



After studying this lesson, you will have better understanding of,

- The Evolution of computer languages
- Using computers to solve problems
- Representation of algorithms
- The use of Visual Basic to develop applications

Day by day the computer is becoming an indispensable tool in our lives. Computers can be used in a wide variety of applications. In this chapter, the main emphasis is on how the computer can be used as a tool to solve problems.

When using a computer to solve a problem, the method to be followed should be given to the computer as a set of commands using a language understood by the machine. The languages that can be directly understood by the computer are called Machine Languages. Generally, the set of commands of a machine language and their structure depend on the processor type of the corresponding computer. Consequently, the machine languages understood by different types of processors are different from each other.

For example, the machine languages designed for the 'Intel' processor family are quite different from the machine languages designed for the 'SPARC' processors. However, there are different types of processors in the market designed to run almost similar machine languages. The processor family 'Pentium' and 'AMD' are examples for such processor types.

Though different processor types recognize different machine languages, one property is common to all machine languages irrespective to the processor type for which the language is defined. That is, all machine language commands are composed

of sequences of binary symbols 0 and 1 only. The main reason for this limitation is that the electronic devices such as computers can interpret commands in a binary code only. Such languages built only from a binary alphabet are called Low Level Languages.

Machine languages are very difficult for humans to understand and program computers. However, these languages are directly understood by computers and the programs written in machine code run fast on computers. Because of these reasons machine languages are considered to be closer to the hardware rather than to humans.

Since machine languages are very difficult to understand and use, a large number of programming languages closer to human languages have been built over the years. These languages are grouped as High Level Languages. The programming languages 'C', 'VB' and 'PHP' are a few examples for such high level languages.

A sequence of statements in a programming language prepared to accomplish some task by using a computer is termed a Computer Program. Statements in a program are either commands for the computer or comments for humans. Comments are typically embodied in a program to explain the code of the program to humans in simple terms. This will be explained later.

Although humans can easily understand programs written in high level programming languages such as C, VB and PHP, the machine cannot. Therefore, to run such programs on a computer, these programs must be transformed into a low level language understood by the computer. The process of transforming a program written in a high level language to a low level language is termed Language Translation or Compilation of the program. Many different ways of language translation is explained later in this chapter.



1.1 Evolution of Programming Languages



All programming languages developed from the inception can be classified into four classes based on the nature of their commands.

1. First Generation Programming Languages
(Machine Languages)

2. Second Generation Programming Languages
(Assembly Languages)
3. Third Generation Programming Languages
(High Level Languages)
4. Fourth Generation Programming Languages
(Artificial Languages)

One of the key features apparent in the evolution of programming languages is that the programming languages have become easily understandable and usable over the years.

First Generation Programming Languages

All machine languages fall into this category.

Common Properties of these languages:

- ❖ All commands are based on the binary code symbols 0 and 1.
- ❖ Coding programs and programming computers are extremely difficult.
- ❖ Depend on the machine type. Therefore, a program coded for one processor type may not run on a machine with a different processor type.
- ❖ Programmer should have a comprehensive knowledge about the hardware of the intended computer system.

Second Generation Programming Languages

These languages were developed primarily to alleviate the problem of using only the binary symbols 0 and 1 to develop programs.

Common Properties of these languages:

- ❖ Commands are represented by symbolic names such as 'ADD', 'SUB'.
- ❖ Ability to use symbolic names to store data at memory locations and to retrieve data from memory by using these symbolic names.

- ❖ Ability to debug programs easier than programs in machine code.
- ❖ Depend on the machine type.
- ❖ Need for Programmer to have comprehensive knowledge of hardware of the intended computer system.

Programs written in Assembly Languages cannot be executed directly on the computer as Assembly Languages are not understood by the processors. Instead, such programs have to be transformed into the machine language of the intended computer using appropriate language translation software. Such language translation software that translate programs written in Assembly Languages to Machine Language are called 'Assemblers'.

Third Generation Programming Languages

Programming languages such as C and VB belong to this type.

Common Properties of these languages:

- ❖ Based on languages close to humans such as English .
- ❖ Do not depend on the machine.
- ❖ Programs can be developed and debugged easily.

Like the Assembly languages, these languages also cannot be directly understood by the machines. Therefore to execute programs written in these languages they have to be translated into machine code using appropriate language translation software.

Fourth Generation Programming Languages

Fourth Generation Programming Languages provide Visual Environment making it easy for the user to write computer programs.

Common Properties of these languages:

- ❖ Close to human languages such as English.
- ❖ Ability to learn the languages within a short period.
- ❖ Ability to accomplish a task by using a smaller number of commands relative to Third Generation Programming Languages.



Activity 1.1

- (1) List the main differences between machine languages and assembly languages.
- (2) List out main differences between assembly languages and high-level languages.



Programming Language Translation

We have seen in the previous sections that a computer can execute programs written only in a machine code. Therefore a programme written in any other language has to be translated into machine code by using appropriate language translation software to execute it on the computer. The process of converting a program from one language to another is termed Language Translation. There are three techniques for language translation:

- (1) Assemblers
- (2) Compilers
- (3) Interpreters

1. Assemblers

Assemblers translate assembly language into machine language which can be recognized by the computer.

2. Compilers

Compilers translate all the commands in a program to an object code which can be understood by the computer. Once a program is compiled, it can be executed by the machine. C and FORTRAN are two examples of compiler based programming languages.

When compiling programs, the original program used for the compilation is called the Source Code Program and the resulting program produced by the compiler is called the Object Code Program. The main process what the compiler does is to

translate Source Code Program into Object Code Program. During the compilation process, the compiler also checks whether the source program is free of errors with respect to the grammar of the programming language. If the compiler detects errors, error messages are produced and the compilation process terminates without producing the object code. Then the user must compile the program again after correcting the detected errors. The user may have to follow this procedure repeatedly until the program is free of errors. Errors found in a program are called 'Bugs' and the process of eliminating bugs from a program is called Debugging.

3. Interpreters

Unlike compilers, interpreters do not translate the entire program into machine code at once. Instead, an interpreter does the translation at the moment the program is run, one line at a time. As a result, the lines of the program have to be translated into machine code each time the program is run. Therefore, the time taken to run a program of this type is typically greater than the time taken to run a similar program that has been compiled. 'VB' and 'Python' are examples of such languages.



1.2 Stages to be followed in solving problems using the Computer

The main steps to be followed in solving a problem using a computer are listed below.

1. Analyze problem .
2. Develop an algorithm to solve the problem.
3. Implement the algorithm by writing a program in some programming language.
4. Identify and remove the bugs in the program.
5. Execute the program on the input data.

1) Analyzing the Problem

The main focus at this stage is to identify the

- (1) input
- (2) output
- (3) process necessary to convert the input into output.

Example : Let us assume that a program has to be developed to compute the area of a rectangle. The input, output and the process needed to perform this task are listed below.

Input : Length and Width of the rectangle.
Process : Perform the computation Length Width
Output : The result of the above process

2) Developing an algorithm to solve the problem

An unambiguous sequence of steps assembled together to solve a particular task is termed an algorithm.

An algorithm developed for the previous example is given below.

- Get length of rectangle.
- Get width of rectangle.
- Area = length width.
- State Area.

3) Translating an algorithm to programming language

Main aim at this stage is to write a program (or programs) in a programming language to carry out the steps in the algorithm developed in the previous stage. Today, a large number of programming languages are available to implement algorithms. C, Python, Php, VB and FORTRAN are a few examples of such languages. The process of writing a program in some programming language to implement an algorithm is termed 'Coding' and the person carrying out this process is designated a 'Coder' or a 'programmer'.

4) Identifying and removing the bugs in the program

During this stage the program is run on a collection of pre-prepared test data sets to identify the bugs in the program. If the bugs are detected the program has to be modified to eliminate these bugs. The process of identifying and eliminating the bugs in a program is called Debugging.

5) Executing the program on the input data

At this stage the final debugged program is executed on the real input data to obtain the required outputs.



Activity 1.2

Identify input and output of a program to find the area of a circular.



Activity 1.3

Identify input and output when building up a program to find the total and the average marks obtained by you for all the subjects in your last term test.



1.3 Representation of Algorithms

There are two different techniques for representing algorithms:


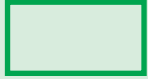

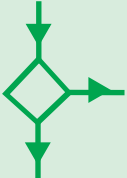

- Graphical Representation
- Textual Representation

One of the popular methods used to represent algorithms graphically is 'Flowcharts' whereas Pseudo codes have been used widely to represent algorithms textually.

