Topic 2 – The Basics of VB Console Applications

2.1 Introduction

In Topic 1, you were introduced to the Visual BASIC programming environment and followed through the stages of the software development process to analyse, design, implement, test and evaluate your first VB program.

Now it is time to learn some of the basics of VB programming.

2.2 What you should already know

This topic assumes that you have already used and understood the following VB constructs:

- Console.Write()
- Console.ReadLine()

2.3 Learning Outcomes

After completing this topic, you should be able to:

- Design, implement and test a simple VB program
- Save and print a VB program
- Input data using Console.ReadLine() or InputBox
- Display messages and data using Console.Write() or MsgBox
- Explain the importance of declaring variables using Dim
- Distinguish between real (single), integer, string and Boolean variables
- Carry out systematic and comprehensive testing
- Evaluate software in terms of effectiveness, usability and maintainability
- Format output using the format function

2.4 Output to a Message Box

Let's have another look at the 'Welcome to VB' program from Topic 1:

```
Sub Main()
   'code for input – output program
   'by A. Programmer on 04/01/10

   Dim name As String

   Console.Write("Enter your name")
   name = Console.ReadLine()
   Console.Write("welcome to VB," & name)

   Console.ReadLine()
End Sub
```

In the 'Welcome to VB' program, the output message appeared as text in the console window.

It is also possible to make the output message appear in its own window, called a message box.

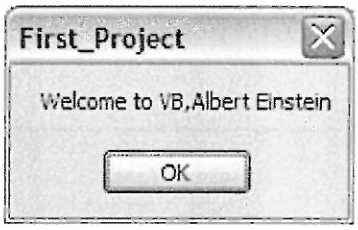
The line of code to do this is: **MsgBox("welcome to VB, " & name)**

Load up your previous program.

Change the line of code from: ***Console.Write("welcome to VB," & name)*** to ***MsgBox("welcome to VB, " & name)***

Run the program again.

The result should look something like this:



Choosing the style of output

You have now seen 2 ways of outputting the same message:

- As text in the console window
- In a message box

Visual BASIC has several other forms of output, which you will learn about later.

The form of output may be stated in the program specification.
If not, it is a decision that you can make at the **design** stage.

2.5 Using an Input Box

In the 'Welcome to VB' program, the name was input by typing into the console window.

An alternative is to use a pop-up **Input Box**.

Here is the latest version of the 'Welcome to VB' program again:

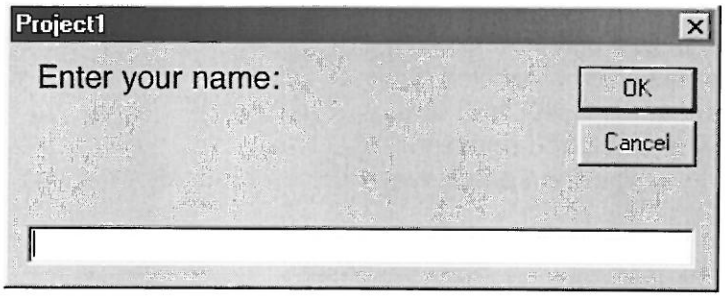
```
Sub Main()
    'code for input – output program
    'by A. Programmer on 04/01/10

    Dim name As String

    Console.Write("Enter your name")
    name = Console.ReadLine()
    MsgBox("welcome to VB," & name)

    Console.ReadLine()
End Sub
```

To change the program to use an input box:

- Delete the lines of code ***Console.Write("Enter your name")*** *and* ***name = Console.ReadLine()***
- Replace them with the line: ***name = InputBox("Enter your name")***

Run the program again.

You should find that an input box pops up asking you to enter your name. When you click OK, it disappears and a Message box pops up with the correct welcome message.



2.6 Summary of Input and Output Techniques

Input techniques	Output techniques
Type text in console window	Text displayed in console window
Pop-up Input Box	Pop-up Message Box

For any program you develop, you can choose which of the two input techniques, and which of the two output techniques, to use. Visual BASIC also has many other methods of input and output, but those above should be enough for all your requirements at present.

2.7 Data Types

One of the most important aspects to consider during the design stage of software development is the type of data that will be processed by the program.

It is **always good practice** when using any programming language to **declare the names and types** of all variables that are being used.

