.Text = txt1.Text & Sum & vbNewLine
      j = k
      k = Sum
Next
End Sub
```

Object Screen

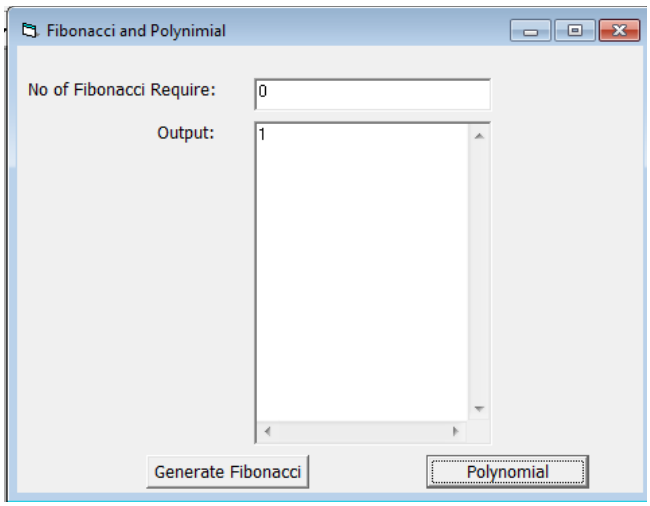


Legendre Polynomial with formula P_n=1, P1=x

Source Code

```
Private Sub Command2_Click()
Dim N, P, X As Single
If Text1.Text = Empty Then
      MsgBox "Please Supply the Number of Fibonacci Required", vbCritical, "Fibonacci"
   Exit Sub
End If
  X = (Text1.Text)
   N = (InputBox("Supply Value of N", "Polynomial"))
      If N = 0 Then
         P = 1
      ElseIf N = 1 Then
         P = X
      Else
         P = ((2 * N - 1) * ((N - 1) / N)) - ((N - 1) * (N - 2) / N)
      End If
   txt1.Text = P
End Sub
```

Output Screen

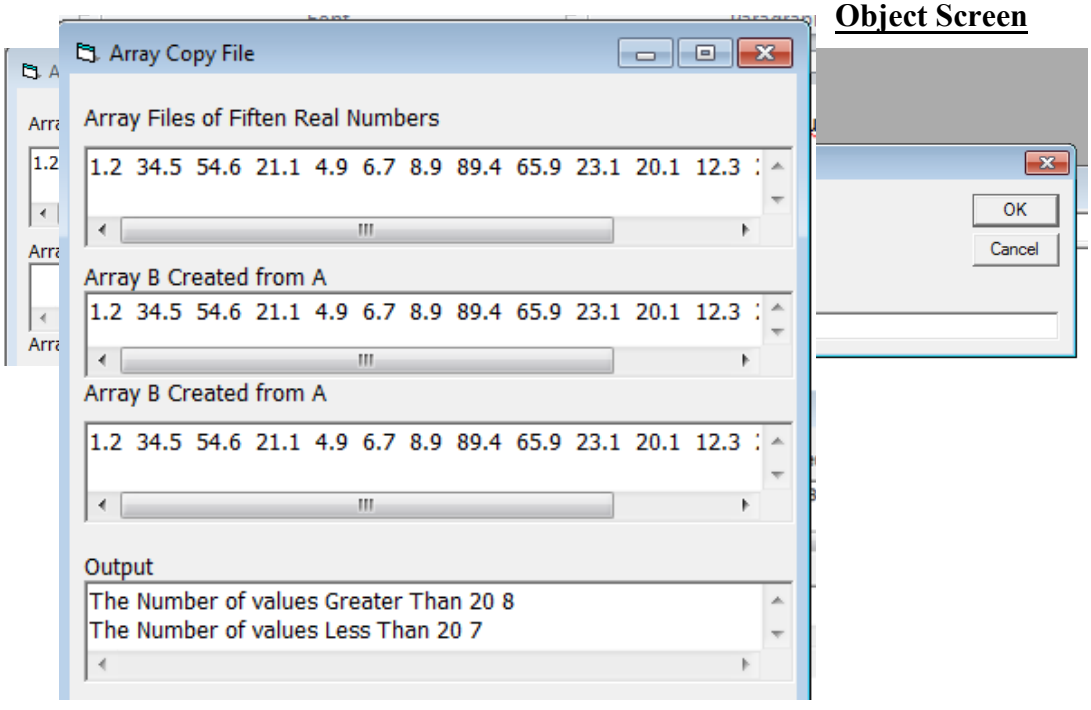


A program that creates a file of 15 real numbers

Source Code