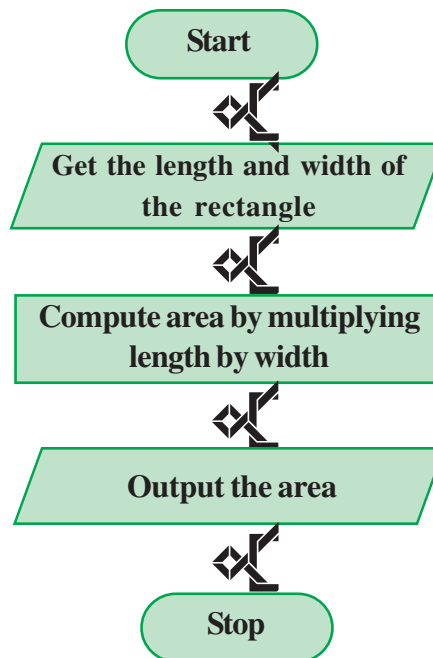


Flowcharts

A flowchart is a graphical representation of an algorithm. Each step in the algorithm is represented by a designated symbol and is linked with arrows showing the direction of data flow. A standard set of symbols has been introduced to be used in flowcharts. A few of the commonly used symbols defined by the standard are tabled below.

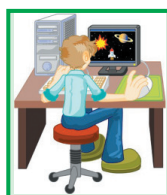
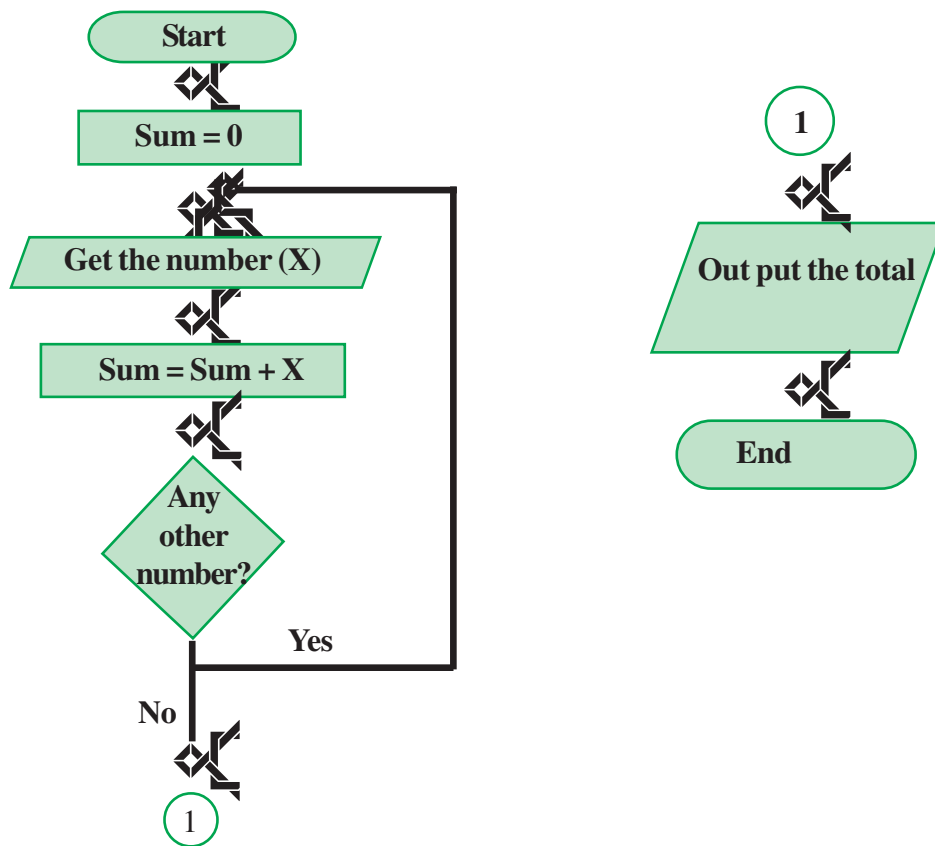
Symbol	Name	Usage
	Terminator	Indicates the Start and Stop of the process
	Process	Represents a command or a sequence of commands
	Data Input / Output	Represents data input and output
	Decision	Represents Decisions .
	Connector	Shows a jump from one point to another
	Flow Lines	Shows the direction of data flow

Example : A flow chart drawn to compute the area of a rectangle is shown below.



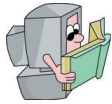
When a flow chart extends across many pages or flow lines have to be drawn across a page the connector symbol can be used to reduce the clutter of the flow chart.

Example : Draw a flowchart to show the steps to be followed in computing the total of a given set of numbers.



Activity 1.4

Expand the flow chart described in the previous example to compute the total and the average of a collection of numbers.



Pseudo code

Typically, a pseudo code does not depend on any programming language. There is no standard available to regulate the construction of pseudo codes. Therefore, when developing a pseudo code, careful attention must be paid in selecting terms and formulating statements so that the resulting pseudo code can be easily understood by a non-programmer.

Example : A pseudo code written to compute the area of a rectangle is given below.

- Start
- Get the length and width of the rectangle
- Area = length width
- Output Area
- End



1.4 Developing applications by using Visual Basic

In this section how Visual Basic (VB) can be used to solve problems is explained.



Visual Basic (VB)

Visual Basic (or VB in short) is a high-level programming language that belongs to the Third Generation of Programming Languages. VB is designed to run on the Microsoft Windows operating systems family. Therefore, programs written in VB cannot be executed on machines with other operating systems such as 'Linux'. This constraint can be seen as one of the main disadvantages of the VB language.

The Visual Basic programming language is tightly coupled with a program development tool. This type of program development tools are designed to provide an environment to assist the programmer in the application development processes that are called 'Integrated Development Environments' (IDE). As a result of the tight binding of VB with an IDE, the procedure to be followed in developing a VB application is different from the process of building applications in other languages. To develop an

application in VB one of the prerequisite is that the developer must have a good knowledge of the VB IDE.



Accessing the Visual Basic IDE

The Visual Basic IDE can be launched by executing the following sequence of commands.

- 1) Start
- 2) All Programs
- 3) Visual Studio 6.0
- 4) Microsoft Visual Basic 6.0

When the VB IDE is launched, an interface window of the Visual Basic Interface Development Environment as shown below would appear on the screen.

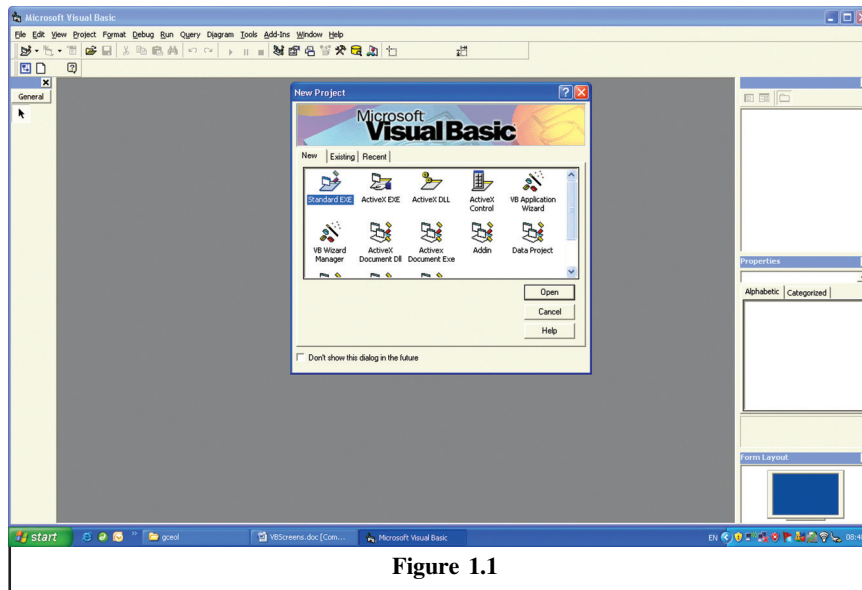


Figure 1.1

Different types of applications can be developed using this IDE. The icons that correspond to these different types are shown on this window. In this chapter the main objective is to explain how to construct applications of the type “Standard.EXE”. Therefore, by clicking over the icon labeled “Standard.Exe”, you would be able to open a window as shown below to develop an application of this type.

