VB .NET PROGRAMMING

IT Systems
282
Polgahamula Junction
Peradeniya

Mr. W.M.A.K.B. Giragama
(Software Developer)

Introduction

With the release of its new .NET platform, Microsoft also released a new version of the Visual Basic language, Visual Basic .NET. VB .NET is a from-the-ground-up rewrite of Visual Basic that not only adds a number of new features, but also differs significantly from previous versions of Visual Basic. From a high-level view, two of these differences are especially noteworthy:

- Until the release of VB .NET, Microsoft focused on creating a unified version of VBA, the language engine used in Visual Basic, which could serve as a "universal batch language" for Windows and Windows applications. With Version 6 of Visual Basic, this goal was largely successful: VB 6.0 featured VBA 6.0, the same language engine that drives the individual applications in the Microsoft Office 2000 suite, Microsoft Project, Microsoft FrontPage, Microsoft Visio, and a host of popular third-party applications such as AutoDesk's AutoCAD and Corel's WordPerfect Office 2000. With the release of VB .NET, this emphasis on a unified programming language has, for the moment at least, faded into the background, as the hosted version of Visual Basic continues to be VBA rather than VB .NET.
- Since Version 4, Visual Basic had increasingly been used as a kind of "glue language" to access COM components and their object models, such as ActiveX Data Objects (ADO), Collaborative Data Objects (CDO), or the Outlook object model. Although VB .NET supports COM for reasons of "backward compatibility," VB .NET is designed primarily to work with the .NET Framework rather than with COM.

With its language enhancements and its tight integration into the .NET Framework, Visual Basic is a thoroughly modernized language that will likely become the premier development tool for creating a wide range of .NET applications. In the past, Visual Basic was often seen as a "lightweight" language that could be used for particular kinds of tasks, but was wholly unsuitable for others. (It was often argued, sometimes incorrectly, that you couldn't create such things as Windows dynamic link libraries or shell extensions using Visual Basic.) In the .NET Framework, VB .NET emerges as an equal player; Microsoft's claim of language independence?that programming language should be a lifestyle choice, rather than a choice forced on the developer by the character of a project?is realized in the .NET platform.

This means that VB .NET can be used to create a wide range of applications and components, including the following:

- Windows console mode applications
- Standard Windows applications
- Windows services
- Windows controls and Windows control libraries
- Web (ASP.NET) applications
- Web services
- Web controls and web control libraries
- .NET classes and namespaces
- Accessing application object models (such as those of the individual applications in the Microsoft Office suite) using COM automation

Most importantly, for the first time with the release of VB .NET, Visual Basic becomes an all-purpose development environment for building Internet applications, an area in which it has traditionally been weak. This means that the release of this newest version should revitalize Visual Basic, allowing it to remain the tool of choice for developing state-of-the-art software for the next generation of software development.

Variables and Data Types

Many programmers take the concept of a variable for granted. In this chapter, we take a close look at variables and their properties, discussing such things as the scope and lifetime of a variable.

Variables

A variable can be defined as an entity that has the following six properties:

Name

A variable's name is used to identify the variable in code. In VB .NET, a variable name can start with a Unicode alphabetic character or an underscore, and can be followed by additional underscore characters or various Unicode characters, such as alphabetic, numeric, formatting, or combined characters.

Address

Every variable has an associated memory address, which is the location in memory at which the variable's value is stored. Note that in many circumstances, the address of a variable will change during its lifetime, so it would be dangerous to make any assumptions about this address.

Type

The type of a variable, also called its data type, determines the possible values that the variable can assume. We discuss data types in detail later

Value

The value of a variable is the contents of the memory location at the address of the variable. This is also sometimes referred to as the r-value of the variable, since it is what really appears on the right side of an assignment statement. For instance, in the code:

```
Dim i As Integer
Dim j As Integer
i = 5
j = i
```

the final statement can be read as "assign the value of i to memory at the address of j." For similar reasons, the address of a variable is sometimes called its l-value.

