✍ Programming in Visual Basic

It's time to learn a foreign language: Visual Basic. Until now, you created programs by placing controls on the form and setting property values for them. Visual programming like that is at the heart of Visual Basic, and visual nature of Visual Basic is what separates it from the more traditional text-based programming languages.

☞ Program Statement

A line of code in Visual Basic is called a *Program Statement*. A *Program Statement* is any combination of Visual Basic keywords, properties, functions , operators , and symbols are collectively create a valid instruction recognised by VB compiler.

☞ Data Types

The Visual Basic language works with all kinds of data. Now you will learn how to distinguish among the various data types.
Visual Basic manipulates and analyzes seven kinds of data. Given table describes the data types that Visual Basic supports. When you write a program, you need to decide which data type best fits your program's data values.

Data Type	Description
integer	Numeric values with no decimal point or fraction. integer values range from –32,768 t 32,767.
long	Integer values with a range beyond than that of integer data values. long data values ra from –2,147,483,648 to 2,147,483,647. Long data values consume more memory stora integer values, and they are less efficient. The long data type is often called long integ
single	Numeric values that range from –3.402823E+38 to 3.402823E+38. The single data ty is often called single-precision.
double	Single numeric values that range from –1.79769313486232E+308 to 1.79769313486232E+308. The double data type is often called double-precision.
Currency	Data that holds dollar amounts from –$922,337,203,685,477.5808 to $922,337,203,685,477.5807.
string	Data that consists of 0–65,500 characters of alphanumeric data. Alphanumeric means data can be both alphabetic and numeric. String data values may also contain special characters such as ^%@.
variant	Used for data stored in controls and for date and time values.
Boolean	Date type stores True/False values.
Date	Used for Date and Time values are stored internally in a special format.
Object	Used for an object variable to access the actual object.

☞ Defining Variables

A program can have as many variables as you need it to have. Before you can use a variable, you must create the variable by defining the variable first. When you define a variable, you tell Visual Basic these two things:

- ? The name of the variable
- ? The data type of the variable

Once you define a variable, that variable always retains its original data type.
The Dim statement defines variables. Using Dim, you tell Visual Basic

- ? that you need a variable
- ? Reminder division is Reminder division is what to name the variable
- ? what kind of variable you want

Dim—short for *dimension*—is a Visual Basic statement that you write in an application's Code window. Dim reserves the space for the variable.
Variables can be defined in two ways:

Dim VarName (In this declaration variable can hold data of any size or format.)

Or

Dim VarName AS DataType

VarName is a name that you supply. When Visual Basic executes the Dim statement at runtime, it creates a variable in memory and assigns it the name you give in the *VarName* location of the statement.

```
Dim A As Integer
Dim B As Double
Dim C As date
Dim D As String
Dim E As boolean
```

Reducing Dim with Variable Suffix Characters: There is a set of suffix characters for variable names that you can use to specify a variable's data type without having to define the variable first. Table 7.2 listed the suffix characters for constant values. Variables use the more complete version shown here:

Suffix	Data Type	Example
%	Integer	Age%
&	Long	Amount&
!	Single	Distance!
#	Double	KelvinTemp#
@	Currency	Pay@
$	String	LastName$

The statement Option Explicit cannot appear in the (general) section of the Code window because the suffix characters are not enough to define variables.

☞ Mathematical Expressions

Data values and controls are not the only kinds of assignments that you can make. With the Visual Basic math operators, you can calculate and assign expression results to variables when you code assignment statements that contain expressions.
It is easy to make Visual Basic do your math. Table describes Visual Basic's primary math operators. There are other operators, but the ones in Table will suffice for most of the programs that you write. Look over the operators. You are already familiar with most of them because they look and act just like their real-world counterparts.

☞ The primary mathematical operators.