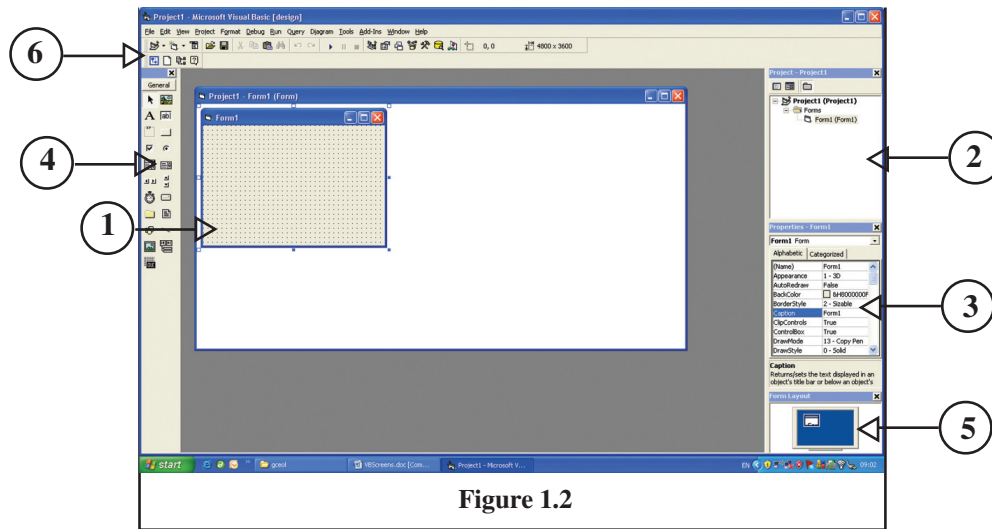


Figure 1.2

- | | |
|----------------------------|-----------------------|
| 1. Form Design Window | 4. Tool Box |
| 2. Project Explorer Window | 5. Form Layout Window |
| 3. Window Properties | 6. Main Menu |

N.B. : Depending on the version of the operating system and the applications installed, the sequence of commands to be executed to launch VB IDE on your computer may be different.

All main tools necessary to develop a VB application of type “Standard.EXE” are shown on the above IDE window.

Typically, a VB application can comprise many files of different types. All files that belong to a single application is commonly referred to as a VB Project.

The object marked 1 in the above figure is called the “Form Design Window”. This window is used to structure the screen layout or application interfaces of your application. Typically you should include all controls necessary to run your application on the application interface. The section of the above figure numbered 2 is called the “Project Explorer Window”. This window displays all projects you have opened and the different files in each project.



Activity 1.5

Observe whether all the windows that are shown in figure 1.2 are available in your computer. If any window is not shown, find out how to get that window to the screen.

The window numbered ③ in the above figure is called the “Properties Window”. This window displays the properties of the controls and the objects created in your application. This window can also be used to change the values of these properties.

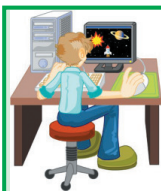


Activity 1.6

Click the mouse on the name of the current project in the Project Explorer Window and then go to the “Project Properties” window and change the name of the project to “First Project”.

The window labeled 4 in the above figure is called the “Tool Box”. A number of controls mostly used to construct VB applications can be found here.

The window labeled 5 shows how your application interface window will appear on your computer screen when it is executed.



Activity 1.7

Change the Caption property of ‘Form 1’ in the form design window to ‘First Form’. What is the width of this form?



Activity 1.8

Execute the commands Run → Start from the Main Menu labeled 6 in the above figure and observe the result. Now execute the commands Run → End and observe the result.

Executing the commands **File** → **Save Project** from the Main Menu, save the project you have created with any name you like.

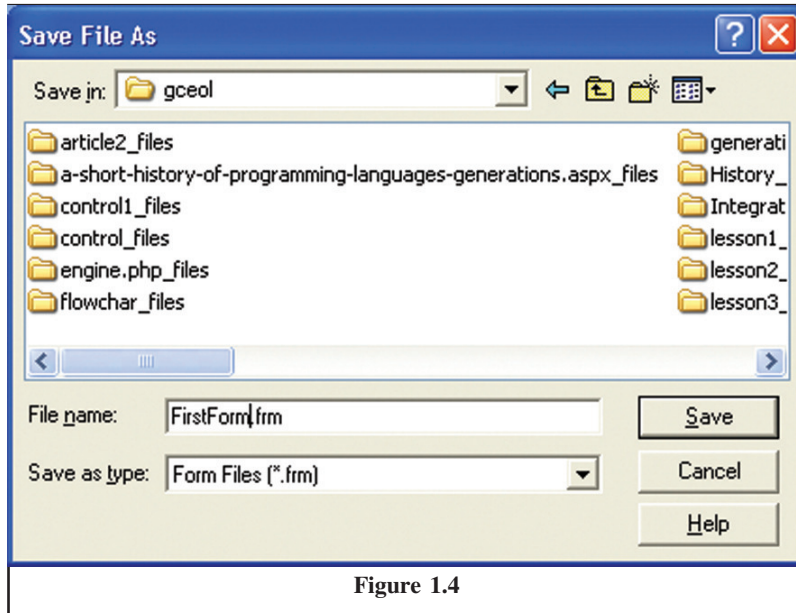


Figure 1.4

When saving a project, VB will request you to supply a name for each form of your project. The sequence of windows typically displayed by VB when saving a project is shown below.

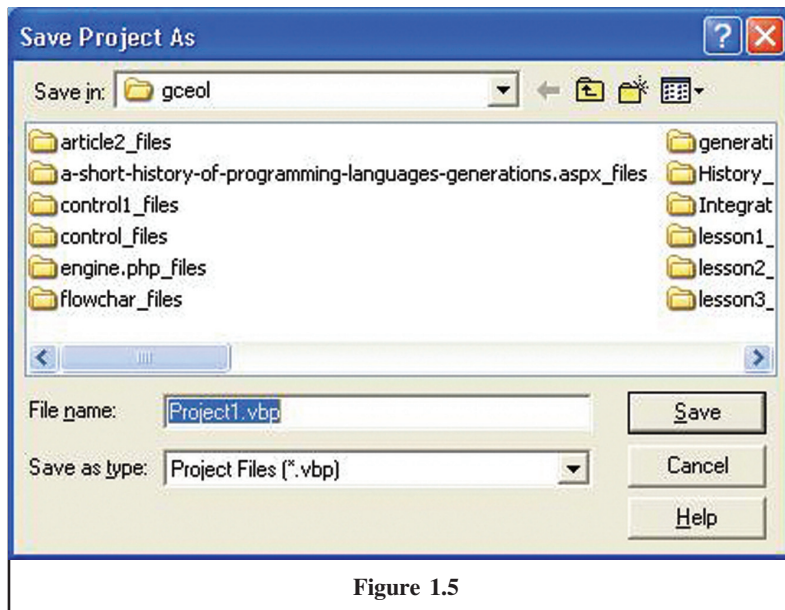


Figure 1.5

You can exit the Visual Basic IDE by executing the commands **File -> Exit** from the Main Menu.



Activity 1.9

Move to the directory where you have saved your first VB project and double click on the icon. Observe the result.



Activity 1.10

Explore how you can re-open the previously saved project by executing the commands **File -> Open Project** from the Main Menu of the VB IDE.



1.5 Sequence of Stages to be Followed in Developing a Visual Basic Application

Visual Basic language is designed primarily to develop Event Driven types of applications. The behaviour of this type of event driven application depends on the actions (such as 'Click', 'Change', 'GotFocus', 'KeyPress') performed by the user on the objects of the application interface.


The objects on a VB application interface window that can generate events are called 'Controls'. The Form that appears at the start of the VB IDE is also a control object.

There is a large collection of VB controls that can be used to develop an application. Out of this large collection, a few controls commonly used in developing many applications appear in the Toolbox window.

The Tool Tip that appears when you place the mouse pointer on top of an icon on the Toolbox gives you a hint on the function of that icon.



Activity 1.11

1. Take the mouse over different controls in the tool box window and identify the tools.
2. Change the height and width of the form in the VB IDE window by dragging the corners of the form. Observe how the Height and width property values of the form object change in the properties window as you drag the form corners.
3. Observe what will happen when you click the icon  at the top right hand corner of the tool box. Get the tool box back on the IDE by clicking View Toolbox from the main menu bar at the top of the IDE.