Scope

The scope of a variable determines where in a program that variable is visible to the code. Variables and constants can be declared with any of the following access modifiers

- `Public`
- `Private`
- `Friend`
- `Protected`
- `Protected Friend`

Scope is discussed in detail in Section

Lifetime

A variable's lifetime determines when and for how long a particular variable exists. It may or may not be visible (that is, be in scope) for that entire period. Lifetime is discussed in detail in Section

Data Types

The following lists the data types supported by VB .NET, along with their underlying .NET type, storage requirements, and range of values:

Boolean
Storage: 2 bytes
Value range: True or False

Byte
Storage: 1 byte
Value range: 0 to 255 (unsigned)

Char
Storage: 2 bytes
Value range: A character code from 0 to 65,535 (unsigned)

Date
Storage: 8 bytes
Value range: January 1, 1 CE to December 31, 9999

Decimal
Storage: 12 bytes
Value range: +/- 79,228,162,514,264,337,593,543,950,335 with no decimal point; +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/- 0.0000000000000000000000000001

Double (double-precision floating point)
Storage: 8 bytes
Value range: -1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values

Integer
Storage: 4 bytes
Value range: -2,147,483,648 to 2,147,483,647

Long (long integer)
Storage: 8 bytes
Value range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Object
Storage: 4 bytes
Value range: Any type can be stored in an Object variable.

Short
Storage: 2 bytes
Value range: -32,768 to 32,767

Single (single precision floating point)
Storage: 4 bytes
Value range: -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values

String (variable-length)
Storage: 10 bytes + (2 * string length)
Value range: 0 to approximately 2 billion Unicode characters

The Language Reference

This chapter documents VB.NET language elements. To help you speed the process of finding the right element to perform a particular task

Functions

The entry for each function provides the standard information that you'd expect for a function: its syntax, parameters (if it has any), return value, and description. In addition, we list rules for using the, frequently provide examples, and list related language elements.

For the first time, Visual Basic supports both named and positional arguments for all functions, procedures, and methods, with just a few exceptions. Functions, procedures, or methods that accept parameter arrays as arguments don't accept named arguments if the ParamArray parameter is present. And "functions" that are actually resolved by the compiler at compile time (the conversion functions fall into this category) do not accept named arguments. To see how named arguments work; let's look at the syntax of the Mid function:

```
Mid(Str As String, Start As Integer,
Length As Integer)
```

Using positional arguments, you might call the function as follows:

```
iPos = Mid(strName, 12, 10)
```

The same function call using named arguments might appear as follows:

```
iPos = Mid(start:=12, str:=strName,
length:=10)
```

Since named arguments are nearly universally accepted, we only note when you can't use named arguments with a particular function. The name of each argument is provided in the function's syntax statement. Finally, we've noted any differences between the operation of the function under previous versions of Visual Basic and under VB .NET.

Procedures

Procedures are really functions that don't return a value to the caller. Consequently, except for the absence of a return value, the same information is presented for procedures as for functions.

Statements

Visual Basic statements are not class members, don't support named arguments, and don't return a value. Aside from these three items, the same information is presented for statements as for procedures and functions.

Directives

Visual Basic directives are really statements that provide instructions to the VB .NET compiler or to a .NET development environment like Visual Studio. Like statements, they are not class members, don't support named arguments, and don't return a value. In general, the same information is presented for directives as for statements.

Branching

What is branching?

A program, as we learned, is a series of instructions to be carried out by a computer. In the programs we wrote, for a given event, the sequence of instructions were the same all the time. The data could be different, but the same instructions were carried out.

This is not always the case. Quite often, there are situations where the instructions sequence depends on the data. For example certain instructions will be carried out only for certain values of data. There may be an alternative set of instructions to be carried out for different values of data.

The selective execution of instructions is called branching. Of course, it is the programmer's job to specify the instructions should be carried out under various different situations.

