INTRODUCTION TO PROBLEM SOLVING AND BASIC COMPUTER PROGRAMMING USING VB.NET

for the subject

INTRODUCTION TO SYSTEMS DEVELOPMENT

FET NCV NQF Level 2

(TOPIC 4)

Notes prepared by:

Erasmus HG

Pretorius CM

© COPYRIGHT: THE AUTHORS

All rights reserved.

These notes remain the intellectual property of the authors. Copyright assigned to FET-colleges and to the DoE to duplicate the document in hard copy format only. Any other reproduction method is prohibited. No changes or alterations, in any way, are permitted.

Suggestions and comments may be e-mailed to:

henrietteerasmus@gmail.com

April 2008

PREFACE

The following notes were developed to cover the missing outcomes from the amended Subject and Assessment Guidelines of December 2007 (NCV-document) for the subject INTRODUCTION TO SYSTEMS DEVELOPMENT.

These notes are intended to be used as a source to cover the newly included subject outcomes for Topic 4. The nature of this document and subject reflect and exhibit cross curricular content. Therefore many concepts covered in the notes will also address some of the subject outcomes of other topics in the NCV-document, i.e.

- Topic 2, SO3, LO1
- Topic 2, SO4, LO1
- Topic 2, SO4, LO2
- Topic 3, SO1, LO1
- Topic 3, SO1, LO3
- Topic 5, SO1, LO1
- Topic 5, SO1, LO2
- Topic 5, SO1, LO3

The notes reflect the interpretation of the authors regarding Topic 4 of the NCV-document. It remains the responsibility of the lecturer to ensure that all subject outcomes in the NCV-document are covered.

The notes focus on the basic principles of problem solving and programming as directed from each of the subject outcomes. Each of these major subject outcomes for Topic 4 is covered. These outcomes include

- Describe the term problem solving
- Produce and document an algorithm
- Produce and document pseudocode for a given problem
- Produce and implement alternate design methods to document a specification or solution for a given problem
- Demonstrate an understanding of computer data types
- Implement a program language to solve a given problem

Various problems have been supplied and the subsequent steps to develop solutions have been provided and discussed in detail. The authors followed a step-by-step approach from problem to solution, including the following

- Identifying appropriate input and output variables
- Algorithm design, as supplied in the IPO-chart
- Detailed algorithm in pseudocode
- Selection of appropriate data types for variables
- Selection of relevant components and designing a user interface

- Program implementation in Visual Basic.NET which includes
  - Form design
  - Coding
- Testing and debugging the program with suitable test data

Solutions include sequence, selection and/or iteration control structures.

The authors are aware that the technique of implementing input boxes in the examples pertaining to loops is not entirely used in industry or as a norm. However, due to the nature of this document and the level of complexity involved, input boxes used in a loop to capture data illustrate the concept sufficiently. In later levels of problem solving, loops and iteration structures could be illustrated in conjunction with topics such as arrays, file processing, string manipulation, animation etc.

The primary design method used in these notes is that of pseudocode. Alternative methods are discussed in Appendix A, where a clear example of each method is provided. It is the responsibility of the lecturer to ensure that these alternative methods are also covered as part of your lesson planning. A student that is able to develop proper pseudocode should experience no difficulty in drawing a flowchart or a Nassi Shneiderman diagram.

Visual Basic.NET was chosen as a programming language to implement the solutions because it consists of a set of tools and technologies to assist in the easy development and creation of Windows based applications.

In this language, the procedure header for every button click event is a very lengthy statement such as

```
Private Sub btnClear_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnClear.Click
```

Therefore, in every program example, the authors have made use of the ellipse as a method to save space and to fit the procedure header onto one line.

```
Private Sub btnClear_Click(. . .) Handles btnClear.Click
```

The notes contain various questions and exercises for you to test and apply your knowledge and they are all identified by one of the following icons:



Questions

This icon will be followed by short questions that are suitable for class discussions. Sometimes these questions are used to refresh your memory on previous chapters and other times questions about the newly discussed content.