```
Private Sub Form_Activate()
'Read a file of fiften Array Real Numbers
Dim A(15) As Single
Dim B(15) As Single
Dim N, Count, Kounter As Integer

   Count = 0
   Kounter = 0
   (A)
   For N = 1 To 15
      A(N) = (InputBox("Supply Value into the Array " & N, "Array File"))
      txtfile.Text = txtfile.Text & A(N) & Space(2)
   Next
(B)
'New Array to be created from The Above Array File
   For N = 1 To 15
      txtA.Text = txtA.Text & A(N) & Space(2)
   Next
'The File of B(15) to be created from A(15)
   For N = 1 To 15
      B(N) = A(N)
      txtCreated.Text = txtCreated.Text & B(N) & Space(2)
   Next
'Count The Number of Array Elents Greater Than 20 and Less Than 20
(C)
    For N = 1 To 15
      If (A(N) > 20#) Then
      Counter = Counter + 1
      Else
      Kounter = Kounter + 1
      End If
   Next
   txtOutput.Text = txtOutput.Text & "The Number of values Greater Than 20 " & Counter & vbCrLf
   txtOutput.Text = txtOutput.Text & "The Number of values Less Than 20 " & Kounter

End Sub
```

Object Screen

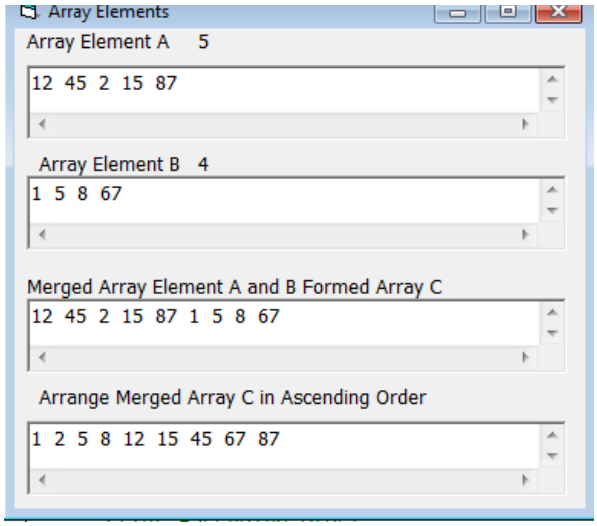


A program that merges two different linear arrays of values into a new stream
Source Program

```
Private Sub Form_Activate()
Dim P, R, D As Integer
Dim Swap, J As Integer
   P = Val(InputBox("Enter Size of Array A", "Array A"))
   R = Val(InputBox("Enter Size of Array B", "Array B"))
Dim A(1000), B(1000), C(1000 + 1000) As Integer
   Label5.Caption = P
   Label6.Caption = R
   For D = 1 To P
      A(D) = Val(InputBox("Enter Element into Array A", "Array A"))
      txtA.Text = txtA.Text & A(D) & Space(2)
   Next
   For D = 1 To R
      B(D) = Val(InputBox("Enter Element into Array B", "Array A"))
      txtB.Text = txtB.Text & B(D) & Space(2)
   Next
   For D = 1 To P
      C(D) = A(D)
   Next
   For D = 1 To R
      C(P + D) = B(D)

   Next
   For D = 1 To P + R
       txtC.Text = txtC.Text & C(D) & Space(2)
   Next