The 'If...Then' Statement

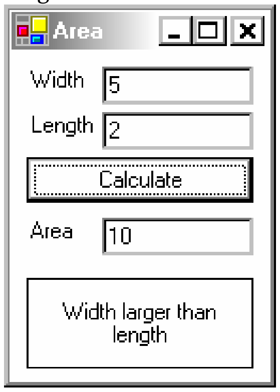
The simplest form of branching is the 'if...then' statement, where we restrict the executions of one instruction to the instances where a given **condition** is **true**.

The statement has the form

```
If condition then statement
```

The statement can be any visual basic command. The condition is generally a comparison of two quantities (e.g. x > 5). We will presently describe the general form of a condition, but before that let us look at an example.

Example: Improve program 1.1 so that a warning is displayed when the given width is more than the given length



```
  Private Sub cmdCalculate_Click(ByVal
➔sender As System.Object, ByVal e As
➔System.EventArgs) Handles
➔cmdCalculate.Click
    Dim w As Integer, a, l As Integer
    lblMsg.Text = ""
    w = txtWidth.Text
    l = txtlength.Text
    a = w * l
    txtArea.Text = a
    If w > l Then lblMsg.Text = "Width
➔larger than length"
  End Sub
```

Run the program, calculate area several times, giving width a mixture of values both less than and more than the length.
The warning is displayed only when width > length. The program goes on to display the area anyway, irrespective of whether the width is larger than length.

Given below is a slightly different method of writing the if....then structure, which allows us to subject more than one statement to the condition.

```
If condition then
   {statement}
End if
```

Thus the above program can be written as follows

```
  Private Sub cmdCalculate_Click(ByVal
➔sender As System.Object, ByVal e As
➔System.EventArgs) Handles
➔cmdCalculate.Click
    Dim w As Integer, a, l As Integer
    lblMsg.Text = ""
    w = txtWidth.Text
    l = txtlength.Text
    If w > l Then
      lblMsg.Text = "Width larger than
➔length"
    End If
    a = w * l
    txtArea.Text = a
  End Sub
```

The important aspect of the above structure definition is the curly brackets around the **statement**. They mean that more than one statement can be written there - that is more than one statement can be subjected to the condition. For example, when the width is more than the length, in addition to displaying the above label, we interchange them on screen.

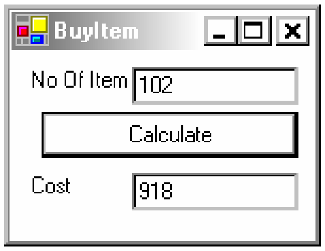
```
  Private Sub cmdCalculate_Click(ByVal
➔sender As System.Object, ByVal e As
➔System.EventArgs) Handles
➔cmdCalculate.Click
    Dim w As Integer, a, l As Integer
    lblMsg.Text = ""
    w = txtWidth.Text
    l = txtlength.Text
    If w > l Then
      lblMsg.Text = "Width larger than
➔length"
      txtWidth.Text = l
      txtLength.Text = w
    End If
    a = w * l
    txtArea.Text = a
  End Sub
```

The If...Then...Else Structure

The 'if...then...else' structure is used to select one course of action out of two alternatives.

```
If condition then
   {statement}
[else
   {statement}]
End if
```

Example: A certain item is sold at Rs 10 each, but if you buy more than 100, you can get a price of Rs 9 each. Write a program which calculates the amount when the quantity is given.