Exercises

This icon will be followed by exercises that the student must solve on paper. This could possibly lead to class discussions at a later stage, but students must first think about the problems and solve it on paper by applying the knowledge they have learned.



Practical

This icon precedes practical assignments that must be solved in Visual Basic.NET. Some of these assignments may already have been planned in previous exercises, but others must be planned and designed from scratch. All of these solutions must be tested with test data.

We hope that these notes will kindle your enthusiasm for problems solving and programming and motivate you to work hard in achieving your goals to become a developer in the IT-Field.

These notes will equip you to solve any basic computer related problem and to write the corresponding program solution in VB.NET. Remember, the control structures forms the foundation for most program solutions. Enjoy your journey in exploring ways of solving problems by making use of the basic control structures!

Science starts only with problems

Karl Popper

To fail to plan, is to plan to fail.

Robert Wubbolding

Success is the natural consequence of consistently applying the basic fundamentals.

Jim Rohn

TABLE OF CONTENT

CHAPTER		PAGE
1	Understanding problems and how to solve them by using a computer	1
2	Introduction to Visual Basic.NET (Visual Studio 2005)	8
3	General concepts and arithmetic	33
4	Creating a user interface with basic controls	62
5	Write algorithms and programs in sequential steps	88
6	The selection control structure – part 1	105
7	The selection control structure – part 2	151
8	Iteration using a fixed count loop	188
9	Iteration using the do-loop	218
APPENDIX		
A	Tools for planning programs	249
B	Unified Modelling Language	270
BIBLIOGRAPHY		275

CHAPTER 1
UNDERSTANDING PROBLEMS AND HOW TO SOLVE THEM BY USING A COMPUTER

1.1 INTRODUCTION

This chapter will introduce you to the basic goal of this course – understanding problems and how to solve it. We will start by introducing problems encountered in everyday life situations and will then move on to problems that can be solved by using a computer.

OUTCOMES

When you have studied this chapter you should be able to:

- read a problem and analyse it in order to understand exactly what it is that must be solved,
- understand how a computer processes data,
- name the steps to be taken to find an excellent, efficient and working solution for any problem,
- know what an algorithm is and how it can be applied to solve a simple problem

1.2 PROBLEM SOLVING

When a student starts reading these notes and wants to learn how to solve a problem by using a computer, it is first of all important to understand what the problem is!

The student must read the problem statement a number of times to ensure that he/she understands what is asked before attempting to solve the problem.

The following steps need to be followed:

- read the problem carefully
- understand what the problem entails
- and only then, write down the steps to solve the problem.

An everyday example will be:
I need to go to school. Write down the steps to take from waking up in the morning until I am at school.

This problem statement is very simple and once I have read it, I understand what is asked. But now I need to write down the steps:

1. Wake up.
2. Get out of bed.
3. Wash.
4. Put on clothes.
5. Prepare something to eat.
6. Eat.
7. Take my bag.
8. Go to school.

These are the basic steps. It is possible to include more steps to ensure that nothing is left out and it is also possible to omit certain steps e.g. prepare something to eat, because my mother prepared food; or I do not eat in the mornings. But on the other hand, it is not possible to go to school before waking up, getting out of bed, or putting on clothes! Some of the steps can also be swapped. You may eat your food before getting dressed.

But very important, your steps must always be in a logical order!

These steps are called an **algorithm** that can be defined as a set of sequential instructions to solve a problem.

1.3 UNDERSTANDING THE PROBLEM

When the programmer is faced with a problem to solve, the problem must be read carefully and maybe, read a number of times to be very sure that the problem is understood.

It is advisable to delete all unnecessary information given in the problem statement.

Example: Calculate the floor area of the boardroom that is on the second floor of the building next to the road to Johannesburg. The length of the room is 7 meters and the width is 4 meters. It is the largest room on the second floor.

