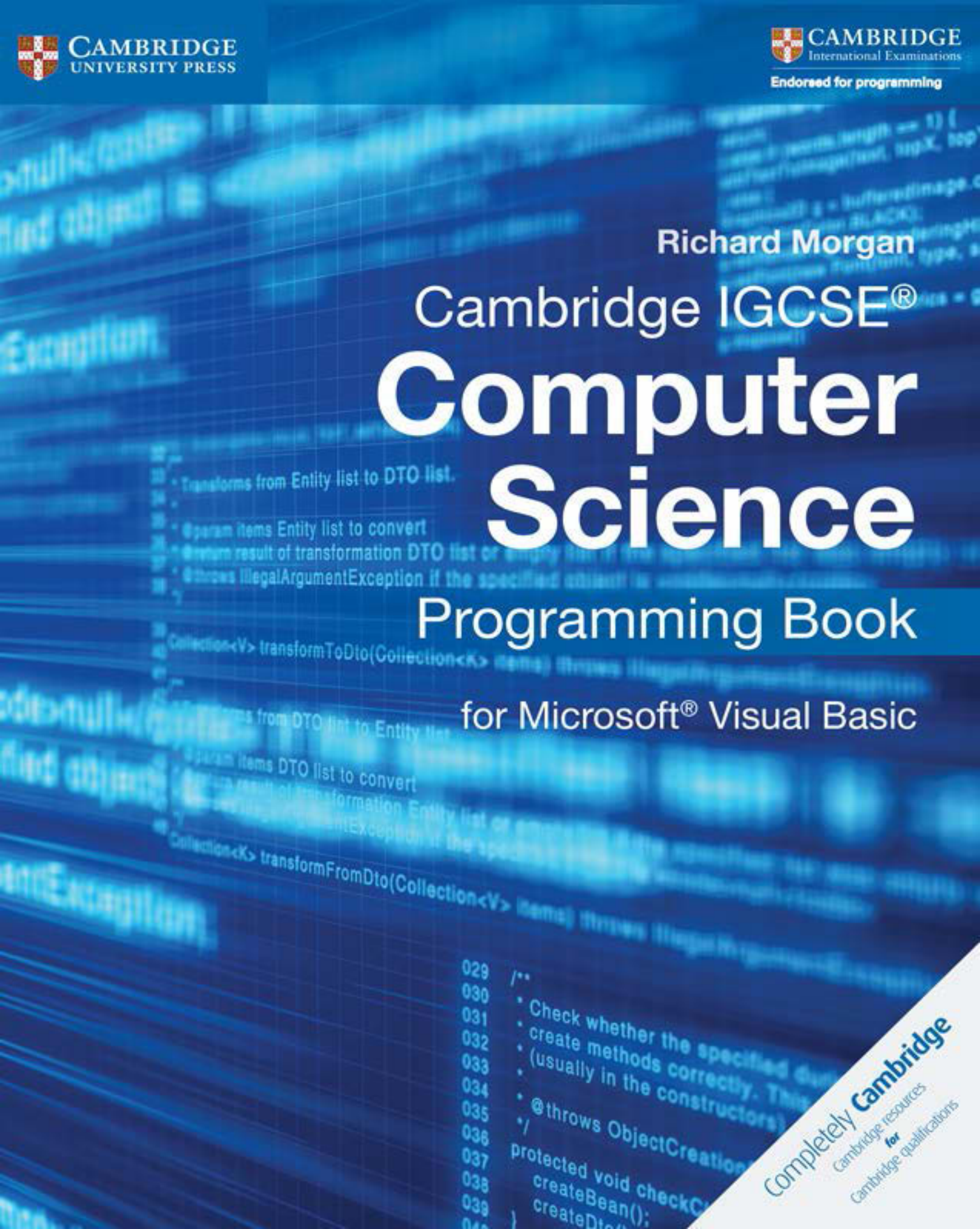
CAMBRIDGE UNIVERSITY PRESS

CAMBRIDGE International Examinations
Endorsed for programming

Richard Morgan

Cambridge IGCSE® Computer Science Programming Book

for Microsoft® Visual Basic



Richard Morgan

Cambridge IGCSE®
Computer Science

Programming Book

for Microsoft® Visual Basic

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning and research at the highest international levels of excellence.

Information on this title: education.cambridge.org

First published 2015

Printed in the United Kingdom by Latimer Trend

A catalogue record for this publication is available from the British Library

ISBN 978-1-107-51864-3 Paperback

Contents

Introduction

When I wrote this book I had two aims in mind. The first was to provide a programming book that specifically covered the material relevant to the Cambridge IGCSE® syllabus. The second, and perhaps more important, aim was to provide the student with a start to the exciting and rewarding process of being able to create their own computer programs.

Language

The syntax and structures used to implement programming techniques will vary across different languages. The book is entirely based around Visual Basic, one of the three recommended languages for the A Level syllabus. Visual Basic offers the student, as a programmer, two modes of application. There is a simple console window in which the student can learn and develop programming skills. It also offers a Windows Forms application, which allows the student to program commercial-style applications that offer a graphical user interface through which users can interact with programs.

The language is supported by a fully functional development environment called Visual Studio Express, which is available free directly from Microsoft. They also provide excellent support and language-specific tutorials via the Microsoft Developer Network. All the code and language specific comments in this book relate to Visual Studio Express 2013.

Examination focussed

The course will test computational thinking independent of any specific programming language. It will do this through the use of program design tools such as structure diagrams and flowcharts. It will also make use of pseudocode, a structured method for describing the logic of computer programs.

It is crucial that the student becomes familiar with these techniques. Throughout this book all the programming techniques are demonstrated in the non-language-specific format required. This will help prepare the student to answer the types of question they will meet in their studies.

To support learning, many of the chapters include examination-style tasks. Chapter 13 has examples of appropriate code solutions showing how to turn logical ideas into actual programs. There is also a series of examination-style questions in Chapter 12, which has a sample mark scheme giving possible solutions and showing where the marks might be awarded.

Developing programming skills

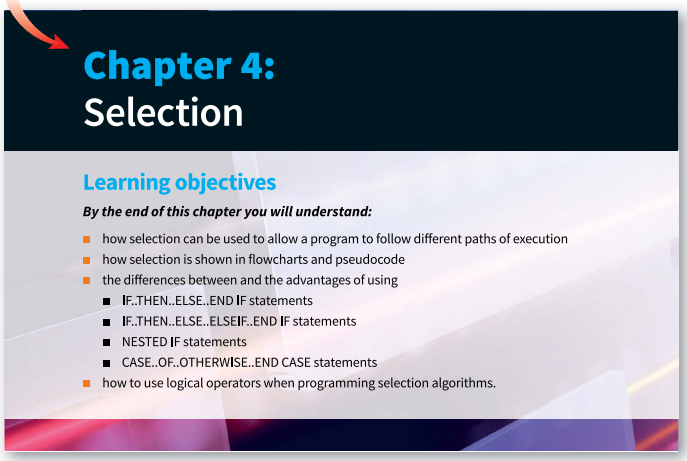
One of the advantages of Visual Basic is that it provides a language that encourages the student to program solutions making use of the basic programming constructs: sequence, selection and iteration. Although the language does have access to many powerful pre-written code libraries, they are not generally used in this book.

Computational thinking is the ability to resolve a problem into its constituent parts and to provide a logical and efficient coded solution. Experience tells me that knowing how to think computationally relies much more on an understanding of the underlying programming concepts than on the ability to learn a few shortcut library routines.

This book is aimed at teaching those underlying skills which can be applied to the languages of the future. It is without doubt that programming languages will develop over the coming years but the ability to think computationally will remain a constant.

How to use this book: a guided tour

Chapter – each chapter begins with a short list of the facts and concepts that are explained in it.



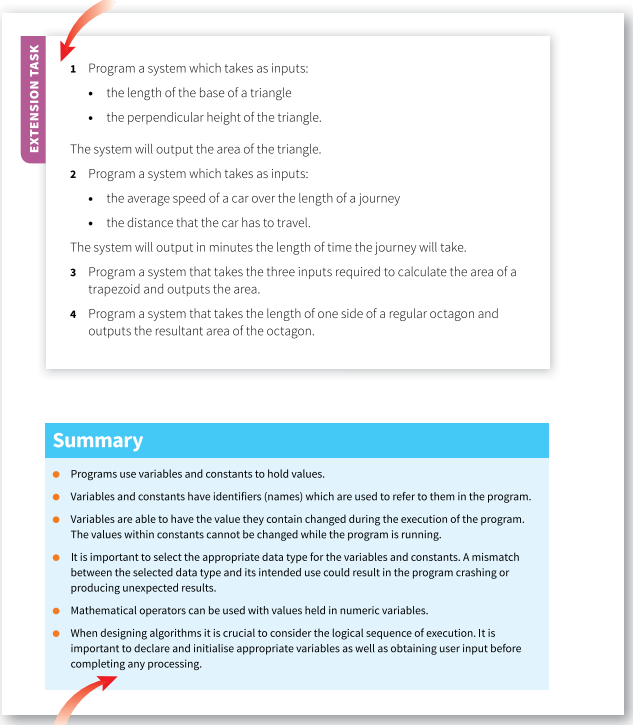
Chapter 4: Selection

Learning objectives

By the end of this chapter you will understand:

- how selection can be used to allow a program to follow different paths of execution
- how selection is shown in flowcharts and pseudocode
- the differences between and the advantages of using
  - IF..THEN..ELSE..END IF statements
  - IF..THEN..ELSE..ELSEIF..END IF statements
  - NESTED IF statements
  - CASE..OF..OTHERWISE..END CASE statements
- how to use logical operators when programming selection algorithms.

Extension Task – extension of an existing exercise for the student to further develop their knowledge and understanding.



EXTENSION TASK

1 Program a system which takes as inputs:
- the length of the base of a triangle
- the perpendicular height of the triangle.

The system will output the area of the triangle.

2 Program a system which takes as inputs:
- the average speed of a car over the length of a journey
- the distance that the car has to travel.

The system will output in minutes the length of time the journey will take.

3 Program a system that takes the three inputs required to calculate the area of a trapezoid and outputs the area.

4 Program a system that takes the length of one side of a regular octagon and outputs the resultant area of the octagon.

Summary

- Programs use variables and constants to hold values.
- Variables and constants have identifiers (names) which are used to refer to them in the program.
- Variables are able to have the value they contain changed during the execution of the program. The values within constants cannot be changed while the program is running.
- It is important to select the appropriate data type for the variables and constants. A mismatch between the selected data type and its intended use could result in the program crashing or producing unexpected results.
- Mathematical operators can be used with values held in numeric variables.
- When designing algorithms it is crucial to consider the logical sequence of execution. It is important to declare and initialise appropriate variables as well as obtaining user input before completing any processing.

Summary Checklist – at the end of each chapter to review what the student has learned.