Operator	Example	Description
+	Net + Disc	Adds two values
-	Price - 4.00	Subtracts one value from anothe
*	Total * Fact	Multiplies two values
/	Tax / Adjust	Divides one value by another va
^	Adjust ^ 3	Raises a value to a power
& or +	Name1 & Name2	Concatenates two strings

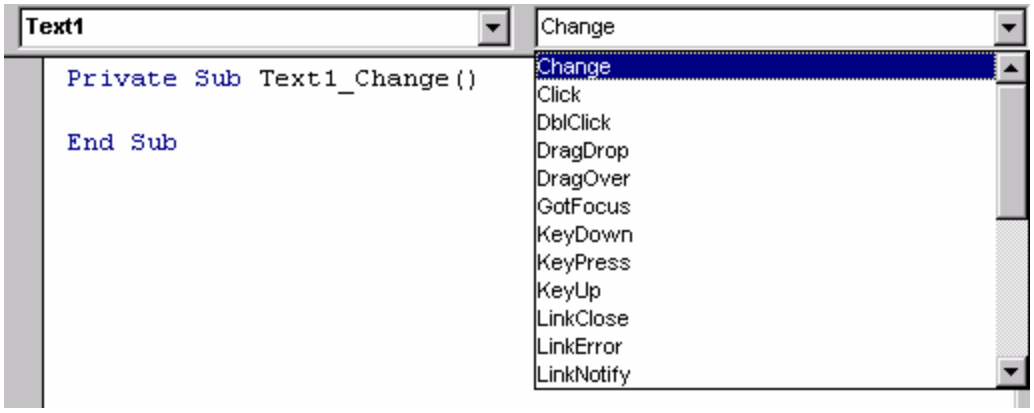
☞ Operator precedence

The order in which mathematical operators used when formula is solved.

Operators	Order of Precedence
( )	Values within parentheses are always evaluated first.
^	Exponentiation (raising a number to a power) is second.
-	Negation (creating a negative number) is third.
* /	Multiplication and division is forth.
\	Integer division is fifth.
Mod	Reminder division is Sixth.
+ -	Addition and subtraction are last.

☞ Event-Driven Progamming

Each object in visual basic has a predefined set of events it can respond to. These events are listed for each object in the procedure drop-down list box in the code window. You can write an event procedure for any of these events, and if that event occurs in the program, Visual Basic will execute the event procedure that is associated with it . Partial listing of the events for Text box object are shown below:



☞ Events Common to Many Controls

Event	Occurs When
Change	The user modifies the text in a text box or combo box.
Click	The user clicks and object with the primary mouse button (usually the left button).
Dbclick	The user double-clicks an object with the primary mouse button.
DragDrop	The user drags a control to another location.
DragOver	An object is dragged over a control.
GotFocus	An object receives the focus.
KeyDown	A key is pressed while and object has the focus.
KeyPress	A key is pressed and released while an object has the focus.
KeyUp	A key is released while an object has the focus.
LostFocus	The focus has left an object.
MouseDown	A mouse button is pressed while the mouse pointer is over an object.
MouseMove	The mouse cursor is moved over an object.
MouseUp	A mouse button is released while the mouse pointer is over an object.

☞ The Relational Operators

Visual Basic supports the use of six operators that produce true or false results based on data values. you can combine them operators with the If statement to add power to your programs. You use the relational operators to compare data values. They are easy to use. If you take any two numbers, one number is always be greater than, equal to, or less than the other.

Operator	Usage	Description
>	Sales > Goal	The greater than operator. Returns true if the value on the left side of > is numerically or alphabetically greater than the value on the right. Otherwise, false.
<	Pay < 2000.00	The less than operator. Returns true if the value on the left side of < is numerically or alphabetically less than the value on the right. Otherwise, false.
=	Age = Limit	The equal to operator (sometimes called the equal operator). Returns true if the values on both sides of = are equal to each other. Otherwise, false.
>=	FirstName >= "Mike"	The greater than or equal to operator. Returns true if the value on the left side of >= is numerically or alphabetically greater than or equal to the value on the right. Otherwise, false.
<=	Num <= lblAmt.Caption	The less than or equal to operator. Returns true if the value on the left side of <= is numerically or alphabetically less than or equal to the value on the right. Otherwise, false.
<>	txtAns.Text <> "Yes"	The not equal to operator. Returns true if the value on the left side of <> is numerically or alphabetically unequal to the value on the right. Otherwise, false.