This problem statement can be simplified by deleting the unnecessary information as follows:

Calculate the floor area of the boardroom ~~that is on the second floor of the building next to the road to Johannesburg~~. The length of the room is 7 meters and the width is 4 meters. ~~It is the largest room on the second floor.~~

The problem statement is now much easier to read and understand:

Calculate the floor area of the board room. The length of the room is 7 meters and the width is 4 meters.

The problem statement must include enough information to enable the programmer to solve the problem. But, if some data (values) are missing, it must be of such a kind that the user will be able to supply the missing data.

Example: The sum of 2 numbers must be calculated. The problem statement does not supply the values of the numbers, but the algorithm can ask the user to supply the value of the numbers.

The most important aspect of solving a problem by using a computer is to write an algorithm to solve it. An algorithm is a set of steps that must be written in such a way that is it unambiguous and precise. The computer cannot think for itself - you as the programmer must tell the computer exactly what to do. You may never assume that the computer will do something if you have not explicitly included the specific step.

1.4 DATA PROCESSING

An algorithm to solve a computer based problem consists of 3 phases i.e.

- What you have available for the algorithm to solve the problem
- How you are going to solve the problem i.e. what steps you are going to take
- What is the required result

Schematically it can be shown as follows **(Figure 1-1)**:



or



Figure 1-1

Each of the figures above clearly indicates that the algorithm must receive data (input) that must be processed to render meaningful results (output or information).
In other words:

- Input is processed to render meaningful output.
- Data is processed to render meaningful information.

Example:

The manager of a company asks a student to do some part time work for which he will be paid per hour.
The student will have to know how many hours he has to work at what rate of pay before he can calculate the final amount he will receive.

In this case the input (data) to this algorithm will be the hours and the hourly rate of pay. The input (data) must be processed to give the output (information) i.e. the amount the student will receive.

It is clear that the input (data) is meaningless unless it is processed, because if the student knows how much he will receive per hour, but does not know how many hours he will work, the pay cannot be calculated! On the other hand, if the student has worked for 20 hours but the manager does not tell him how much he will receive per hour, it is also meaningless.

Now it is possible to formulate the following:

- Unprocessed data is meaningless.
- Information is processed data.
- Output is processed data or processed input

1.5 SUMMARY OF THE DATA PROCESSING PROCESS

To summarise and formulise what is described in this chapter, the following steps must be taken to find an excellent, efficient and working solution:

1. Analyse the problem.
   Read the problem statement carefully and determine exactly what output is needed. Also write down what input is available and what must still be supplied by the user.

2. Identify ways to solve the problem.
   It is possible that there may be different ways to solve the problem. Identify these solutions and select the best possible solution.

3. Write an algorithm that contains all the steps in sequential and logical order to solve the problem. The steps must be written in an unambiguous way.

4. Evaluate the algorithm for correctness and efficiency.
   To ensure that the problem will be solved by processing the algorithm, the programmer has to check all the steps in logical sequence. The programmer can use different input values to ensure that the correct output is given using the most efficient algorithm.

Example:

Problem:

Calculate the sum of 2 numbers and display the result on the screen:

Input: Not available in the problem statement. The user has to supply the numbers.
Processing: Add the 2 numbers to determine the sum.
Output: The sum as calculated in the processing phase is displayed on the screen of the computer.

Note the following:

- The value of the numbers must be available before the sum can be calculated.
- It is impossible to display the sum before it has been calculated.

These two remarks indicate that the steps must always be processed in a logical order.

1.6 WRITING AN ALGORITHM

There are different ways of planning to solve a computer related problem, but in this course we are going to write algorithms in *pseudocode* to plan a solution. The solution for a problem can also be planned with flowcharts or with Nassi-Shneiderman diagrams (discussed in Appendix A).

Pseudocode is a specific way of writing the steps of the algorithm in English using specific words.

Example:

Problem:

Determine the weekly wage of an employee if the hours he has worked, and the hourly rate are entered by the user.

Prepare the algorithm to solve the problem:

1. Ask: How many hours did the employee work?
2. Enter number of hours
3. Ask: What is the hourly rate of pay?
4. Enter hourly rate of pay
5. Wage = number of hours X hourly rate of pay
6. Show the wage on the screen of the computer.