Adding Controls to a Form

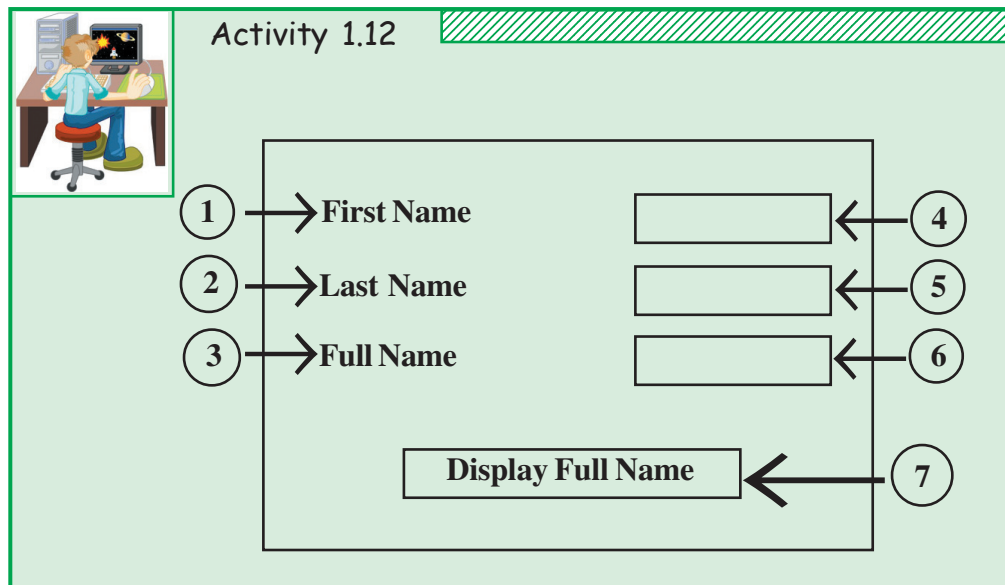
Controls can be inserted on a form in two different ways.

- 1) Double click on the icon of the control in the tool box: This action will cause an instance of the control to appear on the center of the active form. This control can subsequently be placed in the required position by using the mouse.
- 2) Click once on the required icon of the control in the Toolbox window. Move the mouse cursor over the form, click and hold the mouse at the position of the form where you want the upper left hand corner of the control to be. Then drag the cursor to the desired right hand corner of the control and release the mouse.

When you insert a control onto a form, the VB IDE assigns appropriate values to most of its properties automatically. You can change these property values subsequently through the Properties Window.

There are three main stages in developing applications in VB:

- 1) Place the required controls on to a form.
- 2) Change the properties of the control on the form as required.
- 3) Add Subprograms to controls to define their actions.



Develop an application as shown in the activity 1.12 by adding 7 controls to the form on the VB IDE. Some of the properties of these controls are listed below.

Save your project by the name 'MyProject.vbp' by invoking commands **File** → **Save Project** in the main menu.

Here, you have to save the forms you prepared. Let us use the name FirstForm for that.

a) Form Object Properties

Name : frmFirst
Caption : First Form
Width : 10000
Height : 6000

b) Object 1

Type : Label
Name : lblFName
Caption : First Name
Width : 1000
Left : 450
Top : 400
Height : 200

c) Object 2

Type : Label
Name : lblLName
Caption : Last Name
Width : 1000
Left : 450
Top : 950
Height : 200

d) Object 3

Type : Label
Name : lblFullName
Caption : Full Name
Width : 1000
Left : 450
Top : 1500
Height : 200

e) Object 4

Type : Text Box
Name : txtFName
Width : 3000
Left : 1600
Top : 400
Height : 200
Text :

f) Object 5

Type : Text Box
Name : txtLName
Width : 1600
Left : 1600
Top : 950
Height : 200
Text :

g) Object 6

Type : Text Box
Name : txtFullName
Width : 5000
Left : 1600
Top : 1500
Height : 200
Text :

h) Object 7

Type : Command Button
Name : cmdFullName
Caption : Display Full Name
Width : 2500
Left : 450
Top : 2500
Height : 200



1.6 Assigning code segments to controls

Open the project developed in the previous example.

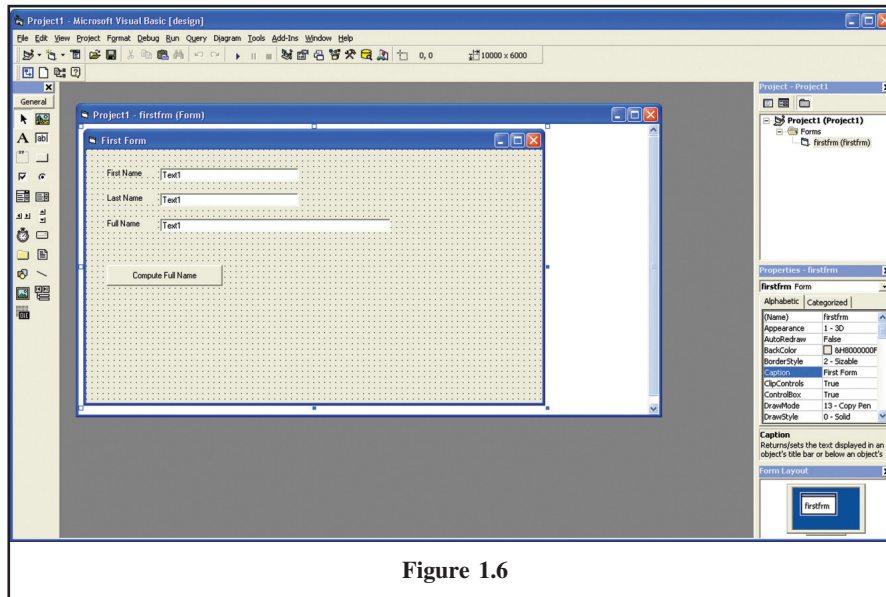


Figure 1.6

Now double click on the “Command Button” control with the caption “Create Full Name”. This will result in the following window on the screen.

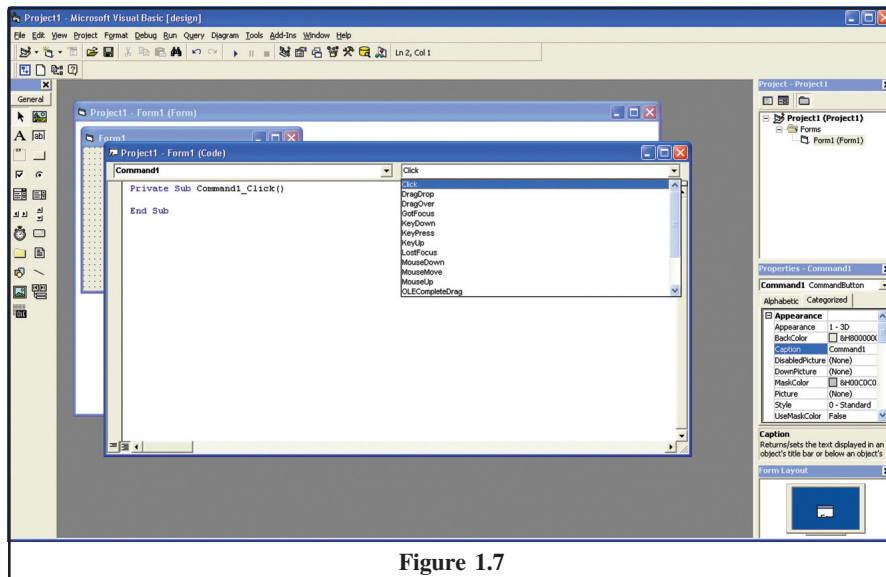
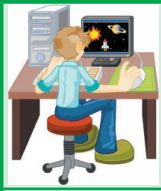


Figure 1.7

This window is called the “Code Window”. The Code Window allows you to add code segments for controls.



Activity 1.13

Observe the following code template that appears on the code window.

```
Private Sub cmdFullName_Click ()  
End Sub
```

Modify this template by adding a body to this Sub-program as below.

```
Private Sub fcmdFullName_Click ()  
    txtFullName.Text =txtFName.Text & " " & txtLName.Text  
End Sub.
```

After that, save this project.

You can execute this application by executing the command “Run” in the main menu of the IDE. Type the first name and the last name of someone in the text boxes and press the command button “Display Full Name” to see how the program you have just coded work on your input.

A VB application can be considered as an aggregation of a number of VB Sub Programs. Therefore, it is essential to know how to code a VB Sub Program.



The Composition of a Visual Basic Application

In Visual Basic, the basic building block of an application is a form which is simply a window that provides the user interface of the application. An application need not to be limited to a single form. Depending on the application requirement it may make use of many forms. When the IDE of the VB is launched it creates a form automatically. If the application needs additional forms, new forms can be created by executing the commands **Project** → **Add Form** from the main menu of the IDE.

Typically an application is created by adding controls onto a form. Some of the commonly used controls can be found in the Toolbox.