   'Code that rearrange the list of Array in descending order
      For D = 1 To (P + R) - 1
         For J = 1 To (P + R) - 1
            If (C(J) > C(J + 1)) Then
               Swap = C(J)
               C(J) = C(J + 1)
            C(J + 1) = Swap
            End If
         Next
      Next
   'Print Ascending Order
   For D = 1 To P + R
         txtAsc.Text = txtAsc.Text & C(D) & Space(2)
      Next
```

End Sub

Object Program

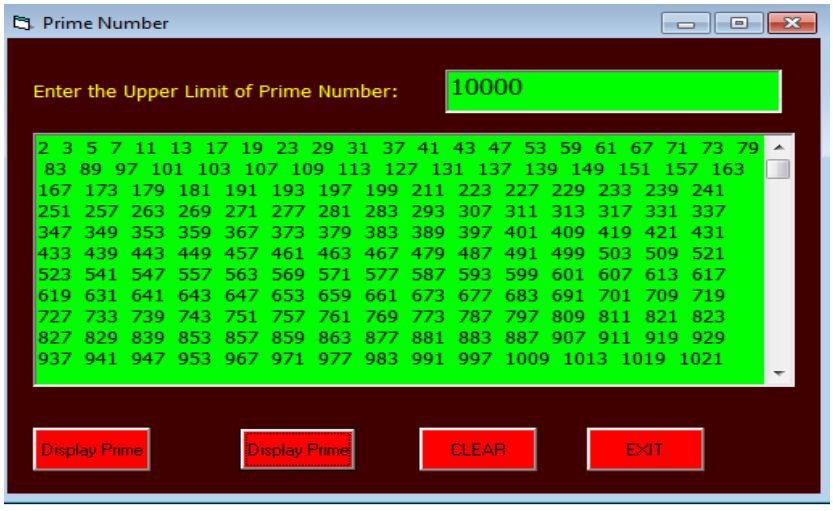


A Prime that print the table of prime Number

Source Code

```
Private Sub CmdPrime_Click()
Dim I, J, Num, Prime As Integer
Prime = 1
Num = Val(txtUpper.Text)
   For I = 2 To Num

      For J = 2 To I - 1
         If I Mod J = 0 Then
            Prime = 0
      Exit For
         Else
            Prime = 1
         End If
      Next   J
         If Prime = 1 Then
         txtoutput.Text = txtoutput.Text & I & Space(2)
         End If
   Next      I
End Sub
```

Object Program

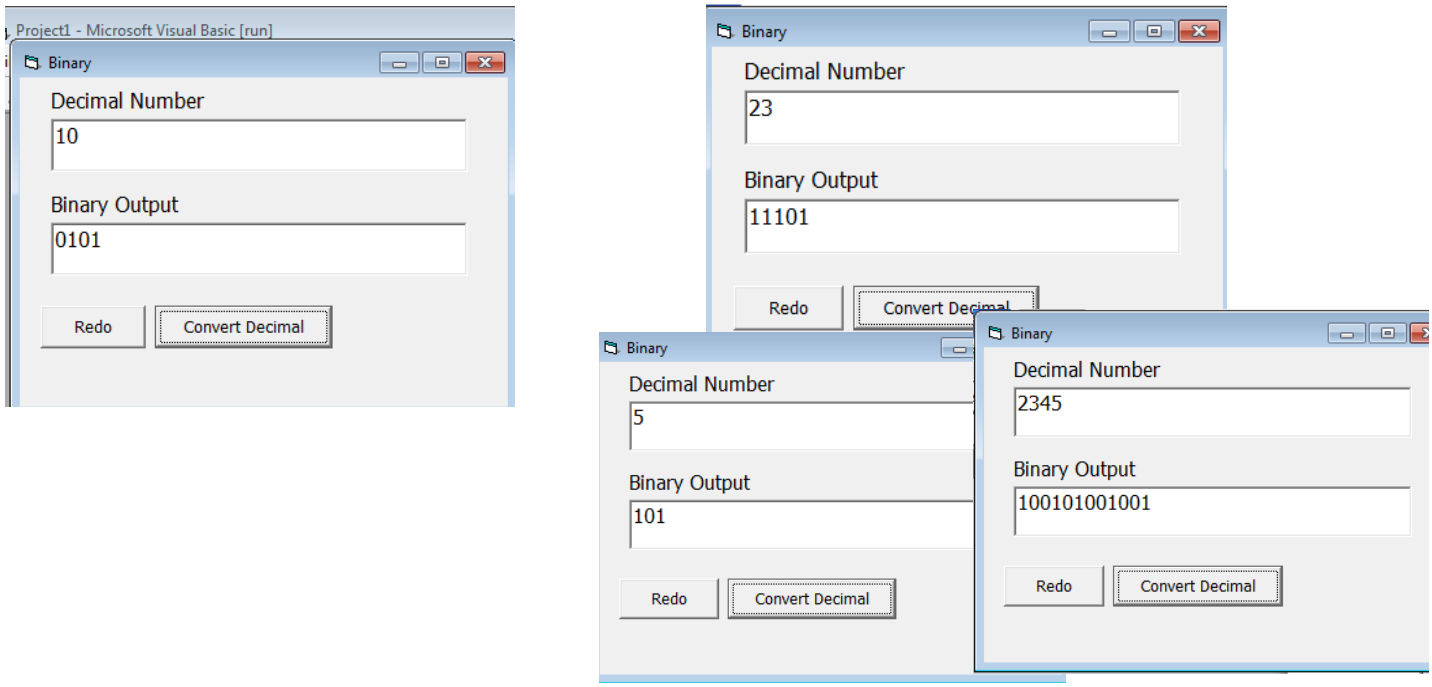


A program that converts decimal number to binary

Source Code