No Of Item: 102
Cost: 918

```
Private Sub cmdCalculate_Click(ByVal
➔sender As System.Object, ByVal e As
➔System.EventArgs) Handles
➔cmdCalculate.Click
    Dim i As Integer
    i = txtNoOfItem.Text
    If i > 100 Then
      txtCost.Text = i * 9
    Else
      txtCost.Text = i * 10
    End If
End Sub
```

Note the square brackets around the else part in the syntax rule. They mean that the else part is optional. That is you can write the structure without else part as

```
If condition then
   {statement}
End if
```

Selecting an Action from Many Alternatives

You can select one course of action from several alternatives with the following statement.

```
If condition then
   {statement}
Else If condition then
   {statement}
Else
   {statement}
End if
```

Note the big curly brackets around the Elseif part. That means that you can write as many elseif parts as necessary. For example, a structure with four alternatives might look like this

```
If condition then
   {statement}
Else If condition then
   {statement}
Else If condition then
   {statement}
Else If condition then
   {statement}
Else
   {statement}
End if
```

Here, the first set of statements will be carried out if the first condition is true. Otherwise, if the second condition is true the second set of statements will be carried out, and so on. If all three conditions are false, the last set of statements will be carried out.

Question: Enter a student's marks and display his grade on the following basis : 0 - 39 Fail, 40 - 59 Simple Pass, 60 -79 Credit Pass, 80 - 100 Distinction.

Note that once a true condition is met, the statements coming under it are carried out, and after that none of the other conditions are evaluated. In other words, a condition is checked only if all the proceeding conditions are false. Thus, in the above program, if m = 25 the grade "Fail" is assigned, and that's that. Although the value m = 25 satisfies the other conditions as well, they are not considered. This is structure is for selecting one alternative out of several.

Nested If Structures

You can put one structure inside another (if statement inside another If statement). Before dealing with the general case let's look at an example.

Question: Ceylon Electricity Board has three tariff structures for its customers.

Domestic customers	: As described in example2.4
Charities	: Rs 5 per unit
Insstitutions	: First 90 units Rs 8 per unit, Rs 12 per unit after that.

Exercise

1. A car rental company charges Rs 25 per km for jouneys lasting not more than 4 hours. For those with more than 4 hours the charge is Rs 20 per km plus Rs 50 per hour.
2. Write down the code for the following program to calculate the cost of a qty of fuel. When the program is started, the user can enter the unit price of all three fuels. Then he can enter various quantities of different fuels and find the cost.
3. In Sri Lanka, a person's first Rs 300,000 is free of tax. The next 300,000 is charged at 10%, the next 200,000 at 20%, the next 200,000 at 30% and the balance at 35%. Write a program to input the income and calculate the tax.
4. Display the roots of the equation ax2 + bx + c = 0 when the,a b and c values are entered.

Looping

What is looping?

So far, in our programs, each command was carried out once, for one execution of the procedure. However, in many tasks the same instruction or a set of instructions has to be carried out a number of times. The repeated execution of one or more lines of the program is called repetition, **looping** or iteration. The section that is carried out many times is called the **loop**. There are a number of ways to implement a loop.

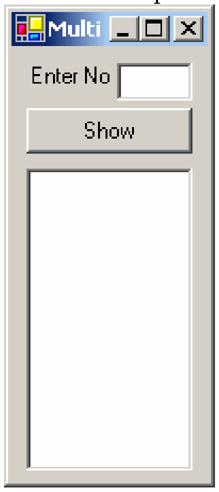
The For.. Next Loop

When the number of iterations is known before the loop starts, the **for..next** loop can be used. It has the form

```
For control-variable = start-value to end value
 {statement}
Next control-variable
```

Before explaining the general case, let us look at an example.

Example Given a number, show its multiples from 2 to 12.



```
  Private Sub cmdShow_Click(ByVal
➔sender As System.Object, ByVal e As
➔System.EventArgs) Handles
➔cmdShow.Click
    i = txtNo.Text
    For x = 2 To 12
      lst.Items.Add(i & " x " & x &
➔vbTab & " = " & i * x)
    Next
  End Sub
```

Again the loop is repeated with **x** taking values 2,3,4....12 respectively. Let us say that **i** is 5. Thus on the first iteration a is 2, thus **i * x** becomes 10. So the line 5 X 2 = 10 is printed.
In the second iteration **x** is 3. Then **i * x** becomes 15.
And so on.