Keep Each Side a Consistent Data Type: The expressions on both sides of a relational operator must have the same data type or at least compatible data types. In other words, you cannot compare a string to a numeric data type. If you try, you will get a Type mismatch error.

☞ The *If* Makes Decisions

The If statement uses the relational operators to test data values. It perform one of two possible code actions, depending on the result of the test.
The If statement makes decisions. If a relational test is true, the body of the If statement executes. Here is one format of If:

```
If relationalTest Then
    One or
    more Visual Basic statements
End If
```

The End If statement informs Visual Basic where the body of the If statement ends.

```
If (txtSales.Text > 5000.00) Then
    Bonus = Val(txtSales.Text) * .12
    CostOfSales = Val(txtSales.Text) * .41
    ReorderCost = Val(txtSales.Text) * .24
End If
```

☞ Handling False Conditions

Whereas If executes code based on the relational test's true condition, the Else statement executes code based on the relational test's false condition. Else is actually part of the If statement. The Else statement, part of an extended If statement, specifies the code that executes if the relational test is false. Here is the complete format of the If statement with Else:

```
If relationalTest Then
    One or more Visual Basic statements
Else
    One or more Visual Basic
    statements
End If
```

☞ Logical Operators

Three additional operators, And, Or, and Not, look more like commands than operators. And, Or, and Not are called *logical operators*. They enable you to combine two or more relational tests. Table describes the logical operators.

Operator	**Usage**	**Description**
And	If (A > B) A	Returns true if both sides of the And are true. Therefore, A must be
	D)	than B and C must be less than D. Otherwise, the expression return
		result.
Or	If (A > B) O	Returns true if either side of the Or is true. Therefore, A must be gr
	D)	than B or C must be less than D. If both sides of the Or are false, th
		expression returns a false result.
Not	If Not(Ans =	Produces the opposite true or false result. Therefore, if Ans holds "
		the Not turns the true result to false.

For example:

```
If (Sales > 5000.00) And (UnitsSold > 10000) Then
   Bonus = 50.00
End If
```

☞ Multiple Choice with *Select Case*

The If statement is great for data comparisons in cases where one or two relational tests must be made. When you must test against more than two conditions, however, the If becomes difficult to maintain.
Visual Basic supports a statement, called Select Case, that handles such multiple-choice conditions better than If-Else. Here is the format of the Select Case statement:

Select Case *Expression*
Case *value*
One or more Visual Basic statements
Case *value*
One or more Visual Basic statements
[Case *value*
One or more Visual Basic statements]
[Case Else:
One or more Visual Basic statements]
End Select

The format of Select Case makes the statement look as difficult as a complex nested If-Else, but you will soon see that Select Case statements are actually easier to code and to maintain than their If-Else counterparts.

☞ CONTROL UNDER VISUAL BASIC

Now you're *really* ready to write powerful programs! You will learn some controls, you've defined some variables, and now will write some programs that make decisions. It's now time to learn how to write programs that perform repetitive data processing. When a computer does something over and over, the computer is said to be *looping*.

- The Do While Loop
- The Do Until Loop
- The Other Do Loops
- The For Loop

☞ The Do while Loop

The Do statement supports several different loop formats. The Do While loop is perhaps the most common looping statement that you'll put in Visual Basic programs.
The Do While statement works with relational expressions just as the If statement does. The relational expression controls the looping statements. Here is the format of the Do While loop:

Do While (relational test)
Block of one or more Visual Basic statements
Loop

Or

```
Do
Block of one or more Visual Basic statements
Loop While (relational test)
```

The Do While loop continues executing a block of Visual Basic statements as long as a *relational test* is true. As soon as the *relational test* becomes false, the loop terminates.

```
Do While ((Age < 10) Or (Age > 99))
   Beep
   MsgBox "Your age must be between 10 and 99", MB_ICONEXCLAMATION,
   "Error!"
   StrAge = InputBox("How old are you?", "Age Ask")
   Check for the Cancel command button
   Age = Val(StrAge)
Loop
```