```
Private Sub CmdConvert_Click()
Dim Num, A, B As Integer
   Num = txtNum.Text
      Do While (Num <> 0)
         A = Num Mod 2
         Num = (Num - A) / 2
         txtOutput.Text = txtOutput.Text & A
      Loop
End Sub
```

Object Program

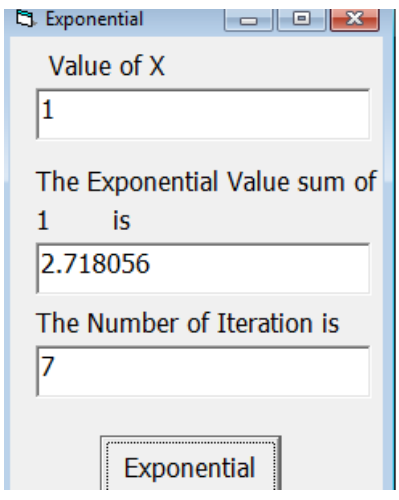


A program to compute the exponential of e^x of a number

Source Code

```
Private Sub cmdExp_Click()
Dim sum, term, x As Single
Dim i As Integer
x = txtX.Text
sum = 1
term = x
i = 1
Do While (term >= 0.001)
sum = sum + term
i = i + 1
term = (term * x) / i
Loop
lblx.Caption = x
txtsum.Text = sum

txti.Text = i
End Sub
```

Object Program

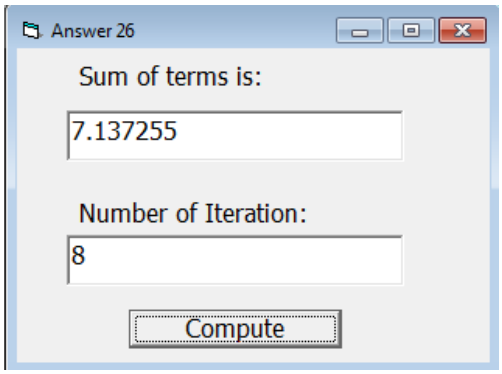


A program to compute the computations of iterations between successive approximations becomes less than 0.001.

Source Program