When writing a FOR loop in Visual Basic you need to follow this format:

```
For i = 1 To 10
    'Code to execute
Next
```

Each individual element of the loop performs an important role in achieving the iteration as shown in Table 5.2.

Table 5.2

Element	Description
`For`	The start of the loop
`i = 1 To 10`	`i` is a counter variable that records the number of iterations that have been run. This is usually incremented by 1 every iteration. In Visual Basic there is no requirement to declare the counter variable separately – it is automatically declared as part of the FOR loop. The value of the counter variable can be used within the loop to perform incremental calculations.
`Next`	The end of the iteration section The value of the counter variable is incremented and the flow of the program goes back to the `For`. The loop will evaluate if the counter value is within the condition (10 in this example). If the counter has exceeded the end value, the loop will direct the flow of the program to the line of code following `Next`; if not it will rerun the loop.

Any code that is placed within the FOR loop will be repeated on each iteration. The repeated code can itself include complex processes such as selection or additional loops.

KEY TERM

FOR loop: A type of iteration that will repeat a section of code a known number of times.

TIP

As the conditions are checked at `For`, `Next` will always pass execution of the loop back to `For` to check the conditions. It is a common misconception that once the maximum number of iterations has been reached `Next` will exit the loop. This is not true. Consider a situation where a FOR loop is written to execute 10 times. Although the loop counter may have reached 10 `Next` will still increment to counter to 11 before passing execution to `For`. The value of the loop counter will be outside the criteria and `For` will then exit the loop.

A system is required to output the multiples of a given number up to a maximum of 10 multiples. For example the multiples of 6 are 6, 12, 18, 24, 30, 36, 42, 48, 54 and 60. Figure 5.1 shows the flowchart and pseudocode for the design of the algorithm. Although the counter is automatically declared in Visual Basic this is not the case with all languages so it is normal to include the declaration in the design.

SYLLABUS CHECK

Pseudocode: understand and use pseudocode for counting (e.g. Count ← Count + 1).

Key Term – clear and straightforward explanations of the most important terms in each chapter.

Tip – quick suggestions to remind the student about key facts and highlight important points.

Syllabus Check – links programming concepts explained in the text to the Cambridge IGCSE syllabus.

TASK

FOR Loop

1 Extend the multiply system to include two inputs. The first input is the number to multiply, the second is the number of multiples required.

2 Produce a system that accepts two numbers A and B and outputs A^B. For example if A = 3 and B = 4, the output will be 81 ($A^4 = A \times A \times A \times A$).

Task – exercises for the student to test their knowledge and understanding.

Acknowledgements

The authors and publishers acknowledge the following sources of copyright material and are grateful for the permissions granted.

Cover Soulart/Shutterstock; p. 1 isak55/Shutterstock; p. 13 aimy27feb/Shutterstock; p. 19 Image Source/Getty Images; p. 31 Magictorch/Ikon Images/Getty Images; p. 51 alexaldo/iStock/Getty Images; p. 67 Ioana Davies (Drutu)/Shutterstock; p. 77 Devrimb/iStock/Getty Images; p. 85 Mclek/Shutterstock; p. 93 Kutay Tanir/Photodisc/Getty Images; p. 103 ILeysen/Shutterstock; p. 113 Kamil Krawczyk/E+/Getty Images; p. 114 John Howard/Science Photo Library

Screenshots of Microsoft Visual Studio Express 2013 for Windows used with permission from Microsoft.

Chapter 1:
Visual Studio Express

Learning objectives

By the end of this chapter you will understand:

- the two programming applications used in this book
- how to code and save a basic program in Console Application
- how to obtain input data and provide output in Console Application
- how to code and save a basic program in Windows Forms Application
- how to use the main programming windows in Windows Forms Application
- the format of the event-driven subroutines used in a Windows Forms Application.

1.01 Getting Visual Studio Express 2013 for Windows

Visual Studio Express 2013 is the current version of free developer tools provided by Microsoft. They include the programming languages Visual Basic, Visual C++ and Visual C#. The examples in this book have been produced using Visual Studio Express for Windows.

I have used Visual Basic with both GCSE and A Level students for the last five years as it provides an interface that allows students to develop programming skills while at the same time producing satisfying systems. Visual Basic also provides programmers with access to a large class library. Classes are templates that hold prewritten code that support functionality of objects. As students' skills increase they are able to use this feature-rich development environment to produce and publish complex systems. System requirements and download options can be found at www.visualstudio.com/products/visual-studio-express-vs.

1.02 The Integrated Development Environment (IDE)

The default start page for Visual Studio Express 2013 is shown in Figure 1.01. It consists of a number of connected windows which offer different functionality based on the type of project that you open. To begin select New Project.

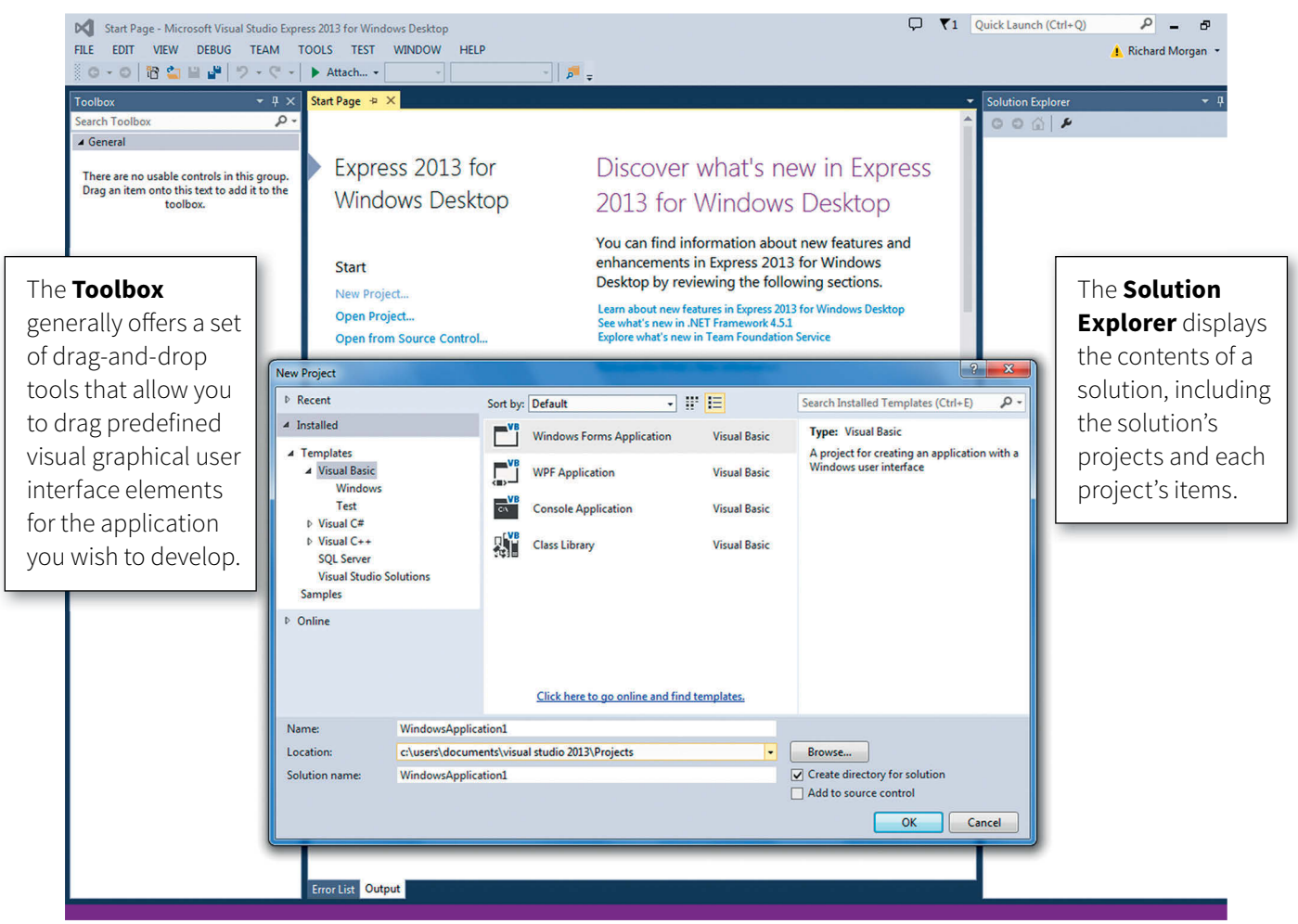


Figure 1.01 Visual Studio Express start window

The New Project window provides you with a choice of types of application. The two that are used in this book are Console Applications and Windows Forms Applications. Both make use of very similar coding approaches to algorithms but differ in the way that a user interfaces with them.

A Console Application provides a textual interface uncluttered by the need to support a graphical user interface (GUI). Although IGCSE does not stipulate the use of any programming language or mode the Console Application is the required format in the A Level syllabus.

A Windows Forms Application provides a graphical user interface that can be customised to provide users with visual input, output and processing options. The flow of the program is largely controlled by routines that are triggered in response to the user interacting with the interface.

To create a new project select the preferred application type (in this case, Console Application), change the default name to something meaningful and select OK.

1.03 Console Application

The default layout of the console mode consists of the main **programming window** (see Figure 1.02) which provides an area in which you write the program code required to accept inputs, process data and produce the required outputs. The **Solution Explorer** displays the contents of a solution, which includes the solution's projects and each project's items. This is where you will find the program you have written, listed as a .vb file.