☞ The Do Until Loop

Until keywords can be used in Do loops to cycle *until* a certain condition is true. The Until keywords can be used at the top or bottom of a Do loop to test the condition.

```
Do
   Block of one or more Visual Basic statements
Loop until  (relational test)
```

The Do Until loop continues executing a block of Visual Basic statements as long as a *relational test* is true. As soon as the *relational test* becomes false, the loop terminates.

```
Do
  Impname= Inputbox ("Enter your name or type Done to quit.")
   If impname <> "Done" then
     Print impname
   End if
Loop until  Impname = "Done"
```

In above example, do loop uses the until keyword to loop repeatedly until user enters the word *Done* in the inputbox.

☞ For Loop

A For…Next loop lets you execute a specific group of program statements a set number of times in an event procedure.
Syntax for For…Next loop :

For *variable* = *start* to *end* step <integer value>
  Statements to be repeated
Next *variable*

Example:

```
For I= 5 to 20 step 5
  Print I
Next  I
```

It will print the following sequence of numbers on the form:
5
10
15
20

☞ Array

Arrays can be declared in a program code as the variables are declared. If an array declared *locally* , it can be used only in the procedure in which it is declared. If an array is declared *Publicly* in a standard module, it can be used anywhere in the program.

Syntax for a public Fixed-size array is :
Public Arrayname(Dim1elements, Dim2elements, ….) As datatype

- *Public* is a Keyword that creates a global array
- *ArrayName* is the variable name of the array.
- *Dim1elements* is the number of elements in the first dimension of the array.
- *Dim2elements* is the number of elements in the second dimension of the array.
- *Datatype* is a keyword corresponding to the type of data that will be included in the array.

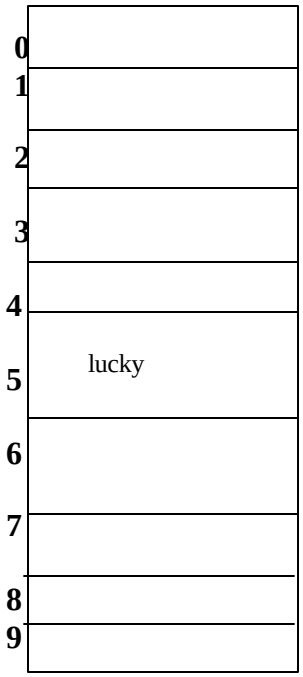
Note : To declare array locally in an event procedure, replace the *public keyword* with the *static keyword* and place the declaration inside an event procedure. Local array can be used only inside the procedure in which they are declared.

Example: To declare a Public one-dimensional array named Employees that has room for 10 Employee names the statement will be:

Public Employees(9) as string

When array is created, Visual Basic sets aside room for it in memory.

Employees



Employees(5)="Lucky"

Fixed-size arrays: Arrays that contains a set number of elements are called fixed-size arrays. For example:

Public scoreboard(1,8) as variant

Dynamic arrays : Arrays that contains variable number of elements (arrays that can be expand during the execution of the program) are called Dynamic arrays.

For example:

Public Temperatures() as variant

Specify the name and type of the array in the program in the design time, omitting the number of elements in the array as shown above.

Example of Dynamic array:

```
Public temperatures as variant
Public days as integer

Days = Inputbox("How Many days?" , "Create Array")
```

```
If days >0 then
   Redim  Temperatures(days)     " Redim statement sets the size of the temperatures
                                 array at the runtime by using the days variable"
   Prompt$ = "Enter the High Temperature."
   For I% = 1 to days
     Title$ = "Days  " &  I%
     Temperatures(  I%) = Inputbox(Prompt$, Title$)
   Next I%

   This program will aassign the values
    Temperatures(1)= Days 1
   Temperatures(2)= Days 2
   Temperatures(3)= Days 3
   .
   .
   …….So on.
```

Few Examples : Given in Classroom Assignments at the end of Reference Material.