```
Private Sub CmdDiff_Click()
Dim Sum, Term1, Term2, Num As Single
Num = 4#
Dim count As Integer
count = 1
   Term1 = (2# * 2#) / (1 * 3)
   Term2 = (Num * Num) / ((Num - 1) * (Num + 1))
   Do While ((Term1 - Term2) >= 0.001)
      Sum = Sum + Term2
      Term1 = Term2
      count = count + 1
      Num = Num + 2#
      Term2 = (Num * Num) / ((Num - 1) * (Num + 1))
   Loop
   txtSum.Text = Sum
   txtIte.Text = count
End Sub
```

Object Program

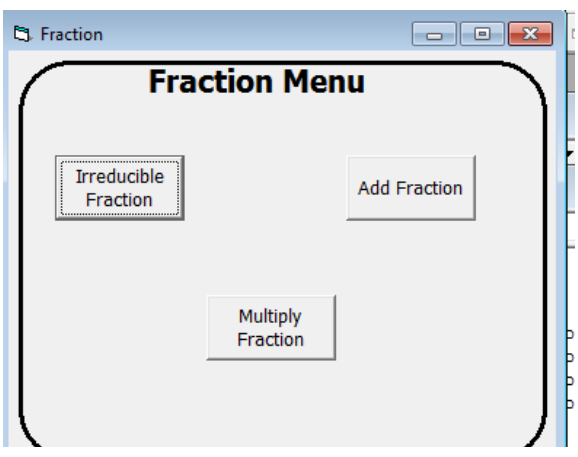


A program to reduce fraction to irreducible form, adding 2 fractions and multiplying 2 fractions.

Source Code

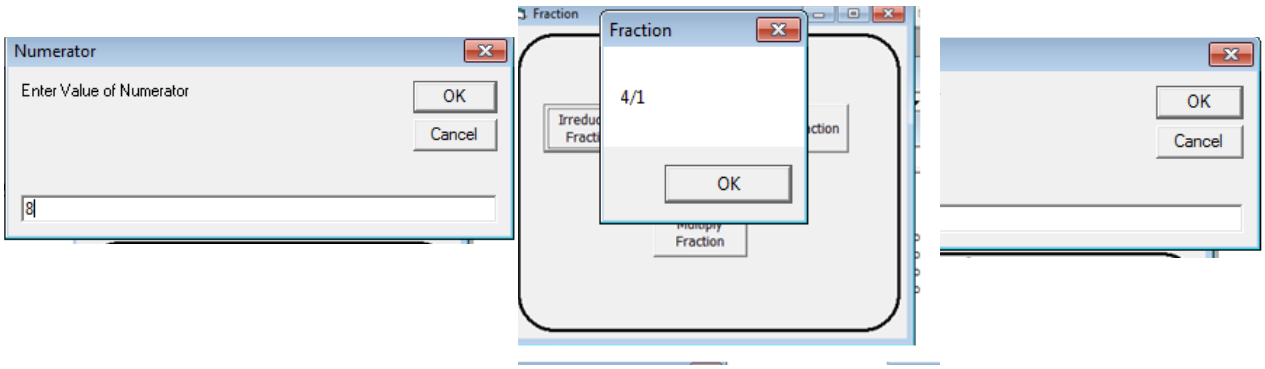
```
Public Sub Irreduciblefra(N As Integer, D As Integer)
 Dim Divisor As Integer
   Divisor = 2
      N = Val(InputBox("Enter Value of Numerator ", "Numerator"))
      D = Val(InputBox("Enter Value of Denumerator ", "Denumerator"))
   While (((N Mod 2) = 0) And ((D Mod 2) = 0))
      N = N / 2
```

```
        D = D / 2
       out = N & "/" & D
 MsgBox out
 Wend
End Sub
Private Sub Command1_Click()
  Dim A As Integer
   Dim B As Integer
    'a = txtA.Text
    'b = txtB.Text
   Irreduciblefra A, B
End Sub

Private Sub Command2_Click()
Dim N1, D1, N2, D2 As Integer
 Dim Divisor As Integer
 Dim A, B As Integer
       N1 = Val(InputBox("Enter Value of N1 ", "N1"))
       D1 = Val(InputBox("Enter Value of D1 ", "D1"))
       N2 = Val(InputBox("Enter Value of N2 ", "N2"))
       D2 = Val(InputBox("Enter Value of D2 ", "D2"))
   Divisor = 2

       B = D1 * D2
       A = (((N1 * B) / D1) + ((N2 * B) / D2))
         Do While ((A Mod 2) = 0) And ((B Mod 2) = 0)
              A = A / 2
              B = B / 2
            'MsgBox A & "/" & B
          'Wend
           Loop
      MsgBox A & "/" & B
End Sub

Private Sub Command3_Click()
Dim N1, D1, N2, D2 As Integer
 Dim Divisor As Integer
 Dim A, B As Integer
       N1 = Val(InputBox("Enter Value of N1 ", "N1"))
       D1 = Val(InputBox("Enter Value of D1 ", "D1"))
       N2 = Val(InputBox("Enter Value of N2 ", "N2"))
       D2 = Val(InputBox("Enter Value of D2 ", "D2"))
   Divisor = 2

       A = N1 * N2
       B = D1 * D2
        Do
```




```
While ((A Mod 2) = 0) And ((B Mod 2) = 0)
A = A / 2
B = B / 2
'MsgBox A & "/" & B
'Wend
Loop
MsgBox A & "/" & B
End Sub
```

Object **Screen**


Numerator
Enter Value of Numerator
OK
Cancel
8
Fraction
Fraction
4/1
OK
Fraction
OK
Cancel