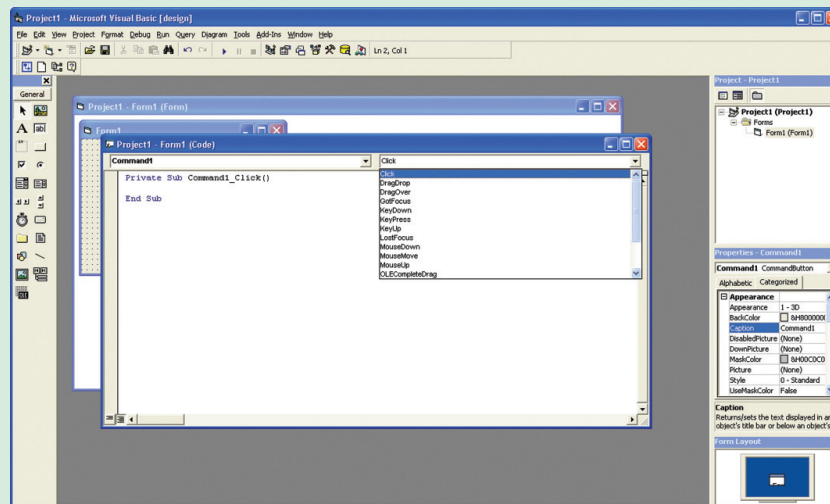
There are events affiliated with controls. Procedure explains the way these activities should occur according to the program. Procedure can be named as a program comprised of controls.

When a VB application is executed, the form(s) of the application together with its controls appears as a window on the computer screen. The user can interact with the application by performing the defined actions such as 'Click', 'Move', 'Drag' on the controls. When the user works on the control, the procedure relevant to that is executed.



Activity 1.14

Create a Command Button Control on a VB Form. Now double click on this Command Button to get the Code Window of the IDE as shown in the figure below. All events defined for a Command Button Control appears on the pop-down menu on the top right hand side of this Code Window. Observe different events defined for a Command Button Control. Select a few of these events, one after the other, and observe the code templates generated by IDE on the code window.





Visual Basic Reserved Words / Key Words

Like many other languages, Visual Basic also provides a large collection of pre-defined commands, functions, and methods. The names assigned to these pre-defined commands, functions and methods are known as “*reserved words*” or “*keywords*”. For example ‘Private’, ‘Sub’, ‘End’, ‘If’, ‘End If’ are a few examples of such reserved words. Usage of reserved words should conform to their definitions. For example the keyword ‘Sub’ can only be used to indicate the start of a subprogram and ‘End Sub’ to mark the end of a subprogram. Keywords are not allowed to be used as names for the other programming elements such as variables or procedures. When you code a program in the VB Code Window, the keywords you typed would appear in blue.



Visual Basic Procedures

VB program mentioned earlier consists of number of procedures. VB provides three types of Procedures.

- 1) Event Procedures
- 2) General Procedures
- 3) Function Procedures

Event Procedures are linked with controls on a Form of the application. Event procedures should be declared with the key word ‘Private Sub’ and can be executed only through the controls of the specific form to which the procedure is bound.

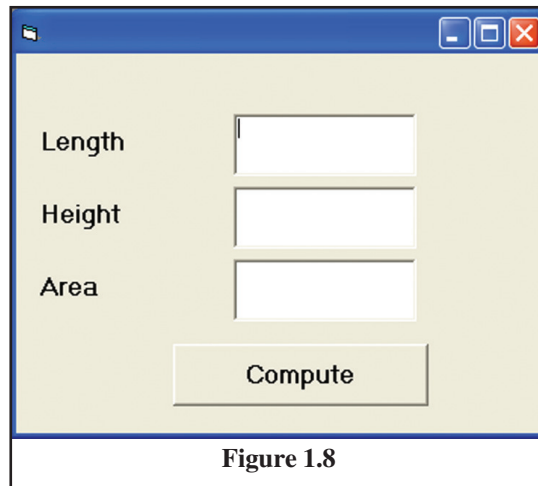
The procedures not directly linked with controls are called General Procedures. General Procedures can be declared by using the keywords ‘Private Sub’ or ‘Public Sub’. The ‘Public Sub’ procedures can be executed (called) from anywhere in your application whereas ‘Private Sub’ procedures are only accessible in the context (for example the form) in which it is declared.



Activity 1.15

Find out how the Function Procedures differ from the other types of procedures. Find out the keywords to be used to include a Function Procedure in a VB program.

Let us try to develop a VB application by using a Function Procedure to compute the area of a rectangle given its length and width. We can construct the interface of our application as below.



In the above interface the controls with the captions 'Length', 'Height' and 'Area' are of type 'Label Control' whereas the controls displayed in front of these three controls are of type 'Text Box Control'. Finally, the control with the caption 'Compute' is of type 'Command Box Control'. To do this, commands should be applied to the control 'compute'.

When a user clicks the "Compute" control after entering values for the Length and Height "Text Box Controls", the application should compute the area and the result should be displayed in the Area "Text Box Control".

The following Procedure can be linked with the 'Click' event of the 'Compute' control to perform these actions.

```

Private Sub cmdCompute_Click()
    L = Val(txtLength.Text)
    H = Val(txtHeight.Text)
    Area = L * H
    txtArea.Text = Area
End Sub

```

Observe the following characteristics in the above VB code.

1. The code starts with the declaration key word 'Private Sub' and concludes with the matching key word 'End Sub'. Therefore, the code segment defines a procedure. Since the name of this procedure ends with the event name '_Click ()' it is connected with the 'Click' event of a control. Thus the procedure can be identified as an 'Event Procedure'. This 'Event Procedure' is declared as a 'Private' procedure. Therefore, it can be executed only through a control created on the associated form.
2. To identify this procedure, the name 'cmdCompute_Click()' is given. The name 'cmdCompute' is the name of the command box while '_Click' is given automatically by VB to denote that it is relevant to the click event.
3. The body of a procedure comprises a collection of commands. Generally, these commands are sequenced in such a way as to have a single command in a line of the code. The code in a line is referred to as a 'Statement'.
4. The first executable statement in the code is

```
Length = Val (txtLength.Text)
```

The operator '=' in this statement is called the 'Assignment Operator'. This operator assigns the value obtained by simplifying the section in the right hand side to the variable on the left hand side. The Val(txtLength.Text) construct calls the function named 'Val' with the parameter 'txtLength'. The construct 'txtLength' is used to get the 'text' property value of the control named 'length'. The

operator '=' in this statement is called the assignment operator. In VB, such symbols are called operators. The 'L' in the left hand side is a variable. Variables can be considered as names which can be called to a location in the main memory. The operator '=' assigns the value obtained by simplifying the section in the right hand side to the variable on the left hand side. In the construct 'Val(txtLength.Text)', 'txtLengh' implies to get the value of the control 'Length'. The value returned by 'length.text' is of type text and cannot be used in a mathematical operation such as *. Therefore, the 'Val' function is called on this text value to convert it into a numerical value. Thus, the value of a control can be obtained by the structure:

Control Name. Property Name

5. In the construct 'Area=L*H, the symbol * shows multiplication. What happens here is the values of 'L' and 'H' are taken from the memory and the result obtained from the multiplication of them are stored in the 'Area' variable.
6. The last statement of this code is, 'txtArea.Text=Area'. This is to assign the value of variable 'Area' into the property 'text' in the control 'txtArea'.

When going through the code you may observe the following facts.

1. Procedure is a collection of statements.
2. Variables can be used in a procedure to hold values temporarily.
3. Statements are used to express the activities to be done by VB. Such statements are comprised of variables and various VB operators.
4. To access or to change the value associated with a particular property of a control can be done by using the naming convention 'ControlName. PropertyName'.

The code of the above application can be combined as an aggregation of a 'Event Procedure' and a 'General Procedure' as below.

```

Private Sub cmdCompute_Click ()
    Call ComputeArea (Val(txtLength.Text),Val(txtHeight.Text))
End Sub


```

```

Sub ComputeArea (L,H)
    Area=L*H
    txtArea.Text=Area
End Sub

```

The 'ComputeArea' procedure shown above is not connected to any event. Therefore, it defines a 'General Procedures'. Since it is not a 'Private' procedure it can be called from anywhere in the application code.



Activity 1.16

Develop the above VB program. Observe the result after compiling that.



Comments

Comments are the narrative description of program code embedded in the program itself. Unlike commands, comments cannot be executed. The main object of embedding comments in a program is to make the program more understandable. Comments can be included in a VB code by using the symbol ' ' or the reserved word 'rem'. When VB detects one of these at a particular position of the code, the entire text appearing on that line after this symbol till the end of the line is treated as a comment. If a comment extends to multiple lines each comment line should be started with the symbol ' '. In VB IDE comments appear in green colour.