For … Next Loop with step

Actually the full form of the **For..Next** structure is as follows.

```
For control-variable = start-value to end value [step increment]
 {statement}
Next control-variable
```

In this case the control variable is incremented by the given increment after each iteration. Thus the following program will show only the even multiples of the given number.

The Do Until Loop

Now we have seen quite a few examples involving the For...Next loop, let us look at a different looping command. The **For..Next** loop is useful when we know the number of repetitions before start the loop. Quite often the number of repetitions that will be required is not known when we start looping. Instead, we have to instruct the computer to repeat the loop until a certain condition is met. This is the statement.

```
Do Until condition
   {Statement}
Loop
```

In the **until** loop, the loop is carried out again and again until the given condition becomes true. For example if the loop is given as

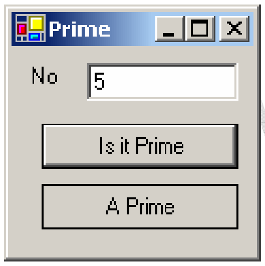
```
  Do Until a = b
     {Statement}
  Loop
```

The loop will be repeated until **a** becomes equal to **b**.

Example Given a number show whether it is a prime.
First let me present an algorithm.

1. Input the number to n.
2. Set x equal to 2.
3. Increment x until x exactly divided n.
4. If x equals n, display “prime”, otherwise “not a prime”

(Note that we stop once x exactly divides n. At this point, if x is less than n, that means n is divisible by some number between 1 and n, hence n is not a prime. But if x equals n, that means n was not divisible by any number between 1 and n, thus n is a prime)



```
  Private Sub cmdIsPrime_Click(ByVal
➔sender As System.Object, ByVal e As
➔System.EventArgs) Handles
➔cmdIsPrime.Click
    Dim x As Integer, n As Integer
    n = txtNo.Text
    x = 2
    Do Until n Mod x = 0
      x = x + 1
    Loop
    If x = n Then
      lblAns.Text = "A Prime"
    Else
      lblAns.Text = "Not a Prime"
    End If
  End Sub
```

The Do While Loop

Here the structure is

```
Do while condition
   {Statement}
Loop
```

In contrast to the **Do Until** loop, the **Do While** loop is carried out as long as the given condition is true. It stops when the given condition becomes false. Thus the loop **do while a = b** will be carried out as long as **a = b**. It will stop when a <> b.

Actually any loop that can be written as an '**until**' loop, can be writen as a **while** loop as well. You just have to invert the condition. Thus the loop **do until a = b** is equivalent to **do while a <> b**

Thus the above program can be written as follows. Here we have replaced **do until n mod x = 0** with **do while n mod x > 0**

Testing at the End of the Loop

Visual Basic allows you to put your condition at the end of the loop as well. The structures used are

Do
{Statement}
while condition
	Do
	{Statement}
	until condition

With this type of loop, the condition is first checked after the first iteration. Thus the loop will be carried out once no matter what. For example if you the loop starts with **do until a = b,** and **a** was equal to **b** to begin with, the loop will not be carried out even once. But if you put the condition at the end of the loop, the computer checks the condition after carrying out the loop once, so the loop will be carried out once anyway

Thus the above loops can be written as

x = 1
do
x = x + 1
while n mod x > 0
	x = 1
	do
	x = x + 1
	until n mod x = 0

Note that here we start with x = 1. Since the condition is at the end of the loop, if we start with 2, it will be 3 before the condition is tested, and if 2 is a factor it will be missed.

Nested Structures

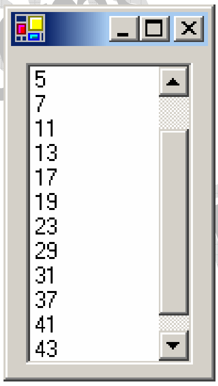
Now we know quite a few statement pairs like sub-endsub, if-endif, do-loop etc. A statement pair together with the enclosed statements is often called a structure. For example, an if-endif pair together with the enclosed statements is called an if structure. Two structures in a program must be completely outside each other, or one completely inside the other.