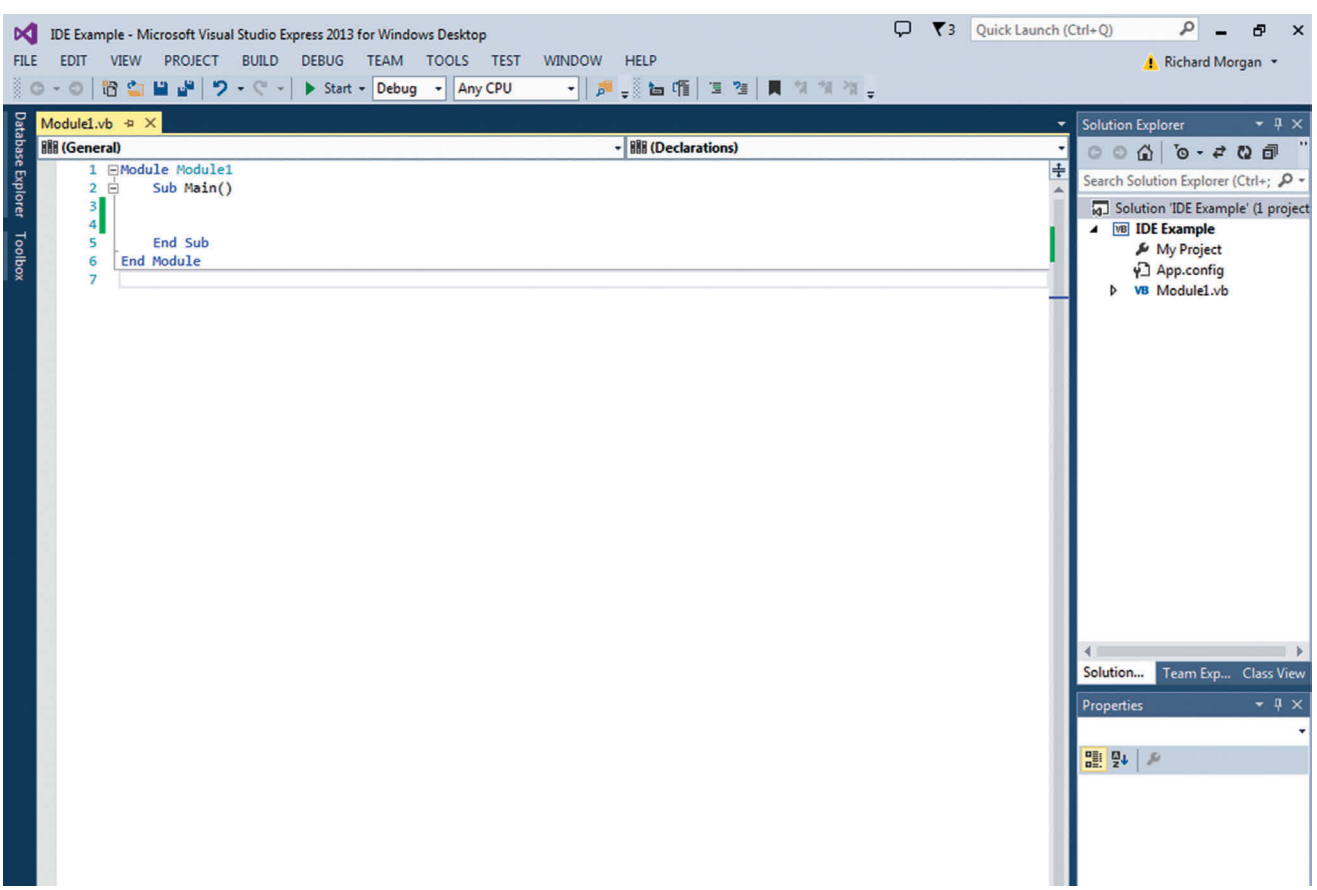


Figure 1.02 Console application programming window

1.04 Make Your First Program using Console Mode

When the console mode is first loaded the code window will contain four lines of code.

```
Module Module1

    Sub Main()
    End Sub

End Module
```

A module is a container of code and can hold a number of subroutines that perform specific actions. `Sub Main()` is the entry point for the program and `End Sub` indicates the end of the subroutine. Code written between these points will be executed when the application is run. You can use the Enter key to add additional lines.

To produce the text 'Hello World' you need to code the application to display the required text.

In Visual Basic the functionality of a class is accessed by use of the dot symbol. Reading inputs and displaying outputs makes use of the Console class which provides access to a library of methods that allow the user to interact with the console.

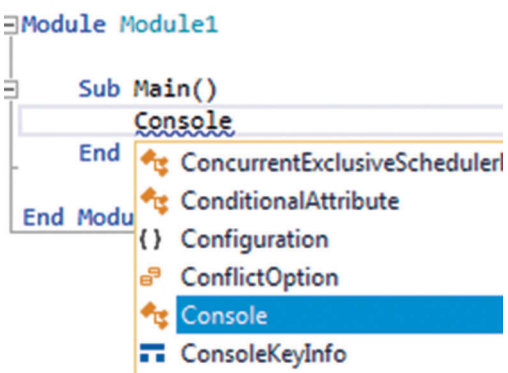


Figure 1.03 Auto-completion window

Type the word 'Console' into the code window (see Figure 1.03). As you type you will notice that the IDE provides an auto-completion window listing all the code inputs or objects that match the letters you have typed. You can double click the correct item, or press Spacebar when the item is highlighted, to auto-complete the entry. This will speed up your coding as once you have typed in the first few characters of the instruction the software will automatically highlight the closest match.

When Console has been completed type a dot symbol. This will show a list of all the available methods for the Console class (see Figure 1.04). The method we need is WriteLine which will display a textual value in the console window when the code is executed. The method has to be passed the required text. To provide the required text the full line will be:

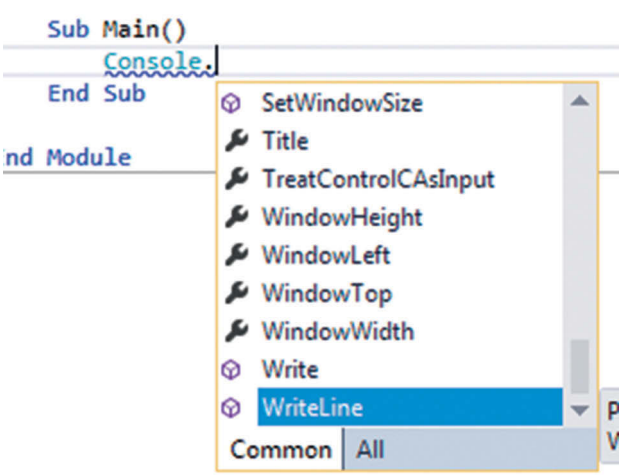


Figure 1.04 Methods window

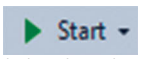
```
Console.WriteLine("Hello World")
```

Note that the text to be included is in speech marks to indicate that it is text and not a reference to another object. This code will display the required text in a console window when the application is run but the window will close as soon as the code has been executed. To prevent this, the console ReadKey method is used to pause the execution until a key is pressed on the keyboard. The final code will be:

```
Module Module1
    Sub Main()
        Console.WriteLine("Hello World")
        Console.ReadKey()
    End Sub
End Module
```

TIP

Type **Imports System.Console** before the **Module Module1** line to avoid having to type 'Console' every time you need to use the class.

To run the code click the Start option on the toolbar 

or use F5 from the keyboard. This will launch the console window (Figure 1.05) and display the text 'Hello World'. The execution of the code will be halted until a key is pressed at which time the window will close.

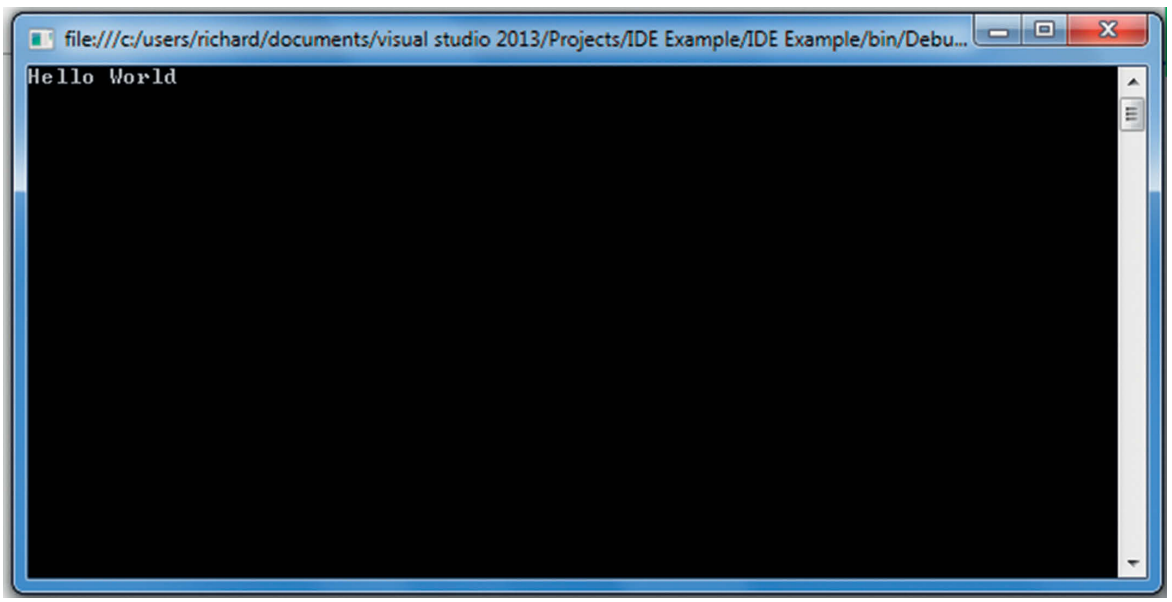
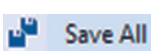


Figure 1.05 Console window

Your first console mode program is complete.

Use 

or the **Save All** option under the **FILE** menu to save your project.

1.05 Windows Forms Application

Console mode makes use of a single user interface to accept text-based inputs and display text-based outputs. Windows Forms provides a visually richer environment which makes use of a range of graphical user interface tools, to produce systems that have more in common with commercial applications.

The interface is more complex as programmers are required to design and produce the graphic user interface that will allow the user to interface with the system. Visual Basic is an event-driven procedural language in which events trigger subroutines that execute the code within them. In this first Windows Forms application clicking a button on the form will trigger an event that delivers the message 'Hello World'.

The default layout (Figure 1.06) contains five main windows.