Example: Program 8

```
' -----  
' sub program to add two numbers  
' -----  
  
Private Sub cmdAddNos_Click()  
  
    Sum=0  
  
    ' Add two numbers  
  
    Sum = Val(txtFirst.Text) + Val(txtSecond.Text)  
  
    txtTotal.Text = Sum ' Print the Result  
  
End Sub
```



Activity 1.17

1. What are the comments included in the previous example.
2. What will be the result if you change the line

```
sum = val(txtFirst.text)+Val(txtSecond.text)  
'sum = val(txtFirst.text)+Val(txtSecond.text)
```



Visual Basic Data Types

In many programming languages data allowed to be used in a program are classified into several types based on their nature the size of memory required to store the data and the type of operations that can be performed on the data. Different classes of data allowed in a programming language are called 'Data Types'. Data types allowed in a VB program are described below.

1. Boolean - A value of Boolean variable has only two states, **True** and **False**. VB also equates the numeric value 0 to the Boolean value 'False' and all other numeric values to Boolean 'True'. Three widely used operators defined on the Boolean data type are AND', 'OR' and 'NOT'.
2. Numeric - All numbers are treated as of type numeric.

Based on the range of values to be stored, this data type is further subdivided into the following 5 data types

1. Byte
2. Integer
3. Long
4. Single
5. Double

Out of these five data types, 'Byte', 'Integer' and 'Long' are defined for integer numbers whereas the other two types are defined for decimal numbers.

3. String : Data comprising characters (symbols, numbers) falls into this type. Character aggregations such as "Sri Lanka", "Low" are two examples for data values of this type. Data values of this type should always be enclosed inside the symbols " and ". Operators defined on numeric data types cannot be performed on data of this type. There are many operators defined on strings. One of the commonly used operator on strings is the string concatenation, identified by the '&' symbol. This operator combines two strings to a single string.

Example : Consider the following string concatenation

"My name is" & "Saman"

The '&' operator combines the two strings "My name" and "Saman" and produces the single string "My name is Saman" as the result.

- CURRENCY** - This data type can be used to store and manipulate currency (money) values and the values of this type can have up to 4 digits after the decimal point.
- DATE** - This data type can be used to store and manipulate dates and times.
- Variant** - All above data types are subtypes of this common data type.



Variables

When writing a program you often need to store values temporary in the computer storage (RAM). Like most other programming languages, Visual Basic uses variables for storing values temporary. A variable can be viewed as a symbolic name constructed to access a particular block of memory in the main storage (RAM). Once a variable is created the designated block of memory can be accessed by using this symbolic name (variable name). In other words the variable name can be used to store values or to retrieve values from the designated memory block.

Example 1 :

Consider the following VB assignment.

$A = 25$

The actions taken by VB when executing the above statement are given below

01. Acquire enough storage from the main memory for a variable named A.
02. Bind the symbolic name A to this acquired memory block.
03. Store the value 25 in the memory block now identified as A.

Example 2 :

$$A = 10$$

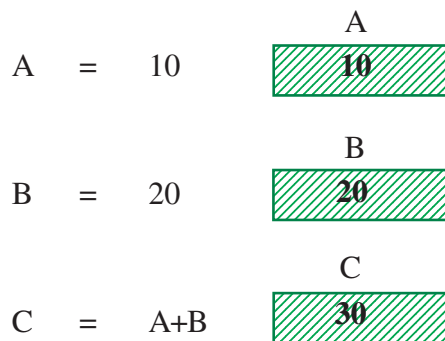
$$B = 20$$

$$C = A + B$$

The actions performed by VB in executing the above code segment can be described as below:

01. Acquire a sufficient segment of memory for a variable and assign the symbolic name A for this memory block. Store the value 10 at this memory block.
02. Similarly acquire a segment block for the variable B and store the value 20 at this memory block.
03. Retrieve the values stored for the variables A and B from the respective memory segments, add them together, store the results in a separate memory segment and assign the symbolic name C for this new memory segment.

The changes that take place in memory, when executing the code can be visualized as below.





Important Facts to Remember on Variables

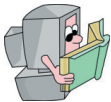
1. A variable can be considered as a symbolic name assigned to some memory segment in the main memory (RAM) of the computer.
2. Different values can be stored in the memory segment designated for a variable at different times. However, when a new value is assigned to a variable the value stored previously will be overwritten by the new value.

Consider the following example

```
A = 10
A = 20
```

When the first line of this code is executed, VB will store the value 10 at the memory segment designated to the variable A. When the second line is executed, VB will replace the previously stored value 10 with the new value 20.

3. Memory is allocated to variables temporarily. Upon the completion of a program, all memory allocated to the program variables are released back to the operating system of the computer.



Naming Visual Basic Variables

When creating names for variables in a program, a specific set of rules has to be followed. Typically the set of rules to comply with depends on the programming language.

All Visual Basic variable names should comply with the following rules.

1. Should start with an English letter.
2. The characters following the first letter can be a mixture of letters from the English alphabet, numbers or the symbol '_' (Underscore).
3. The maximum number of characters a name can contain is 255.
4. The VB command names cannot be used as variable names.

The set of names given for Visual Basic commands are called **Reserved Words**.



Activity 1.18

Which of the names listed below can be used as VB variable names? Explain why the other names cannot be used as variable names.

- | | |
|------------|----------|
| 1) A5 | 6) Name? |
| 2) MyName | 7) abc |
| 3) My_Name | 8) 1f |
| 4) My Name | 9) 5A |
| 5) _Name | 10) A_5 |



Variable Declaration

In VB data types of variables can be defined at the beginning of a program. The statements that define variables are called 'Variable Declarations'.

The keywords 'Dim' together with 'As' is used to declare VB variables as below;

Example : Dim Value1 As Boolean
Dim Value2 As Boolean
Dim Name As String

In the above example two variables namely 'Value1' and 'Value1' are declared as of type Boolean and another variable 'Name' is declared as of type String.

The variables not declared explicitly by using the keyword 'Dim' are automatically treated as of type 'Variant'.



Activity 1.19

In a VB program it is not necessary to declare the type of a variable before its use. However, there are many advantages in defining the type of a variable before its use. Find out these advantages.



Activity 1.20

What are the operations defined on the data types 'Numeric' and 'String'. Study the rules to be followed when using these operators in a program.



Mathematical Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation
\	Integer division
Mod	Modulus



1.8 Operator Precedence

Consider the following statement

$$5-3 \ 2$$

This statement can be simplified in two different ways as follows;

$$5-(3 \ 2) = -1 \qquad (5-3) \ 2 = 4$$

It is clear from the previous simplifications that a statement with different operators can result in many different answers if the simplifications are done in different ways. In a program a simplification of a statement cannot result in different answers at different times. To eliminate this problem priorities are pre-defined on operators. The priorities defined on operators are called 'Operator Precedence'. In VB the precedence defined on the main mathematical operators are as below.

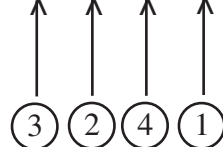
Operator	Precedence order (low to high)
1)	()
2)	+ , -
3)	* , /
4)	= ^

Example :

Consider the following VB statement

$$5 \ 2 \ 3-5 \wedge 2$$

$$((5+(2*3)-(5^2)))$$



$$= ((5+(2*3)-25))$$

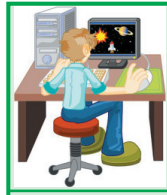
$$= ((5+6)-25)$$

$$= 11-25$$

$$= -14$$

How VB simplifies this expression by following the precedence rule is explained below by using parenthesis.

The default operator precedence defined by a programming language can be easily overridden by using parenthesis.



Activity 1.21

Discuss how parenthesis could be used to change the order of computation



Conditional Operators

The result of a conditional operator is always of the type Boolean (True or False).

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Example :

Conditional Statement	Result
1) $5 = 6$	False
2) $3 + 5 = 4 + 4$	True
3) $5 < 6$	True
4) $6 < 5$	False
5) $5 < > 6$	True
6) $x < = 6$	True
7) $x < = 5$	False

When the conditional operations are applied on complex statements the complex statements are simplified to single values prior to the application of the operator.

Example :

Consider the following conditional expression

$$(5 * 3) + 2 > (6 / 2) * 5$$

This expression is simplified first as below

$$17 > 15$$

Since 17 is greater than 15, the condition expression finally results in the Boolean value 'True'.



Logical Operators

These operators are defined on the logical data type. There are three widely used logical operators namely 'AND', 'OR' and 'NOT'. The definitions of these operators are given in the following truth tables.

A detailed explanation on Boolean variables and operations can be found in 4th chapter of the grade 10 book.