For example, if you have a if-endif pair and a do-while pair, in a program, it is possible to have one pair after the other, or one inside the other.

OK. - Two structures completely outside each other	OK. One structure completely inside the other.		WRONG. Two structures are partly overlapping.
if .... endif ... do while .... loop	. if ... do while .... loop ... endif	do while ... if ... endif ... loop	if ..... do while .... endif ... loop

When one structure goes inside another, the **inner** structure is said to be **nested** inside the **outer** structure. Given below are some examples of nesting.

Example Display the prime numbers from 1 to 50.
Here we have to put the do until loop of the above program inside a for-loop that runs from 1 to 50. The program display numbers directly on form.



```
5
7
11
13
17
19
23
29
31
37
41
43
```

```
  Private Sub frmPrimeRange_Load(ByVal
➔sender As System.Object, ByVal e As
➔System.EventArgs) Handles MyBase.Load
    Dim x, n As Integer
    For n = 2 To 50
      x = 2
      Do Until n Mod x = 0
        x = x + 1
      Loop
      If x = n Then lb.Items.Add(n)
    Next n
  End Sub
```

Exercises

1. Given the first term, the common difference and the number of terms, display the corresponding arithmetic progression.

2. Given the first term, the common ration and the number of terms, display the corresponding geometric progression.

3. Display odd numbers between 50 and 100.

4. Display the squares and cubes of numbers from 1 to 20.

5. Display the first 10 triangle numbers. (1, 3, 6,. 10, 15 ...)

6. In the Fabonacci series, the first two terms have to be given. Then each term is obtained by adding the two previous terms. For example, if the first two terms are given as 1,2, the series would be 1,2,3,5,8,13,21.... Given the first two terms, and the number of terms, display the series.

7. Given a number, show whether it is a square.

8. Given a number show whether it is a triangle number.

9. Resolve a number into prime factors as fallowing algorithm first.

```
i.   Input the number into n
ii.  Set x equal to 2
iii. While x does not exactly divide n, increment x.
iv.  Display x as a prime factor.
v.   Divide n by x.
vi.  If n is more than 1, go back to step 3.
```

The .NET Framework Library

There are more useful method and function in the vb.net framework. Please refer your own note book for full example

```
boolean << IsNumeric(< value read from textbox or variable >)
```

Usage Example

```
If IsNumeric(s) thern
```

Exercises

- Update the area calculation program of a rectangle with error message for wrong data

if you enter a numeric value to text box the answer will **true**

Convert Data Type Function

ToBoolean (<value>**)**
Converts a value to a Boolean

ToByte (<value>**)**
Converts a value to a Byte

ToChar (<value>**)**
Converts a value to a Char

ToDateTime (<value>**)**
Converts a value to DateTime (Date in Visual Basic)

ToDecimal (<value>**)**
Converts a value to Decimal

ToDouble (<value>**)**
Converts a value to Double

ToInt16 (<value>**)**
Converts a value to Int16 (Short in Visual Basic)

ToInt32 (<value>**)**
Converts a value to Int32 (Integer in Visual Basic)

ToInt64 (<value>**)**
Converts a value to Int64 (Long in Visual Basic)

ToSByte (<value>**)**
Converts a value to SByte, the unsigned-byte data type in the BCL