The reason for this is that computers store different types of data in different ways.

Some earlier versions of VB allowed programmers to avoid declaring variables. Any new variable would simply be assigned to a type called 'variant'. This was not good practice, for several reasons:

- It made the program less readable, and therefore less maintainable
- It encouraged sloppy attitudes to the software development process
- It made programs less efficient

We will consider 4 types of data in this course.

Look at these items of data. Can you group them into 4 types?

120	*computing*	*true*	*699*
book	*29.5*	*Int2*	*5.7*
A.Einstein	*-100*	*TD7*	*false*
5700	*dog*	*Monaco*	*0.006*
9999	*1357.9075*		

You might have grouped them into these 4 lists:

List 1:	120	699	-100	5700	9999	
List 2:	computing	book	A.Einstein	TD7	dog	Monaco
List 3:	29.5	5.7	0.006	1357.9075		
List 4:	true	false				

List 1 is made up of whole numbers – we called them **integers** – so does VB

List 2 is made up of groups of characters – VB calls them **strings**

List 3 is made up of numbers with a decimal point – we call these **real numbers** – in VB we declare these as type **single**

List 4 is made up of the words true and false – these can be stored using a **Boolean** variable

Classify each of the following as integer, string, single, or Boolean:

(a) 150 (b) Bob the Builder

(c) 49.99 (d) true

(e) 0.5 (f) – 500

(g) false (h) 123

(i) Albert (j) -99.99

(k) 0.00006 (l) 5 High Street

(m) 3B4 (n) 23.5×10^6

Declaring variables and program efficiency

To illustrate how declaring variables of the right type can improve efficiency, look at this table which shows the amount of storage allocated by the VB environment to the 4 data types, and to the 'variant' data type:

Data type	Example	Range	Storage allocated
Single	235.35	-3×10^{38} to 3.4×10^{38}	4 bytes
Integer	235	-32,768 to +32,767	2 bytes
String	two	any characters	1 byte per character
Boolean	true	true or false	2 bytes
Variant	anything		minimum 16 bytes

Note: VB also has other data types, including long integer (4 bytes), double precision real (8 bytes), date (8 bytes) and currency (8 bytes).

Using the correct variable type should reduce the amount of RAM required by the computer when running the program. This will also reduce the time required to transfer data between processor and memory, so it should improve efficiency.

Of course, this is not significant for small programs on computers with fast processors and plenty of RAM. However, it can be important when developing large-scale commercial software.

Option Explicit

The default setting in VB 2005 or 2008 EE is that all variables must be declared. To check that this is the case, you should:

- Select the Tools menu
- Select Options
- Select Projects and Solutions, VB Defaults
- Check that Option Explicit is On

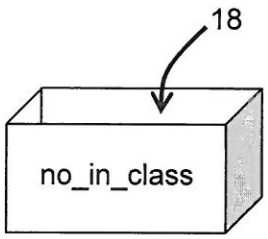
2.8 Declaring variables in Visual BASIC

Visual BASIC needs to know what type of data it will be storing and processing in any program. To do this we 'declare variables' at the start of the program, using lines like:

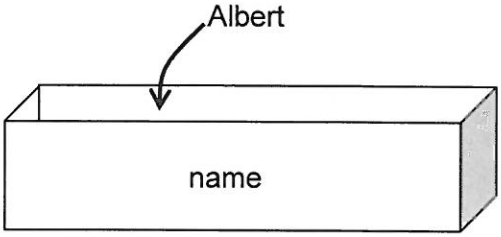
```
Dim no_in_class As Integer

Dim name As String

Dim price As Single

Dim valid_check As Boolean
```

When the VB system 'reads' these statements at the start of a program, it sets up a storage space of the appropriate type in the computer's RAM, and labels it with the variable name given. Of course, these are 'electronic' storage locations, but it is useful to imagine them as labelled boxes in which data can be stored, like this:

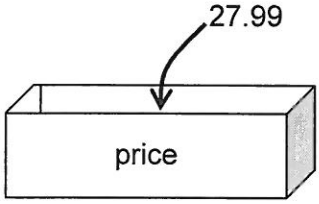
An **integer** variable, called **no_in_class**, storing value 18



A **string** variable, called **name**, storing the word 'Albert'



A **single** variable, called **price**, storing value 27.99



These 'storage boxes' are called **variables**, because the actual value of the data they store can vary or change during the running of a program.