Let 'a' and 'b' be two Boolean variables:

AND	a	b	a AND b
	True	True	True
	True	False	False
	False	True	False
	False	False	False

Example :

Consider the following Boolean statement


$$(2 > 3) \text{ AND } (3 > 2)$$

The main operator of this statement is 'AND', which can work only on Boolean values. Therefore, the two sub-expressions $(2 > 3)$ and $(3 > 2)$ on both sides of the 'AND' operators should be computed prior to the application of the 'AND' operator. Simplification of these two sub-expressions $(2 > 3)$ and $(3 > 2)$ results in the Boolean values 'False' and 'True'. Computation of the resulting simplified statement 'False AND True' will produce the value 'False' as the final answer to the original statement.

	a	b	a OR b
OR	True	True	True
	True	False	True
	False	True	True
	False	False	False

	a	NOT a
NOT	True	False
	False	True

The definitions of the 'OR' and 'NOT' Boolean operators are given in the following truth tables.



Activity 1.22

The following application interface can be developed to show the effects of the 'AND' and 'OR' Boolean operators.

Control Number	Type	Name
1	Combo Box	cboAndLeft
2	Command Button	cmdAnd
3	Combo Box	cboAndRight
4	Label	lblAndEqual
5	Text Box	txtAnd
6	Combo Box	cboOrLeft
7	Command Button	cmdOr
8	Combo Box	cboOrRight
9	Label	lblOrEqual
10	Text Box	txtOr

Change the 'Data Format' property of the controls numbered 1,3,5 and 7 as 'List' and add the two values 'True' and 'False' as the list values of these controls. You can enter many values for the 'List' property of a 'Combo Box' by entering the first value at the 'list' property and the other values by hitting the keys 'Ctrl-Enter'.

The following procedures can be linked with the controls 2 and 6.

Program 9

```
Private Sub cmdAnd_Click()  
    Dim LeftAndValue As Boolean  
    Dim RightAndValue As Boolean  
    Dim ResultAnd As Boolean  
    LeftAndValue = cboAndLeft.List(cboAndLeft.ListIndex)  
    RightAndValue = cboAndRight.List(cboAndRight.ListIndex)  
    ResultAnd = LeftAndValue And RightAndValue  
    txtAnd.Text = ResultAnd  
  
End Sub  
  
Private Sub cmdOr_Click()  
    Dim LeftOrValue As Boolean  
    Dim RightOrValue As Boolean  
    Dim ResultOr As Boolean  
    LeftOrValue = cboOrLeft.List(cboOrLeft.ListIndex)  
    RightOrValue = cboOrRight.List(cboOrRight.ListIndex)  
    ResultOr = LeftOrValue Or RightOrValue  
    txtOr.Text = ResultOr  
  
End Sub
```


Click on the Command Button 'And' or 'Or', no sooner the program starts and observe what happens there. State the reasons for the changes.

Note the result obtained when it is clicked on the Command Button 'And' and 'Or' after changing the values of the Combo Boxes.

Now, note down the changes you see when the below mentioned code is added to the program and run as previous.

```
Private Sub Form_Load()  
    cboAndLeft.ListIndex = 0  
    cboAndRight.ListIndex = 0  
    cboOrLeft.ListIndex = 0  
    cboOrRight.ListIndex = 0  
End Sub
```



Flow Control

The execution order of statements in a program is called the flow control. Typically the statements are executed in the same order they appear in the program. However, this sequence of execution can be changed by using different 'Control Structures'.

When formulating control structures a certain set of rules, specific to each structure, has to be followed. The set of rules defining the layout of a valid control statement is known as the 'Syntax' of the statement.



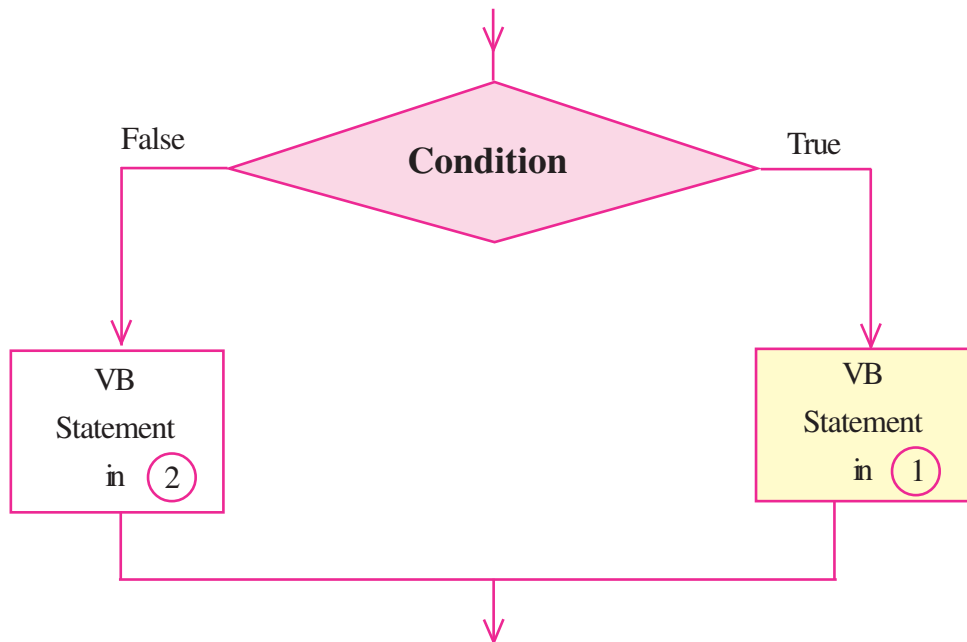
IF...Then.....Else control Structure

If Condition Then
 VB Statements ———— (1)

Else
 VB Statements ———— (2)

End IF

Flow Chart

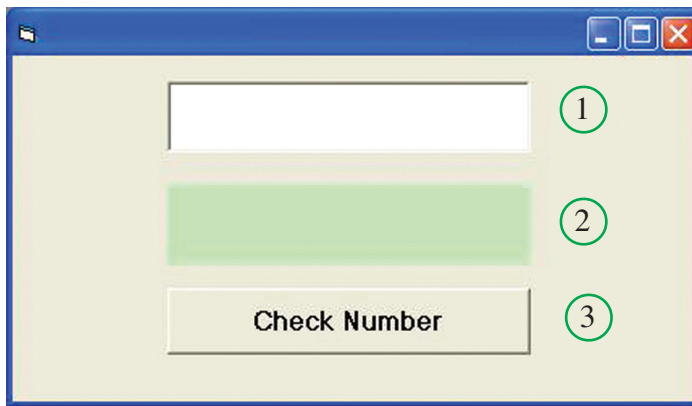


'IF', 'THEN', 'ELSE' and 'ENDIF' are keywords in this control structure. And 'Condition' should result in a Boolean Value. 'VB statements' is a collection of VB commands.

The process of this structure can be explained as follows;

- a) Simplify the 'Condition' and obtain a single Boolean result
- b) If the result of the previous evaluation is 'True' perform all the commands in the section labeled 1.
- c) If the result of the previous evaluation is 'False' perform all the commands in the section labeled 2.

Program 10 - Let us develop a program to find whether a number is odd or even.



Control Number	Type	Name
1.	Text Box	txtNo
2.	Label	lblDisplay
3.	Command Button	cmdCheckNo

```
Private Sub cmdCheckNo_Click()  
    Dim No As Integer  
    No = Val(txtNo.Text)  
    If No Mod 2 = 0 Then  
        lblDisplay.Caption = "Even Number"  
    Else
```

```
lblDisplay.Caption = "Odd Number"
```

```
End If
```

```
End Sub
```



1.10 Select.. Case

Select ... Case control structure can be considered as an aggregation of several If ... Then ... Else control structures.

Select.. Case **Structure**

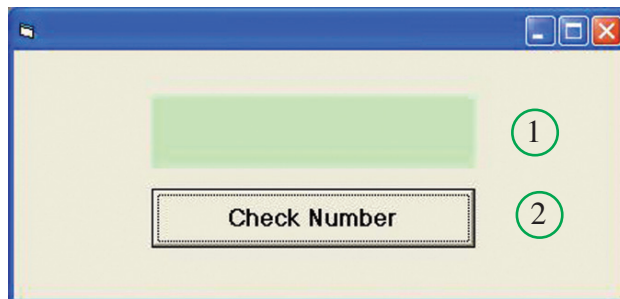
```
Select Case expression ————— (a)
Case Value 1
    VB Statements ————— (1)
Case Value 2
    VB Statements ————— (2)
    .....
    .....
Case Else
    VB Statements ————— (n)
End Select
```

Semantic

'Select Case', 'Case', 'Case Else' and 'End Select' are the keywords in this structure and the data type returned by the 'expression' in the section 'a' must be compatible with all data values 'Value1', 'Value2' used in each individual case statement. When executing this structure 'expression' is simplified to obtain a single value. This value is then compared with the 'Value1', 'Value2' in that order till a

matching value is found. If a matching value is found then all 'VB Statements' within that case branch are executed and the control passed to the first statement after the keyword 'End Select'. If VB could not find a matching value then all 'VB Statements' within the last 'Case Else' branch are executed and the control passed to the first statement after the keyword 'End Select'.