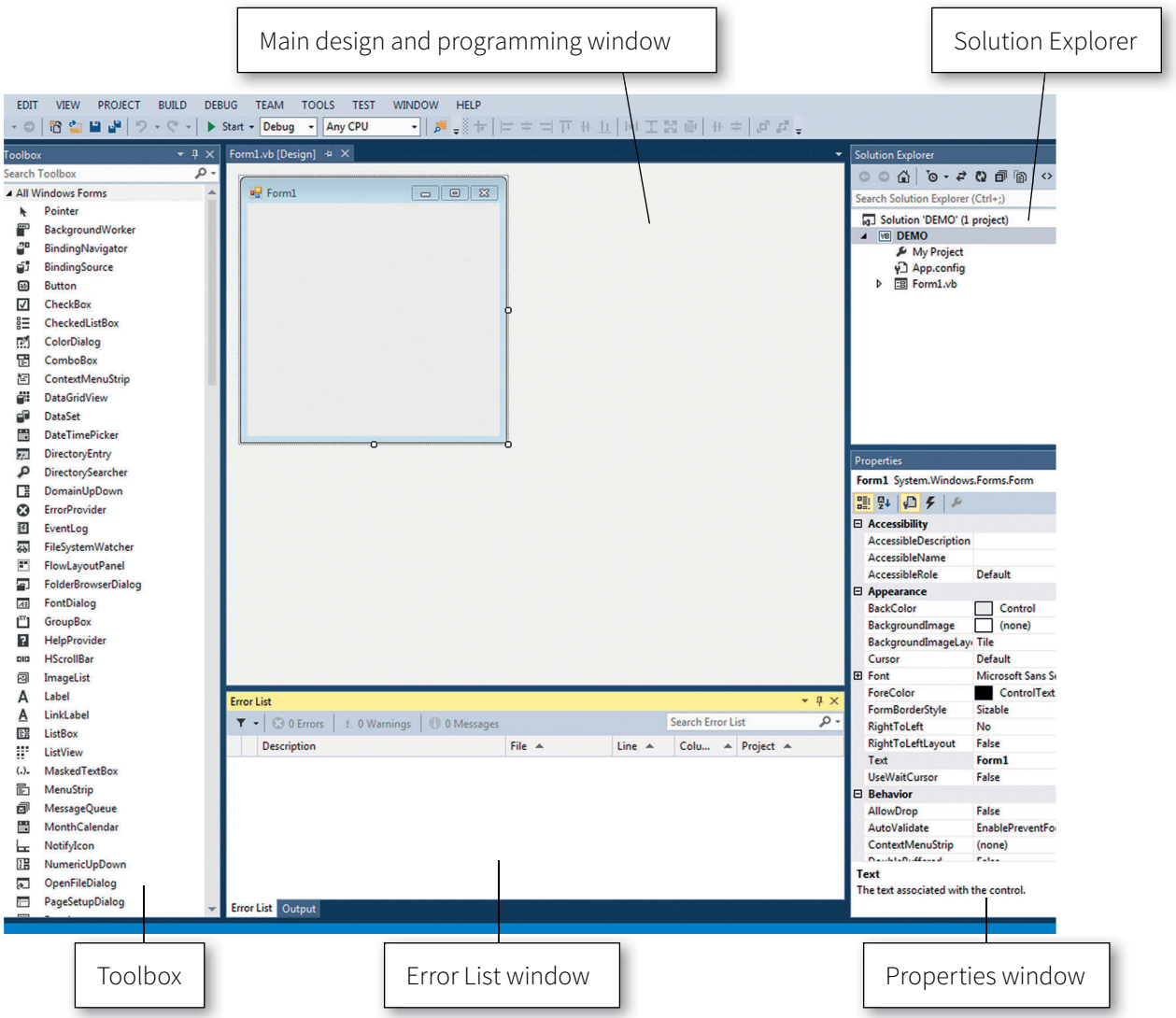


Figure 1.06 Windows Forms application programming interface

Main design and programming window provides an area in which you design your system's interface and write the program code required to accept inputs, process data and produce the required outputs.

Solution Explorer displays the **contents of a solution**, which includes the solution's projects and each project's items. This is where you will find your forms and the program files that support them.

Toolbox provides a set of tools that will allow you to use predefined visual GUI and control objects for the application you wish to develop.

Properties window is used to view and edit configuration-independent, design-time properties and events of selected objects.

Error List window displays any errors, warnings or messages produced as you edit and compile code.

TIP

Individual windows can be docked or set as floating. It is possible to open windows via the **View** menu, and the **Window** menu.

1.06 Make Your First Windows Forms Application

In a Windows Forms Application the traditional 'Hello World' message will be achieved in two steps:

1 Design and construct the user interface

2 Code the program that will generate the required output.

Design the Interface

Find the button object Button in the Toolbox. Click to select the tool and move the mouse over the form in the main design window. The mouse icon will change to show the icon of the selected tool. Click and drag will generate a button on the form. Using the standard Windows mouse controls it is possible to resize and move the button.

Use the same process to generate a textbox object TextBox on the form.

It is considered good practice to give objects meaningful identifiers. An identifier is the name of the object. The default identifier structure is the type of object followed by an increasing number of the objects of that type on the form, such as Button1. The Properties window provides the interface to change the properties of the each object. As you select an object on the form, or the actual body of the form itself the Properties window will change to reflect the properties of the object or form. Objects have many properties that can be configured by the designer but we are initially interested in the properties in Table 1.01.

Table 1.01

Property	What it is
⊟ Design (Name)	The identifier (name) used in the code to identify the object
⊞ Font Microsoft Sans Text Button1	The font style used to display text on or in the object The text that will appear on the object

Use properties to give the button and the textbox meaningful identifiers.

Code the Program

To create or edit the program code that will be activated by the form you have designed you will need to open the code window.

To open the code window select Code from the **VIEW** menu. The code window will open as an additional tab in the main design window. Clicking the tabs will switch between design and code windows.

It will contain only two lines of code but they are important as they indicate the start and end of the code attached to the form (Figure 1.07). All additional code will be placed between these two indicators. Generally code placed outside will not be part of the form and will cause an error. The Enter key can be used to make additional lines.

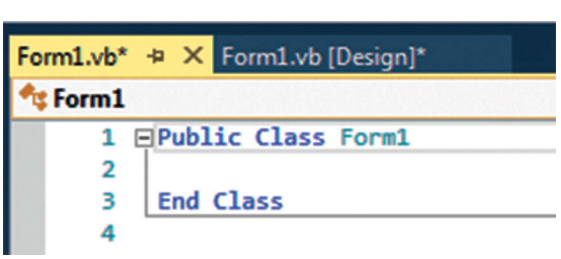


Figure 1.07 Windows Forms application code window

Visual Basic is an event-driven procedural language in which events trigger subroutines that execute the code within them. In this first program clicking the button on the form will trigger an event that delivers the message 'Hello World'. Before writing code the event subroutine has to be created. Creating an event requires you to select the general

object and then the specific declaration required. For example, the Button object can have declaration events such as 'click' and 'mouse over'. These different events can be used to trigger different subroutines.

Figure 1.08 shows both of the drop-down menus but it is not possible for you to view both lists simultaneously in the IDE. When you select an object (from the list shown on the left), the list of declarations (shown on the right) shows only the events that are relevant to this object. Click on the Button1 object to select it. The drop-down menu provides a list of all the possible button events. Select 'Click' to insert the Button Click event.

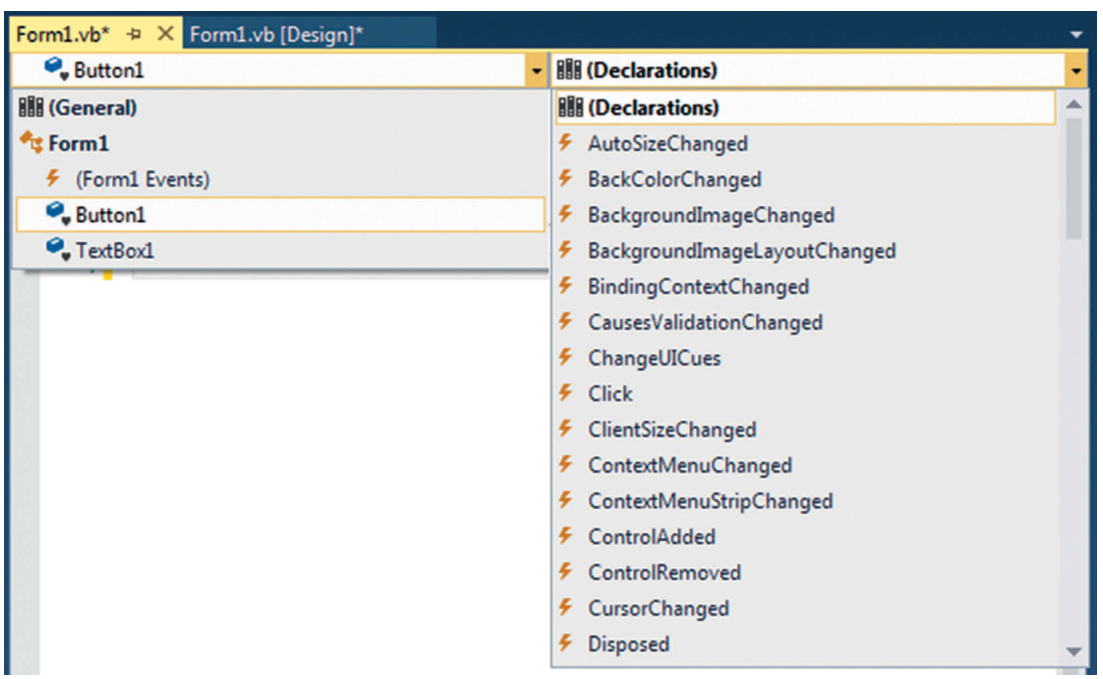


Figure 1.08 Object and Event Declaration selection

TIP

Double clicking an object in the Design window is a shortcut to opening the code window and creating the event subroutine. The software will open the code window and automatically create the default event associated with the object. For example, a Button object's default event is the Click event.

When you select an event the event code will be inserted into your code window.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

End Sub
```

Let us examine the code to identify what each element achieves. Although this makes use of object-oriented language and is outside of the scope of IGCSE, it is useful to have some understanding of how the process works.

Table 1.02

Code element	Description
`Private Sub`	The start of an individual subroutine. Private means that the subroutine is only accessible by this form.
`Button1 _ Click`	The name of the subroutine. The automatic default is to name the routine after the object and event that will trigger the subroutine; however it is possible to rename the subroutine.
`(sender As Object, e As EventArgs)`	The arguments, or data, that are associated with the event. As this is a button click event the arguments are limited – either the button was clicked or it was not. However events associated with mouse activation, for example, will hold data about the location of the mouse on the form and which mouse button was clicked. You should not change or delete any of this data as your subroutine might not work. As you become a more advanced programmer, you will learn how you can manipulate these sections.
`Handles Button1. Click`	The named events that the subroutine can handle. In this case clicking Button1 will call the subroutine and execute the code it contains. It is possible to have a single subroutine triggered by multiple events.
`End Sub`	The end of the subroutine. All the code that is to be executed when the subroutine is called is placed between Sub and End Sub.

In Visual Basic the functionality of a class is accessed by use of the dot symbol. In this example Button1 is an object derived from the Button class and one of the functions available is the Click function. The class library's prewritten code is used when an object of the Button class is clicked. You have no need to write the code – it is contained within the class library and is pretested. As you develop your knowledge of Visual Basic you will find that it makes extensive use of class libraries to provide programmers with functionality.

It is now that we can start to write some code. Within the Button Click subroutine, type in the name of your textbox. As you type you will notice that Visual Basic provides an auto-completion window listing all the code inputs or objects that match the letters you have typed (Figure 1.09). You can double click the correct item, or press Spacebar, to auto-complete the entry.

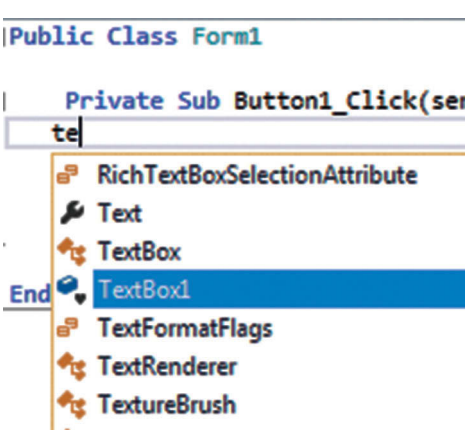


Figure 1.09 Auto-complete window

When the name of the textbox has been entered type a dot symbol. As the textbox is an object of the Textbox class this will show a list of all the available methods which can be attached to a textbox object. The method we need is the Text method which will either set text into a textbox or get text from a textbox (Figure 1.10). Double Click, or press Spacebar, to select this method.

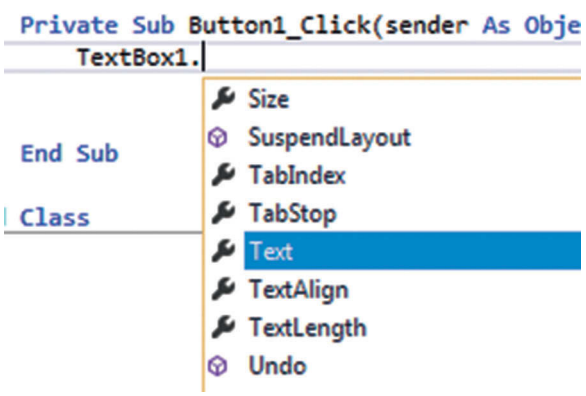


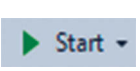
Figure 1.10 List of methods

To indicate the actual text that the method will show in the textbox complete the code as follows:

```
TextBox1.Text = "Hello World"
```

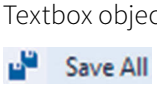

Note that the text to be included is in speech marks to indicate that it is new text and not a reference to another object. The final code will look like this:

```
Public Class Form1

    Private Sub Button1 _ Click(sender As Object, e As EventArgs) Handles Button1.Click

        TextBox1.Text = "Hello World"

    End Sub

End Class
```

To run the code click the Start option on the toolbar 

or use F5 from the keyboard. This will launch the form as a separate interactive window. Click on the button and the text 'Hello World' will appear in the textbox.

Your first Windows Forms Application program is complete. You have made use of the design window and the Toolbox to create the interface. You have generated a subroutine called by the Click method of a Button object. Within that subroutine you have used the Text method of a Textbox object to place programmer-defined text into the textbox.

Use 

or the **Save All** option under the **FILE** menu to save your project.

1.07 The Code Behind the Form

As you may have expected the interface objects created by the Toolbox are supported by code that draws the objects on the form. This is generated for you (Figure 1.11) as you build the required interface.

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class Form1
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.TextBox1 = New System.Windows.Forms.TextBox()
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.BackColor = System.Drawing.SystemColors.ButtonFace
        Me.Button1.Location = New System.Drawing.Point(37, 43)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(79, 42)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Button1"
        Me.Button1.UseVisualStyleBackColor = False
```

Figure 1.11 Example of automatic system generated code supporting the GUI

The file that holds this code is stored in the project folder (Figure 1.12). Until you decide to publish your applications you will not need to have a detailed understanding of the role of the project files.

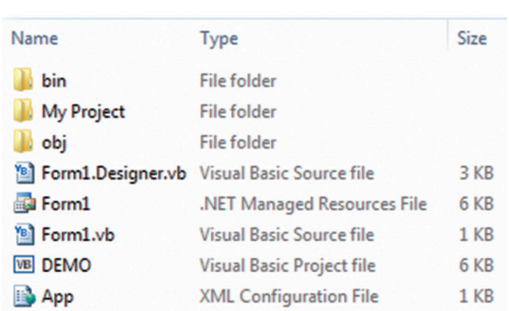


Figure 1.12 Solution Explorer showing project files

1.08 Choosing a Console Application or a Windows Forms Application

Throughout this book the various tasks are completed using one, or sometimes both, of these types of application.

Console Applications offer the benefit of more accurately reflecting the programming style of the IGCSE syllabus and will help prepare you for the expectations of the A level syllabus. In the course, you will not be expected to produce algorithms in any specific language; you will use pseudocode and flowcharts to detail answers to questions. Console applications do not involve the additional complexity of having to reference objects from GUI forms.

Windows Forms Applications will offer a richer visual experience and produce systems similar to those commercially available.

I suggest that making use of both applications will best support the development of your computational thinking.

1.09 Additional Support

The intention of this book is to introduce programming concepts making use of the non-language specific formats included in the syllabus. Visual Basic is used to provide the opportunity for you to use a real programming language to develop your understanding of these concepts. Additional support and guidance on the Visual Basic programming language and Visual Studio Express 2013 can be accessed directly from the Microsoft Virtual Academy.

A range of video tutorials and links to other support can be accessed from http://www.microsoftvirtualacademy.com/training-courses/vb-fundamentals-for-absolute-beginners

Summary

Visual Studio Express provides two coding windows:

- Console Applications provide a simple interface and are one of the required language formats used in A Level Computing. The interface is a simple text based console through which user inputs and outputs are handled.
- Windows Forms Applications offer a richer visual interface for the user of your programs. They involve the use of a design window and a coding window. They offer more flexibility over the way in which user inputs and outputs are handled.

Chapter 2:
Sequence

Learning objectives

By the end of this chapter you will:

- know the difference between the three programming constructs sequence, selection and iteration
- understand the role of flowcharts and pseudocode when designing programs
- understand the main symbols used in flowcharts
- understand the preferred format of pseudocode.

2.01 Logical Design Considerations

When designing programs it is crucial to consider the order in which the task needs to be completed. All tasks will follow some logical order. When working on a solution to a problem you should first apply the top-down design technique, to break down the big problem into smaller ones.

For example to calculate the time it would take to complete a journey you need to know the distance to be travelled and the intended speed. The first logical step would therefore be to calculate the distance to be travelled as without this data the rest of the task could not be completed.

The **sequence** in which instructions are programmed can be crucial. Consider the following algorithm:

```
Distance = Speed * Time
Speed = 12 kilometres per hour
Time = 15 minutes
```

KEY TERM

Sequence: Code is executed in the order it is written.

A human would recognise that the values for speed and time have been given after the calculation. A coded program would simply complete the task in the order given and calculate the distance as zero because at the time of the calculation no values had been provided for speed or time.

A human would probably also recognise the relationship between speed and time, identifying that the speed is quoted 'per hour' but the time is given in minutes and correctly calculate the distance as 3 kilometres (12 * 15/60). Even if the values had been provided before the calculation, as no instructions had been given to convert to a common base the program would calculate distance incorrectly as 180 kilometres by simply multiplying the given values (12 * 15).

2.02 Programming Concepts

Visual Basic and other procedural languages make use of three basic programming constructs. Combining these constructs provides the ability to create code that will follow a logical process. **Selection** and **Iteration** offer a number of alternative approaches and are covered in detail in Chapters 4 and 5.

KEY TERM

Selection: Code branches and follows a different sequence based on conditions being fulfilled.

Iteration: Code repeats a certain sequence a number of times depending on certain conditions.

Sequence

The order in which a process is completed is often crucial to the success of that process. Take the mathematical expression A + B × C + D. The rules of precedence dictate that the multiply operation will be completed first. Had the programmer intended that the operations A + B and C + D be completed before multiplying the two result values then they would have had to be explicit about the required sequence.

In programming the sequence is indicated by the order in which the code is written, usually top to bottom. The program will execute the first line of code before moving to the second and subsequent lines. An example of a sequence error would be completing a process before all the appropriate user inputs had been obtained.

Selection

Often your programs will perform different processes dependent on user input. Consider a system designed to provide access to the school network based on user input of a username and password. The system would need to follow different paths dependent on whether the user input was accurate or not. One path would allow network access if the username and password matched records. If there was no match the system would follow another path in which the user was prompted to re-input details.

Iteration

It is not unusual for a program to perform identical processes on different data items. Consider a program which takes a series of coordinates and produces a line graph. The code that provides the instructions that plot the new coordinates and draw a connecting line from the previous coordinates will be repeated for each of the coordinates given. Iteration provides a method that causes the execution of the code to jump to the beginning of the 'plotting' code sequence for each new set of coordinates.

2.03 Design Tools

When you design programs it is normal to plan the logic of the program before you start to code the solution. This is an important step in the design of effective systems because a flaw in the logic will often result in programs that run but produce unexpected outputs.

The first step in the design process is to break down the problem into smaller problems. This is called 'top-down design'. Once you have the smaller problems defined you can consider each problem separately. This will be easier to plan and finally code. A structure diagram is used to help organise the top-down design. Chapter 6 provides more detail about top-down design and structure diagrams.

The next stage is to design an algorithm for the individual problems. Two approaches that can be used at this stage to help generate logically accurate systems are flowcharts and pseudocode.

To succeed in your course you will be expected to have a working understanding of flowcharts and pseudocode and to be able to use them to answer questions that require you to explain the logic of your solutions to given tasks. Both methods are used throughout this book to indicate the logic of systems and it important that you become familiar with their use.

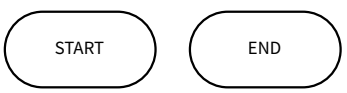
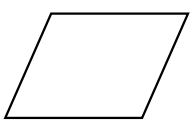
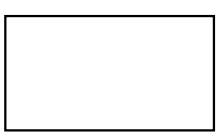
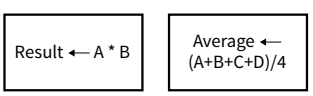
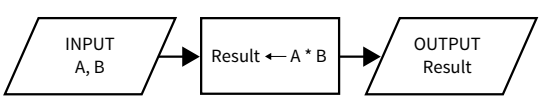
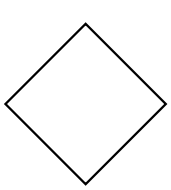
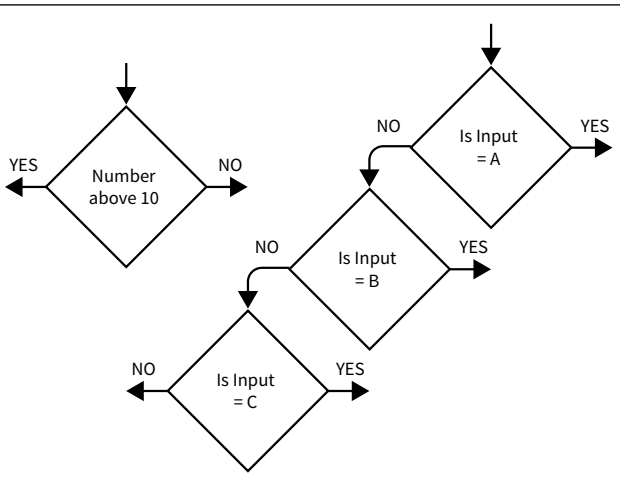
SYLLABUS CHECK

Problem solving and design: Use flowcharts and pseudocode.

2.04 Flowcharts

Flowcharts are graphical representations of the logic of the intended system. They make use of symbols to represent operations or processes which are joined by lines that indicate the sequence of operations. Table 2.01 details the symbols used.