It is important to make sure that all variables are correctly declared – the right **type** (integer, string, single or Boolean) – and with sensible, readable **variable names**.

Variables can have almost any name, but each variable name:

- Must begin with a letter
- Must not be a VB keyword (like End or Print or MsgBox)
- Must not contain spaces (no_of_pupils is OK, but no of pupils is not)

2.9 Developing a calculation program using VB

We can now use what we have learned to design and implement a more complex program.

Here is what the program has to do:

An athletics team manager wants a program which will input a runner's name, event, squad number and time taken in the first three races of the season. It should then calculate the runner's average time (to the nearest whole number of seconds) and display a summary of their season so far.

Stage 1 – Analysis: Program Specification

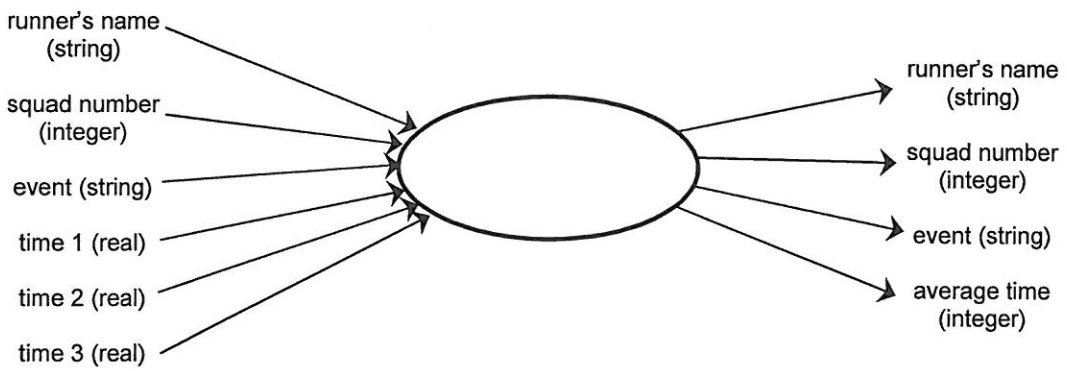
The first stage is to convert this description into a more formal **program specification**. This clarifies all the inputs, outputs and processes. It could look something like this:

Design, write and test a program to:

- Prompt the user to enter a runner's name, event, squad number and time taken in races 1, 2 and 3
- Calculate the runner's average time, rounded to the nearest whole number
- Clearly display the runner's name, event, squad number and average on the screen

Stage 1 – Analysis: Data Flow Diagram

Next, we need to show clearly the flow of data in and out of the program, using a data flow diagram. At this stage it is useful to think about the data types (integer, real or string).



Reminder:

- **Integers** must be whole numbers
- **Reals** can include fractional parts
- **Strings** are for words, letters and symbols

Stage 2 – Design: User Interface

In Visual BASIC, there are a number of ways of inputting and outputting data. For this program we will use simple text in the console window.

Stage 2 – Design: Pseudocode

When the user runs the program, we want:

- The user to be prompted to input each of the six items of data
- The average should then be calculated
- Finally the output should be displayed

In pseudocode:

1. Prompt for and store the runner's name
2. Prompt for and store the runner's event
3. Prompt for and store the runner's number
4. Prompt for and store each of the three times
5. Calculate the average time (rounded to the nearest whole number)
6. Display the runner's name, event and number
7. Display the runner's average time

Stage 3 – Implementation

- Start up Visual BASIC on your computer
- Choose File / New Project, and select Console Application
- Name the project 'Runner'

First, think about the variables that will be needed to store the six items of data. Each variable must have a name and a type:

runners_name will be a **string** variable (e.g. Bill McSporran)
event will be a **string** (e.g. 100m sprint)
squad_number will be an **integer** as it will be a whole number (e.g. 73)
time_1, time_2 and **time_3** will be reals **(singles in VB)** (e.g. 11.35, 12.01 and 12.99)
average will be an **integer** as it is to be rounded to the nearest whole number (e.g. 12)