ToSingle (<value>**)**
Converts a value to Single

ToString (<value>**)**
Converts a value to String

ToUInt16 (<value>**)**
Converts a value to UInt16, an unsigned 16-bit integer

ToUInt32 (<value>**)**
Converts a value to UInt32, an unsigned 32-bit integer

ToUInt64 (<value>**)**
Converts a value to UInt64, an unsigned 64-bit integer

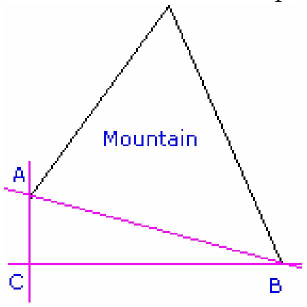
Usage Example
```
s = "false"
b = System.Convert.ToBoolean(s)
b = Convert.ToBoolean(s)
```

Mathematical Function

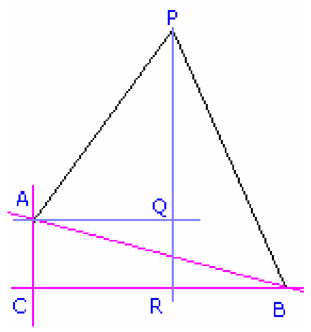
Abs
Absolute value
Acos
Arccosine
Asin
Arcsine
Atan
Arctangent; returns the angle whose tangent is a specified number
Atan2
Arctangent; returns the angle whose tangent is the quotient of two specified numbers
Ceiling
Returns the smallest integer greater than or equal to the argument number
Cos
Cosine
Cosh
Hyperbolic cosine
Floor
Returns the largest integer less than or equal to the argument number
Log
Natural (base e) logarithm
Log10
Common (base 10) logarithm
Max
Maximum
Min
Minimum
Pow
Generalized exponential function
Rnd
Returns a random number
Round
Rounds a given number to a specified number of decimal places
Sign
Determines the sign of a number
Sin
Sine
Sinh
Hyperbolic sine
Sqrt
Square root
Tan
Tangent
Tanh
Hyperbolic tangent

Exercises

- Write a program to find horizontal vertical and direct distance of a two separate point of a mountain



Write a program to calculate AC, BC and AB



We can find AP and BP distance and PBR and PAQ angles
We can input that data to the program and calculate horizontal vertical and direct distance

Mod operator
Returns the modulus, that is, the remainder when number1 is divided by number2

Pi field
Pi, the ratio of the circumference of a circle to its diameter

Randomize statement
Initializes the random number generator

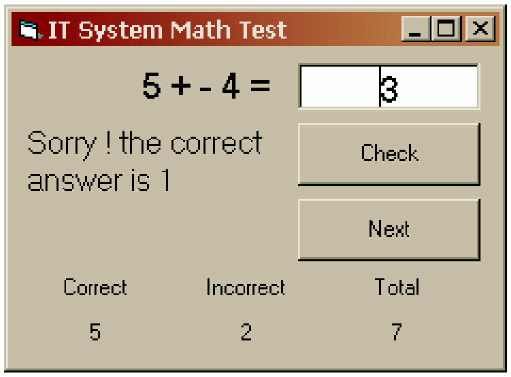
Random Numbers

Many programs, like computer games, simulations and educational programs requires random number. For example, if the program is dealing you a hand of cards, the cards must be selected at random, otherwise the game would be uninteresting. The function **Rnd** is used to obtain random number. The function itself returns an random number between 0 and 1, but it can be used to obtain random numbers of any range. Some examples are given below.

Expression	Random Number Range
Rnd	0 < x < 1
Rnd * 6	0 < x < 6
Int(Rnd *6)	0, 1, 2, 3, 4, 5
1 + Int(Rnd*6)	1,2,3,4,5,6
Int(Rnd*6)*10	0,10,20,30,40,50
Int(Rnd*6)*10+20	20,30,40,50,60,70

Exercises

Write program for testing the arithmetic ability of a student. When clicked the "Next Problem" button, it displays two integers in the range –9 to 9 at random. Also it displays one of the three operations +, - and * at random. Then the user has to type the answer and clcik "Ckeck answer". The program indicates whether the answer is correct, and, if not, what is the correct answer.