Table 2.01

Symbol	Use	Example
 Terminator	The START or END of a system	
 Input or output	A required INPUT from the system user or an OUTPUT to the system user The value being input or output is written on the symbol.	
 Process	A process within the system The flowchart should show sufficient detail to indicate how the proposed process is to be achieved. Beware of making the process too generic. For example if the system was required to calculate an average value, a process entitled 'Calculate Average' would be too generic. It needs to indicate the inputs or other values used to calculate the average.	
Data flow line	Joins two operations The arrowhead indicates the direction of the flow. Iteration (looping) can be indicated by a flow returning to an earlier process in the flowchart.	
 Decision	A point in the sequence where alternative paths can be taken The condition on which the flow is determined is written within the symbol. Where multiple alternatives exist, sequence flows are indicated by chained decision symbols. Each 'No' condition directs to another decision in the process.	

2.05 Pseudocode

Pseudocode is a method of describing the logic and sequence of a system. It uses keywords and constructs similar to those used in programming languages but without the strict use of syntax required by formal languages. It allows the logic of a system to be defined in a language-independent format for a programmer to code using any programming language which is appropriate to the context.

While the programming code required to perform processes can vary considerable across differing languages, the same pseudocode line could be used to describe the logic of a system intended to be written in any language.

Pseudocode follows a number of underlying principles:

- Use capital letters for keywords close to those used in programming languages.
- Use lowercase letters for natural language descriptions.
- Use indentation to show the start and end of code statements, primarily when using selection and iteration.

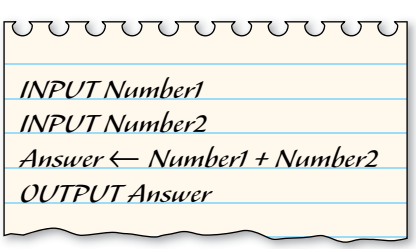
One of the advantages of learning to program using Visual Basic is that the actual coding language is structured in a similar way to natural language and therefore closely resembles pseudocode. Visual Basic also automatically indents instructions where appropriate similar to the approach that should be adopted when writing pseudocode.

SYLLABUS CHECK

Pseudocode: understand and use pseudocode for assignment, using ← .

2.06 Pseudocode Example

This pseudocode is for an algorithm that accepts the input of two numbers. These values are added together and the result is stored in a memory area called Answer. The value in Answer is then displayed to the user. (In Chapter 3 we will learn that this memory area is known as a variable.)



Note the use of ← to show the passing of values. This is distinct from the use of the equals symbol (=) which is used to indicate a comparison of two values. (Visual Basic does not have the ← symbol and uses the = symbol in both situations.)

2.07 Effective use of Flowcharts and Pseudocode

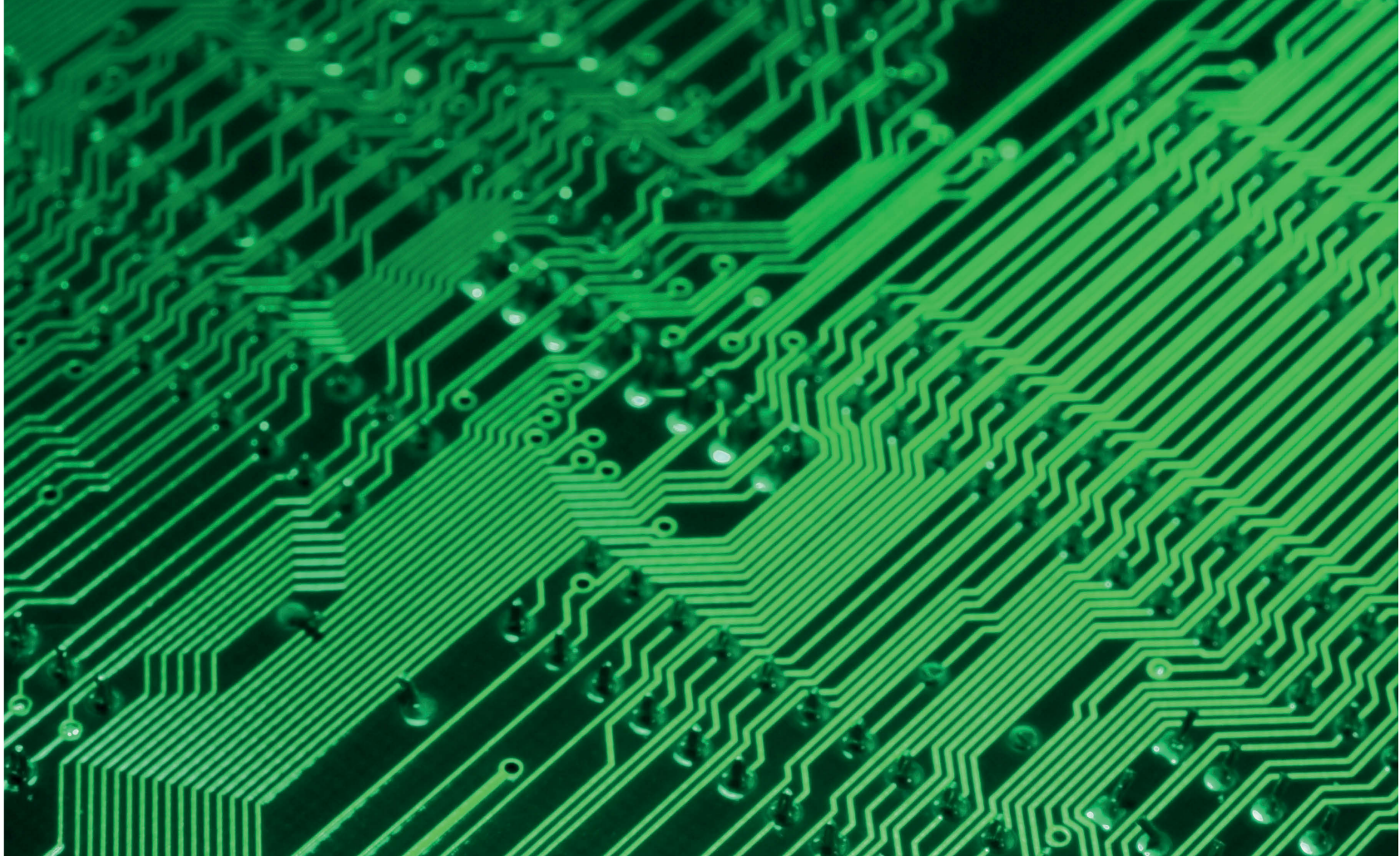
Because of the universal nature of flowcharts and pseudocode they are used extensively in the IGCSE Computer Science syllabus.

The aim of this book is to help you to learn to design effective systems using the programming language Visual Basic. The following chapters make use of flowcharts and pseudocode to define the logic of systems, before moving on to specific Visual Basic coded solutions.

Learning how to detail the logic of programs through the use of these design techniques will be a crucial step not only in your preparation for examination but also for your preparation in using the languages of the future. Language syntax is likely to change in the future but the need for effective logical and computational thinking will remain a constant.

Summary

- Programmers make use of three constructs when writing code:
  - sequence: the logical order in which the code is executed
  - selection: branching of code onto different paths based on certain conditions
  - iteration: repeating of sections of code.
- Before coding a program it is crucial to design an appropriate algorithm.
- Flowcharts and pseudocode are tools used in the design of algorithms.

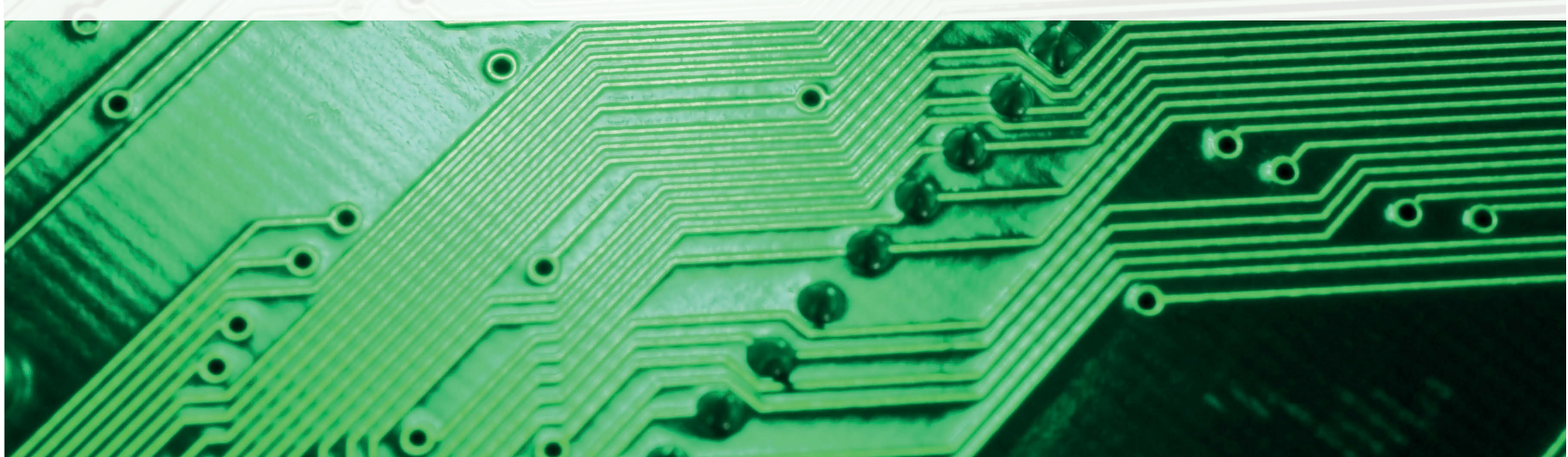


Chapter 3:
Variables and Arithmetic Operators

Learning objectives

By the end of this chapter you will understand:

- how to declare and use variables and constants
- and be able to use the data types Integer, Real, Char, String and Boolean
- how to use basic mathematical operators to process input values
- how to design and represent simple programs using flowcharts and pseudocode.



3.01 Variables and Constants

Programs are normally designed to accept input data and process that data to produce the required output. Data used in programs can vary depending on the aim of the program; a calculator will process numerical data while a program designed to check email addresses will process textual data. When writing programs you will use variables or constants to refer to these data values. A **variable** identifies data that can be changed during the execution of a program while a constant is used for data values that remain fixed. The **metadata** you provide about a variable or constant will be used by the computer to allocate a location in memory in which the data will be stored.

KEY TERM

Variable: The identifier (name) given to a memory location used to store data; the value can be changed during program execution.

Metadata: Data about data; information about the structure or format of the data stored.

3.02 Types of Data

In addition to giving the variable or constant an identifier (name) which is used as a label by the computer to reference the allocated memory, it is also important to provide information about the type of data so that the appropriate amount of memory can be reserved. For example storing a large decimal number will require more memory bytes than storing a single character.

To support this process different data types exist. The basic data types you will need to use are identified in Table 3.01.