In the code window, in Sub Main(), declare all of these variables using Dim statements:

```
Sub Main ()
    'code for the runner program
    'by A. Programmer on 05/01/10

    Dim runners_name, which_event, squad_number As String
    Dim time_1, time_2, time_3 As Single
    Dim average As Integer
```

*Note: **event** is a **reserved word** in VB, so we need to call this variable **which_event***

For the next section of code, translate each line of pseudocode into Visual BASIC:

```
        Console.Write("Enter the runner's name ... ")
        runners_number = Console.ReadLine()

        Console.Write("Enter the runner's number ... ")
        squad_number = Console.ReadLine()

        Console.Write("Which event? ")
        which_event = Console.ReadLine()

        Console.Write("Time in first race ... ")
        time_1 = Console.ReadLine()

        Console.Write("Time in 2nd race ... ")
        time_2 = Console.ReadLine()

        Console.Write("Time in 3rd race ... ")
        time_3 = Console.ReadLine()

        average = CINT((time_1 + time_2 + time_3) / 3)

        Console.WriteLine()

        Console.WriteLine("Name " & runners_name)
        Console.WriteLine("Number " & squad_number)
        Console.WriteLine("Event " & which_event)
        Console.WriteLine("Average time " & average & "s")

        Console.ReadLine()
    End Sub
```

Note: *The* ***CINT*** *function rounds the answer to the nearest whole number*

Note: *Using* ***WriteLine*** *ensures that a new line is taken for each output message*

Note:

The calculation is carried out by the single line of code:

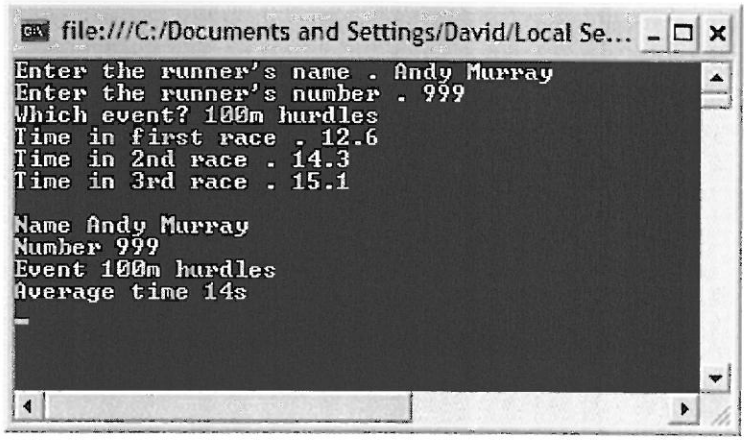
average = CINT((time_1 + time_2 + time_3) / 3)

This means:

- Add together the values of the variables time_1, time_2 and time_3
- Divide them by 3 (in programming we use / to mean divide by)
- Use the CINT function to round the answer to the nearest whole number
- Store the answer in the variable average

Run the program.

The results should look something like this:



Stage 4 – Testing

Run the program a few times with different data.

Do your own calculations to ensure that the program is calculating answers correctly.

If there are any errors, correct them, and retest.

Saving your Program

Choose **Save all** from the **File** menu.

Printing your Program

Select **Print** from the **File** menu to print your program coding.

Stage 5 – Evaluation

The evaluation of your program should answer the following questions:

1. Is the program **effective**? (Does it do what is required by the specification?)
2. Is the program **usable**? (Is it clear what the user has to do?)
3. Is the program **maintainable**? (Could another programmer understand how it works?)

Your evaluation report might look like this:

> Evaluation
>
> The program is effective. It fulfils the specification. It calculates the average correctly for any sensible inputs.
>
> The user interface is easy to use – it prompts for input, and the output is clearly displayed.
>
> The coding has comment lines and uses sensible variable names to make it maintainable by another programmer.

Evaluation is a slightly artificial activity for small programs such as the ones we will be developing in this course, but it is a very important part of the software development process for real, large-scale, commercial software.

2.10 Formatting output

In the previous example, the specification stated that the runner's average time should be displayed to the nearest whole number, so we declared the variable average to be of type integer, and used the **CInt** function to round the answer to the nearest whole number.