Sorry ! the correct answer is 1

Correct	Incorrect	Total
5	2	7

Rounding

Quite often when a result is obtained, or sometimes even when a datum is input, we have to round it. The main reasons for rounding are

1) Avoid practically infeasible values (e.g. fractions of a cent in an amount payable)

2) Avoid unwarranted accuracy (e.g. if you multiplied a number with six digit accuracy with another number with just two digit accuracy, only two digits in the result are meaningful))

Rounding can be done by just formatting if it is the final result and if it involves rounding to a fixed number of digits after the decimal point. You cannot use formatting to round something to the nearest 25 cents, say. Also if the result is an intermediate result which will be used for further calculations, then formatting is not the answer, as "hidden decimals" can cause inconsistent results. For example, if the unit price of something is calculated to be 12.503, and shown to be 12.50 by applying the format code "0.00", when you calculate the price of 10 of these the answer would be 125.03 rather than the expected 125.00.

And, depending on the occasion, we may have to round down, or round up, rather than round to the closest value.

String Handling Function

`Asc, AscW`
Returns an Integer representing the character code for the first character of the string passed to it. All other characters in the string are ignored.
`Chr, ChrW`
Returns the character represented by the character code.
`Filter`
Produces an array of matching values from an array of source values that either match or do not match a given filter string.
`Format`
Allows you to use either predefined or user-defined formats to create various ways to output string, numeric, and date/time data.
`FormatCurrency, FormatNumber, FormatPercent`
Used to format currency, numbers, and percentages.
`FormatDateTime`
Formats a date or time expression based on the computer's regional settings.
`GetChar`
Returns the Char that is at a given position index within a given string.
`InStr`
Finds the starting position of one string within another.
`InstrRev`
Determines the starting position of a substring within a string by searching from the end of the string to its beginning.
`Join`
Concatenates an array of values into a delimited string using a specified delimiter.
`LCase`
Converts a string to lowercase.
`Left`

Returns a string containing the leftmost length characters of string.

`Len`
Counts the number of characters within a string or the size of a given variable.

`LTrim`
The Me operator represents the current instance of a class from within the class module. (Since a form is a class, this includes forms

`Mid`
Returns a substring of a specified length from a given string.

`Replace`
Replaces a given number of instances of a specified substring in another string.

`Right`
Returns a string containing the rightmost length characters of string.

`RTrim`
Removes any trailing spaces from stringexp.

`Space`
Creates a string containing number spaces.

`Split`
Parses a single string containing delimited values into an array.

`StrComp`
Determines whether two strings are equal and, if not, which of the two strings has the greater value.

`StrConv`
Performs special conversions on a string.

`StrDup`
Returns a string that consists of the first character of string duplicated a number of times.

`StrReverse`
Returns a string that is the reverse of the string passed to it. For example, if the string "and" is passed to it as an argument, StrReverse returns the string "dna."

`Trim`
Removes both leading and trailing spaces from a given string.

`UCase`
Converts a string to uppercase.

Like operator
If string matches pattern, results in True; otherwise, results in False.

Mid statement
Replaces section of a string with characters from another string.

Exercises

1. Write a program to change name to upper case
2. A distance is given in feet and inches in ff:ii format. Covert it into inches.
3. Update first program to change only last name to upper case
4. Write a program to input the start and finish times of a telephone call in hh:mm:ss format, and find the duration of the call in seconds.
5. Write a program to separate the user name and the server name in an email address. E.g. pqr@yahoo.com should be separated to “pqr” and “yahoo.com”
6. Write a program to input a binary number as a string, and convert it to decimal. Eg. 101 to be 5
7. Write a program to convert a decimal number to binary.
8. Write a program to input a string of the form “number operator number” where the operator is “+”, “-”, “*” or “/”, and display the result of simplifying it.
9. Write a program to display date of birth and sex when the NIC No is given.
10. Write a program to display the first five digits of the NIC No when the date of birth and sex are given