Table 3.01

Data type	Description and use	Visual Basic
Integer	Whole numbers, either positive or negative Used with quantities such as the number of students at a school – you cannot have half a student.	Can store values ranging from –2 147 483 648 to 2 147 483 647. Uses 4 bytes of memory. If a decimal value is put into an Integer variable, the value is rounded to the nearest whole number.
Real	Positive or negative fractional values Used with numerical values that require decimal parts, such as currency. Real is the data type used by many programming languages and is also referenced in the IGCSE Computing syllabus.	Visual Basic does not use the term Real, the equivalent data type is called '**Decimal**'. The range of values depends on the number of decimal places required. Uses 16 bytes of memory. Stores a much larger range of numbers than the Integer data type. Single and Double also hold fractional numbers.
Char	A single character or symbol (for example, A, z, $, 6) A Char variable that holds a digit, cannot be used in calculations.	Stores a single Unicode character. Uses 2 bytes of memory.
String	More than one character (a string of characters) Used to hold words, names or sentences.	Can store a maximum of approximately 2 billion Unicode characters.

Data type	Description and use	Visual Basic
Boolean	One of two values, either TRUE or FALSE Used to indicate the result of a condition for example, in a computer game a Boolean might be used to indicate if a player has achieved a higher level.	

You can find more information about the data types in Visual Basic at http://msdn.microsoft.com/en-us/library/47zceaw7.aspx.

SYLLABUS CHECK

Programming concepts: understand and use Integer, Real, Char, String and Boolean.

3.03 Pseudo Numbers

Telephone numbers and ISBN numbers both consist of digits but are not truly numbers. They are only a collection of digits used to uniquely identify an item; sometimes they contain spaces or start with a zero, and they are not intended to be used in calculations. These are known as 'pseudo numbers' and it is normal to store them in a String data type. If you store a mobile phone number, as an Integer any leading zeros will be removed and no spaces or symbols will be permitted.

3.04 Declaring Variables and Constants

You will need to select an identifier (name) and a data type for your variables or constants. In Visual Basic it is possible to declare variables without declaring a data type but it is considered good practice to define the data type. This is known as 'strong typing'; it allows the compiler to check for data type mismatches and results in faster execution of the code.

SYLLABUS CHECK

Programming concepts: declare and use variables and constants.

When naming your variables or constants use identifiers that have a meaningful link with the data being stored. For example if you are storing the high score of two players in a game use the names *Player1HighScore* and *Player2HighScore*. Declaring variables with relevant identifiers will help to make your code easier to read and maintain.

It is not possible in Visual Basic to have spaces in identifiers. For identifiers that incorporate more than one word it is normal to start each new word with a capital letter. This is known as CamelCase. Identifiers can include digits, for example Number1, but cannot begin with a digit. Words that are used by Visual Basic are known as reserved words and cannot be used as variable names. For example it is not possible to name a variable 'Integer' or 'Boolean'.

It is important to select the most appropriate data type for your variables. Using inappropriate data types may result in your programs returning unexpected results. For example using the data type Integer to store currency values could result in the decimal element being rounded and the wrong values being output.

It is also considered good practice to give variables an initial value when declaring them: this is known as 'initialising'. Uninitialised variables will hold the default values set by Visual Basic. Some examinations may expect you to initialise variables so getting into the habit is a good idea.

3.05 Declaring Variables in Visual Basic

Declaration of variables is achieved by using the code format shown in Figure 3.01.

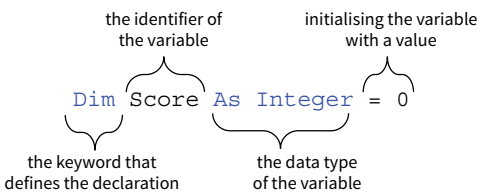


Figure 3.01 Declaring variables in Visual Basic

In the following declarations the variable identifier, data type and initial value are declared.

```
Dim PlayerName As String = ""
Dim DrivingLicence As Boolean = False
Dim Payment As Decimal = 0
```

3.06 Declaring Constants in Visual Basic

The code for declaring constants follows a similar format to that used for variables, but uses the keyword 'Const' to replace 'Dim'.

Here is an example of declaring a **constant**:

```
Const Pi As Decimal = 3.14159
```

KEY TERM

Constant: A named memory location which contains data that can be read but not changed by the program.

3.07 Variable Scope

When declaring a variable the placement of the declaration in the code will determine which elements of the program are able to make use of the variable.

Global variables are those that can be accessed from any routine within the program. They are used for variables that need to be accessed from many elements of your program. To give a variable global status it must be declared outside of any specific subroutine. It is good practice to make all the global declarations at the start of the code.

Local variables can only be accessed in the code element in which they are declared. They are used when the use of the variables will be limited to a single routine. Using local variables reduces the possibility of accidently changing variable values from other code elements.

Figure 3.02 shows the same code in a Console Application and in a Windows Forms Application. There are two global variables (Score and PlayerName) and one local variable (Result).

```
Module Module1

    Dim Score As Integer = 0
    Dim PlayerName As String = ""

    Sub Main()
        Dim Result As Integer = 0

    End Sub

End Module
```

Console mode example

```
Public Class Form1

    Dim Score As Integer = 0
    Dim PlayerName As String = ""

    Private Sub Button1_Click(sender As Object, e As EventArgs)
                Handles Button1.Click

        Dim Result As Integer = 0

    End Sub
End Class
```

Windows Form mode example

Figure 3.02 Declaring Global and Local Variables

3.08 Arithmetic Operators

There are a number of operations that can be performed on numerical data. Combining these operations and appropriate variables allows you to create programs that are capable of performing numerical computation tasks.

The basic operators used in Visual Basic are shown in Table 3.02.

Table 3.02

Operation	Example of use	Description
Addition	`Result = Number1 + Number2`	Adds the values held in the variables Number1 and Number2 and stores the result in the variable Result.
Subtraction	`Result = Number1 - Number2`	Subtracts the value held in variable Number2 from the value in Variable Number1 and stores the result in the variable Result.
Multiplication	`Result = Number1 * Number2`	Multiplies the values held in variables Number1 and Number2 and stores the result in the variable Result.
Division	`Result = Number1 / Number2`	Divides the value in variable Number1 by the value in Number2 and stores the result in the variable Result

TIP

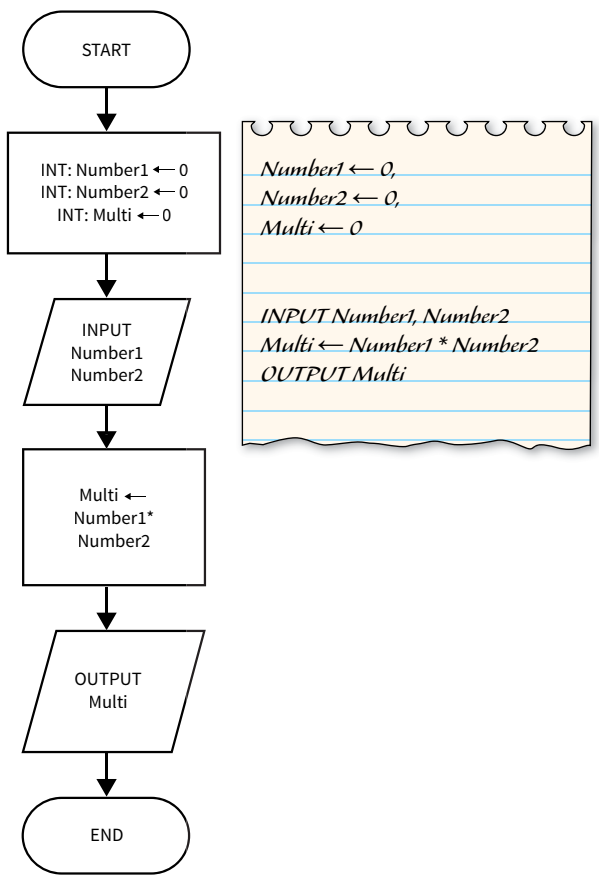
As a division operation can result in a fractional value it would be normal to use a Decimal (or Real) data type to hold the Result.

3.09 Programming Tasks

Multiply Machine

The Multiply Machine takes two numbers input by the user, multiplies them together and outputs the resultant value.

First you need to design the algorithm. Figure 3.03 shows flowchart and pseudocode solutions for the task.



Note the use of the correct and recognised symbols in the flowchart and pseudocode:

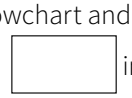
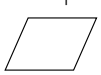
-  indicates a process such as completing the multiplication.
-  indicates either an INPUT or OUTPUT.
- ← indicates the assigning of a value; it is used when initialising variables or passing new values to a variable.

Figure 3.03 Pseudocode and flowchart for Multiplication algorithm

In Visual Basic assigning is indicated by the use of the = symbol. In pseudocode the ← symbol is used.

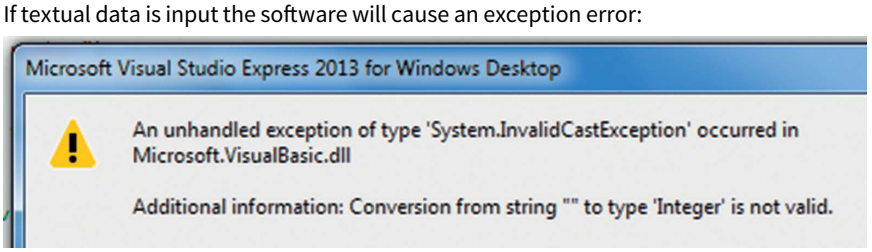
SYLLABUS CHECK

Pseudocode: understand and use pseudocode commands INPUT and OUTPUT.

TIP

Variables that have been declared with numerical data types such as Integer or Decimal can only accept numerical data.

If textual data is input the software will cause an exception error:



A null input will also cause this error.

Here is the console mode program implementation of this solution:

```
Module Module1

    Sub Main()
        'Declaration and initialising of required local variables
        Dim Number1 As Integer = 0
        Dim Number2 As Integer = 0
        Dim Multi As Integer = 0

        'Display a request for the first number
        Console.WriteLine("Please insert first number")
        'Store the user input into the Number1 variable
        Number1 = Console.ReadLine()

        'Request and store second user input
        Console.WriteLine("Please insert second number")
        Number2 = Console.ReadLine()

        'Storing in the local variable the multiplication result
        Multi = Number1 * Number2

        Console.WriteLine("The answer is")
        'Displaying the value held in the variable Multi
        Console.WriteLine(Multi)

        'ReadKey used to pause the console window
        Console.ReadKey()
    End Sub

End Module
```


If you are using the Windows Forms application you will need to design an interface that is capable of taking two values and displaying a result. It could look something like Figure 3.04.