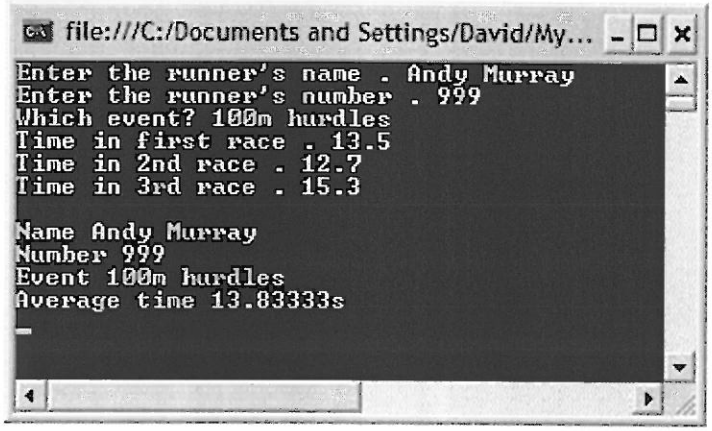
On seeing the final program, the team manager decides that it would be better to display the average more precisely. The program requires some **perfective maintenance** (this is the technical term for adding new features to a program).

There are two other types of maintenance – called **adaptive maintenance** and **corrective maintenance** – but you don't need to know about these at this stage.

Load the 'runner' program and make the following changes to allow it to display the runner's average time more precisely:

- Change the variable average to be a single (real) instead of an integer
- Remove the **CInt** function
- Save the changes you have made
- Run the program

You probably ended up with an output something like this:



The team manager is still not pleased.
The displayed answer is now too precise! He wants it rounded to two decimal places.

To do this, we need to use the **format** function.

Change the line: **Console.WriteLine("Average time " & average & "s")**
to: **Console.WriteLine("Average time " & Format(average, ".00") & "s")**

You should now find that whatever answer is produced by the program, it is displayed to two decimal places.

Try the following variations, and try to explain the format each one produces:

Format(average, ".000")
Format(average, "000.0")
Format(average, "###.0")
Format(average, "fixed")
Format(average, "currency")
Format(average, "percent")
Format(average, ".00\second")
Format(average, "\{00.00\}")

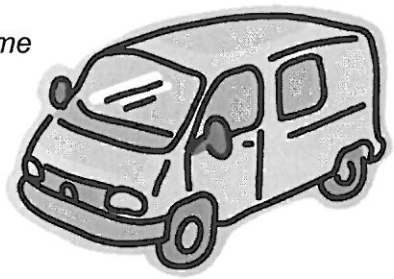
2.11 Other data types

In this section, we have used integer, string, single and Boolean data types. Some programming languages allow a range of other types, including:

- Byte (stores a single byte of data)
- Character (stores a single character)
- Date/time (stores a date or time in a standard coded form)

2.12 Practical Task – The Basics of VB

To check that you have fully understood Topic 2, and are ready to continue with this VB course, try this example software development task.



A delivery driver wants a simple program to help him calculate the time it will take him to make various journeys. The program should prompt him to input the starting place and destination of a journey, the distance he will have to travel, and his estimated average speed. The program will use the formula:

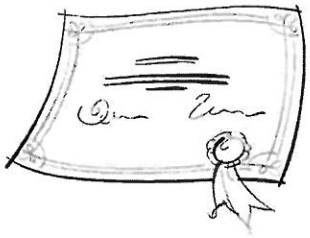
time_taken = distance / average_speed

to calculate the journey time.

The program should display the start, destination, distance, average speed and time (to one decimal place).

You should follow the stages of the software development process as illustrated in the runner example in Topic 2.9.

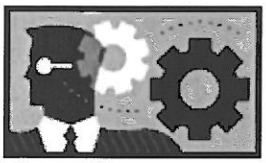
Topic 2 Summary



Well done! You have completed Topic 2. In this topic, you have learned (or revised) how to:

- ☑ Create a simple VB program
- ☑ Save and print a VB program
- ☑ Input data using Console.ReadLine() or InputBox
- ☑ Display messages and data using Console.Write() or MsgBox
- ☑ Declare variables (real, string and integer) to store data
- ☑ Evaluate software in terms of effectiveness, usability and maintainability
- ☑ Format data using the format function

Check all the items on this list. If you are not sure, look back through this section to remind yourself.



When you are sure you understand all of these items you are ready to continue with the next topic.