Program 11 : Let us develop a program to find whether a number is plus or minus



Control Number	Type	Name
1.	Label	lblDisplay
2.	Command Button	cmdCheckNo

```

Private Sub cmdCheckNo_Click()
    Dim N As Single
    N = InputBox("Enter Number")
    Select Case N
    Case Is < 0
        lblDisplay.Caption = "Negative Number"
    Case Is > 0
        lblDisplay.Caption = "Positive Number"
    Case Else
        lblDisplay.Caption = "Zero"
    End Select
End Sub

```



Activity 1.23

Draw a flow chart to show the process of the VB Select ... Case control structure.



1.11 Looping

The looping structures enable the same collection of statements to be executed many times.

For Next Loop

Structure

```
For variable = start value to final value [stepping value]
    relevant coding
Next variable
```

Here, the segment ' [stepping value]' is used only when necessary.

How For Next loop operates

In the For Next loop, first the relevant value is assigned to the variable. Values are assigned according to the way that numbers are increased one after the other, from the starting value to the final value. If any value is given to the step value, it is used to create a gap between two values. (i.e. Generally in For Next loop number increases in ascending order one by one, and for the numbers to increase by any other values, 'step' option can be used.

The stepping value is compulsory to use when the looping is developed to decrease the numbers.

The loop runs forward with the value assigned to the variable while executing the codes and when 'Next' word/line met, it goes to the 'For' line again. There, it is tested whether the value of the variable is the 'final value'. If it is the final value, looping comes to an end and the program goes to the line after 'Next'. When it is not so, the value of the variable is changed accordingly and goes into the loop. When it comes to the 'Next' word/line, it goes again to 'For' line and checks the variable value. This happens till the value of the variable becomes 'final value'.

For Next loop can be used for a process which is to be repeated number of times.

Example :

a) Sum= 0
For i= 0 to 10
 Sum= Sum + i
 Print i
Next
Print Sum

Numbers can be seen printing downwards on the form one after the other from 0 to 10 increasing one by one and finally the total 55 is also printed.

b) Sum= 0
For i= 0 to 10 Step 2
 Sum= Sum + i
 Print i
Next
Print Sum

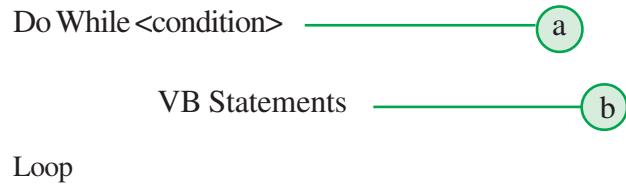
Here, numbers are printed downwards on the form from 0 to 10 increasing two by two (ie: 2,4,6,8 and 10) and finally the total 30 is also printed.

c) Sum= 0
For i= 10 to 0 Step -2
 Print i
Next

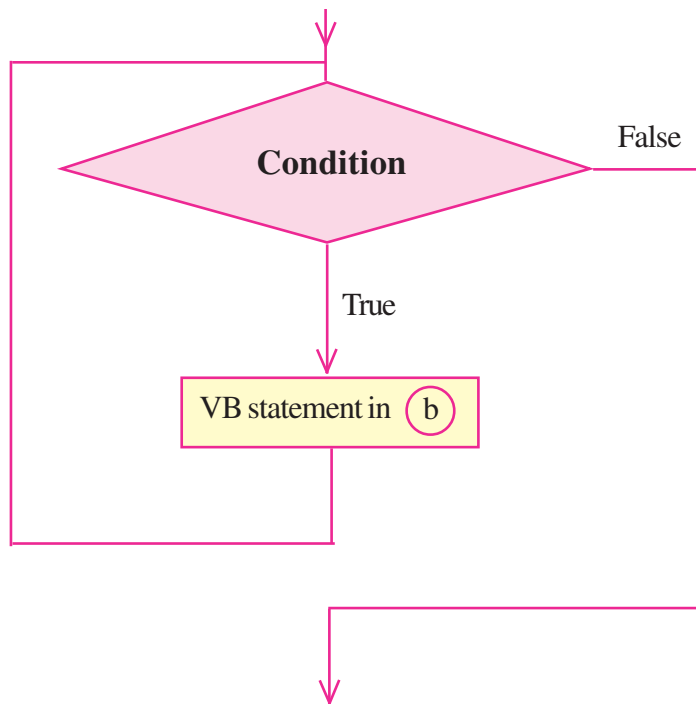
Here, numbers can be seen printing on the from downwards decreasing two by two (10,8,6,4,2 and 0)

Do While Loop

Structure



The VB statements in section 'B' is executed till the 'Condition' returns the value 'True'. When 'Condition' returns a 'False' the loop exits and the control moves to the first statement after the Loop keyword of the structure.



Do, while and Loop are the reserved words in the Do Loop control structure and the computation of the 'Condition' should return a Boolean value.



Activity 1.24

Start a new project, add a Command Button to the form, add the coding below to the Click event and observe the result.

```
Counter= 1
Do while counter<100
    Debug.Print counter
    counter=counter+1
Loop
```

When click on the Command Button, the result is shown on the Immediate Window. Observe the result with the following changes.

- a. **debug.Print counter** is changed as **Print counter**
- b. **debug.Print counter**
counter = counter + 1 is changed as **counter = counter + 1**
debug.Print counter



1.12 Arrays

Visual Basic provides different ways to combine data elements together. Such data aggregations constructed by structuring data elements together are called 'Data Structures'. "Arrays", 'Stacks' and 'Queues' are a few examples of such data structures.

An array is a data structure that can store several data elements of the same type. When you use an array, you can access multiple values stored in the array by the same name but by using a number called an index or subscript to distinguish them from one another. You can visualize an array as shown in the following diagram, which has been created to store 5 integers.

As in most of the programming languages, Visual Basic also provides methods to hold data temporarily on programs. Hence, “data structure” is a program which holds data temporarily. “Arrays”, “Stacks”, “Queues” can be cited as examples.

An array is a data structure which consists of a group of same data type elements that are accessed by indexing. It can be visualized as follows;

20	30	40	55	60	Marks
1	2	3	4	5	

The above structure can be cited as an array created to hold five integer numbers.

When creating such an array in a program a name should be given to the whole array structure. Let us consider ‘Marks’ as the name of the array in the previous example. Then, to retrieve or store data in the structure ‘Marks (Position)’ structure can be used.

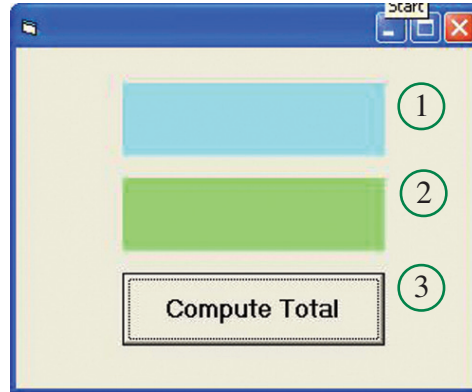
The name of the array is indicated by the first part of the structure (Eg:- Marks) Secondly, the value within brackets implies the position it takes in relevant to the first value.

Example:

Dim Marks (0 to 9) As Integer

This declaration defines an array named ‘Marks’ to store 10 data elements of type “Integer”. The first and last element of this item can be accessed by using the constructs “Marks (0)” and “Marks (9)” respectively.

Let us try to develop a VB application using an array to compute the sum of 10 given numbers. The following type of a window can be used as the user interface for this application.



Control Number	Type	Name
1.	Label	lblNumbers
2.	Label	lblTotal
3.	Command Button	cmdTotal

```

Private Sub cmdTotal_Click()
    Dim myNumbers(1 To 10) As Integer
    Dim i As Integer
    Dim total As Integer
    Dim numbers As String
    'No Numbers entered yet
    numbers = ""
    ' Read the 10 numbers to number array
    For i = 1 To 10
        myNumbers(i) = Val(InputBox("Enter the number " & i))
        lblNumbers.Caption = lblNumbers.Caption & myNumbers(i) & " , "
    Next i
    'Compute the total of numbers in the array
    total = 0
    For i = 1 To 10
        total = total + myNumbers(i)
    Next i
    'Display the total of numbers in the array
    lblTotal.Caption = "Total of the numbers is " & total
End Sub

```