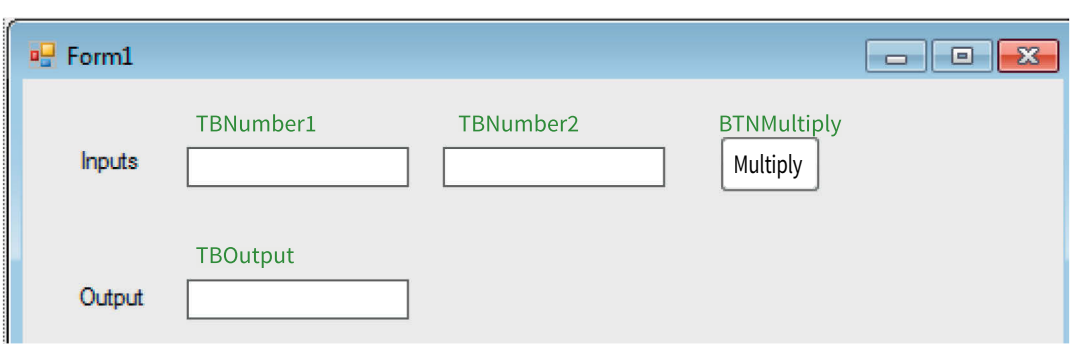


Figure 3.04 Windows Forms interface design

Remember to give the design elements of the form appropriate names. This example has used the names shown in green on Figure 3.04.

The code to achieve this solution should be run under the button click event.

```
Public Class Form1

    Private Sub BTNMultiply _ Click(sender As Object, e As EventArgs) Handles BTNMultiply.Click

        'Declaration and initialising of required local variables
        Dim Number1 As Integer = 0
        Dim Number2 As Integer = 0
        Dim Multi As Integer = 0

        'Storing the values input in the textboxes to the variables
        Number1 = TBNumber1.Text
        Number2 = TBNumber2.Text

        'Storing in the local variable the multiplication result
        Multi = Number1 * Number2

        'Displaying the value held in the variable Multi in the output text box
        TBOutput.Text = Multi

    End Sub

End Class
```

EXTENSION TASK

Multiply Machine Extension Task

Extend this program to include addition, subtraction and division buttons. It will have to be programmed using a Windows Forms application.

Points for discussion:

1. Should all the variables be declared locally?
2. Is Integer an appropriate data type for all the resultant output variables?

Volume of Water in Aquarium

Design a program where the inputs will be the height, width and depth of an aquarium. The output should be the number of litres of water that the aquarium will hold (1 litre = 1000 cm^3).

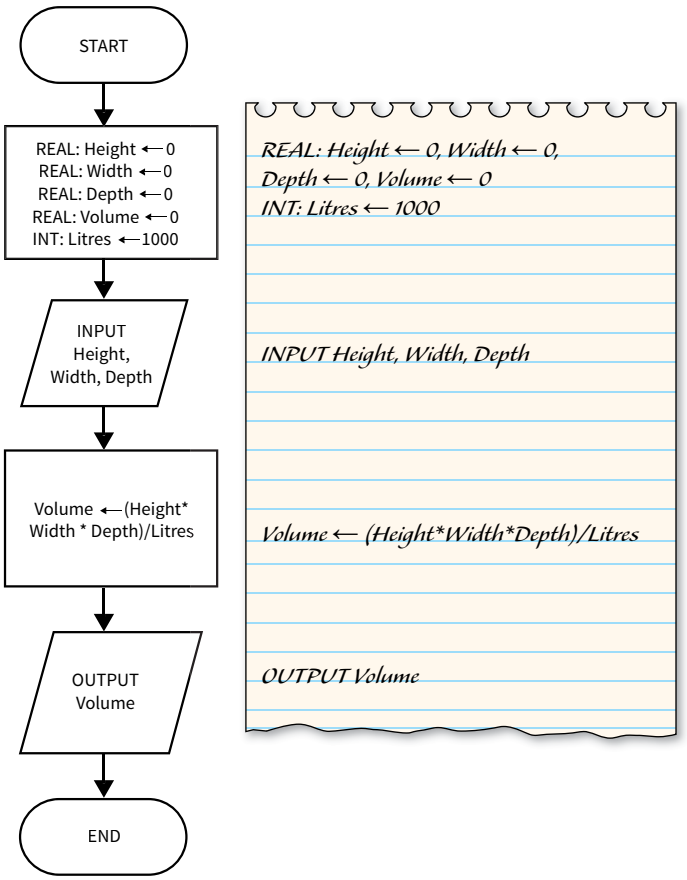


Figure 3.05 Flowchart and pseudocode for the aquarium algorithm

The following code shows the console mode program implementing this solution. Note how the logical sequence of the code follows the flowchart or pseudocode design.

```
Module Module1

    'Global variables to hold the inputs
    'Note the addition of the 1 to the names to overcome the reserved word conflict
    Dim Height1 As Decimal = 0
    Dim Width1 As Decimal = 0
    Dim Depth1 As Decimal = 0
    'Constant to hold the ratio of cubic centimetres to litres
    Const Litres As Integer = 1000

    Sub Main()
        'Local variable to hold the resultant volume
        Dim Volume As Decimal = 0

        'Request and store user inputs
        Console.WriteLine("Please insert Height and press Return")
        Height1 = Console.ReadLine
        Console.WriteLine("Please insert Width and press Return")
        Width1 = Console.ReadLine
        Console.WriteLine("Please insert Depth and press Return")
        Depth1 = Console.ReadLine
```

```
        'Calculate Volume and store in local variable
        Volume = Height1 * Width1 * Depth1
        'Convert the value in volume current in cubic centimetres to litres
        Volume = Volume / Litres

        'Display the value held in the variable Volume
        Console.WriteLine("The volume is")
        Console.WriteLine(Volume)

        Console.ReadKey()
    End Sub
End Module
```

Using the Windows Form application the interface could look something like Figure 3.06.

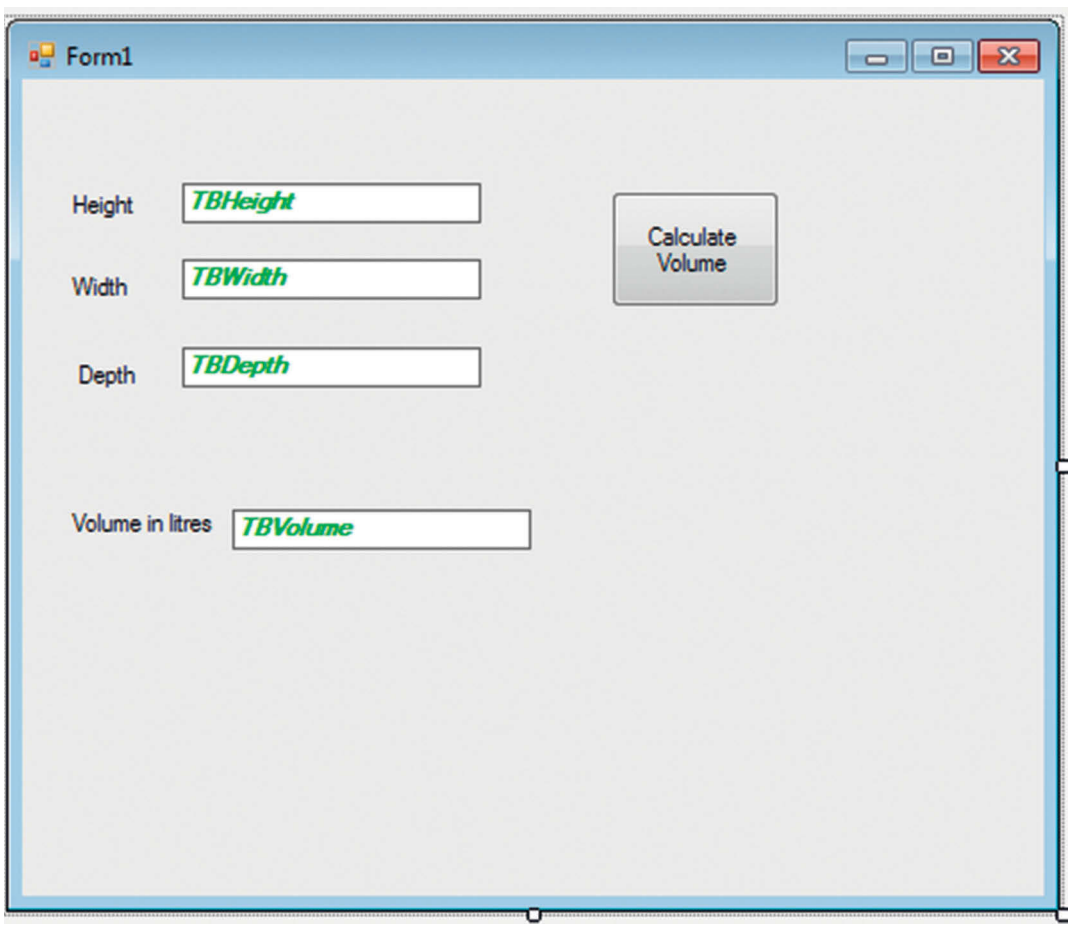


Figure 3.06 Interface design for Windows Forms Aquarium algorithm

Remember to give the design elements of the form appropriate names. Figure 3.06 shows the names in green.

The Windows Form application code to achieve this solution should be run under the button click event:

```
Public Class Form1

    'Global variables to hold the inputs
    'Note the addition of the 1 to the names to overcome the restricted word conflict
    Dim Height1 As Decimal = 0
    Dim Width1 As Decimal = 0
    Dim Depth1 As Decimal = 0
    'Constant to hold the ratio of cubic centimetres to litres
    Const Litres As Integer = 1000

    Private Sub BTNVolume_Click(sender As Object, e As EventArgs) Handles BTNVolume.Click

        'Local variable to hold the resultant volume
        Dim Volume As Decimal = 0

        'placing values from the textbox into variables.
        Height1 = TBHeight.Text
        Width1 = TBWidth.Text
        Depth1 = TBDepth.Text

        'Calculate Volume and store in local variable
        Volume = Height1 * Width1 * Depth1
        'Convert the value in volume current in cubic centimetres to litres
        Volume = Volume / Litres

        'Output the value in local variable to the textbox
        TBVolume.Text = Volume

        'The volume calculation could have been achieved in one calculation
        'Volume = (Height1 * Width1 * Depth1)/Litres

    End Sub
End Class
```


TASK

Area and Circumference of a Circle

A system takes the radius of a circle as its input and calculates the area of the circle and the circumference.

1. Draw a flowchart and create a pseudocode algorithm that will output the area of the circle and the circumference based on the input radius.
2. Test that your algorithm works by programming and running the code in Visual Basic.

3.10 Development Challenges

Challenge yourself, or your colleagues, to complete a programming task. The following are some examples of the type of task you might like to consider. The last two are complex mathematical challenges.

For each challenge, you should draw a flowchart and create a pseudocode algorithm before programming and running the code in Visual Basic.