Algorithm in pseudocode as will be discussed in the chapters to follow:

```
CalcWage
~       This program will calculate and display the wage of an employee
        ~ Input (Provide values for the hours and the pay rate)
        display "Provide the hours worked"
        enter hours
        display "Provide the rate of pay per hour"
        enter payRate
        ~ Processing (Calculate the wage)
        wage = hours * payRate
        ~ Output (Display the wage)
        display "The pay is R", wage
end
```

Notes:

- Every algorithm has a descriptive name. This algorithm is called CalcWage.
- The last statement (step) is always *end*.
- ~ indicates that a comment that explains the code, will follow.
- Display means that a message or the contents of a variable is displayed on the screen. (Will be discussed later)
- Enter indicates that data must be entered by using the keyboard.
- The calculation is done by using an equation. In this algorithm *wage* will accept a new value i.e. the product of the hours and the hourly rate of pay.



Exercises

1. Write a detailed set of instructions, in sequence, to complete the following three tasks. Your instructions must be complete and in the correct sequence so that a novice can perform the tasks only by executing the instructions.

   1.1 Bake a cake
   1.2 Make a cup of tea
   1.3 Buy a book

2. Study the following problems, delete redundant information and decide if enough information is provided to obtain an answer. Write down the different steps to follow to solve the problem:

2.1 Jocelyn sold all but 7 of her hamsters for R5 each to a pet shop. The pet shop sells hamsters for R8.50 each. After the pet shop received Jocelyn's hamsters they have double the number of hamsters they had before the transaction. How many hamsters did Jocelyn sell to the pet shop?

2.2 A group of 30 learners, 2 lecturers and some parents took several cars to go to the 3 o'clock show of the circus. The price of an admission ticket is R25 per adult and R17.50 per learner. If each car held five people how many cars did they take?

2.3 Susan and her 3 friends went on vacation to visit her cousin who lives in a village at the sea. They had to travel for 2 days - 720 km on the first day and 485 km on the second day. While they were there, they visited friends in another town. They stayed with Susan's cousin for 5 days and it took them 2 days to return to their home. How many kilometres did they travel in total?

CHAPTER 2
INTRODUCTION TO VISUAL BASIC.NET (VISUAL STUDIO 2005)

2.1 INTRODUCTION

In chapter 1 you have learned about the problem solving approach and the basic concepts and tools needed to solve a computer related problem. Once you have completed an algorithm to solve a problem, the logic must be coded in a specific computer language, it must be compiled to convert it to machine language and the program must be executed to produce the required result.

There are many computer programming languages available. In this course we will be using Visual Basic.NET as a programming tool.

OUTCOMES

When you have studied this chapter you should be able to:

- understand the basic concepts in Visual Basic.NET, such as application, solution, project, file, user interface, controls on the interface and their properties, Code Editor Window versus the Designer Window,
- create a Visual Basic.NET Windows-based application,
- manage the windows in the IDE – Integrated Development Environment,
- set the properties of an object,
- use the group box, label, text box and button tools,
- add a control to the form,
- enter code in the Code Editor Window,
- save a solution,
- start and end an application,
- close and open an existing solution,
- understand the difference between a syntax error, a warning error message, a logic error and a run-time error.

2.1.1 VISUAL BASIC.NET

Microsoft Visual Basic.NET consists of a set of tools and technologies to assist in the easy development and creation of Windows and Web applications.

The user friendly programming environment, along with the relative simplicity of its programming language, allows individuals with little programming experience to create a wide range of programs in Visual Basic.NET.

2.1.2 AN APPLICATION

A Windows Application has a windows interface and is designed to run on a desktop computer in a MS Windows environment. The interface is what users see on the screen when they use the program and therefore it is also called the user interface. A web application runs from a web server and displays the user interface by using a web browser and includes web forms pages that have a web interface.

2.1.3 AN INTEGRATED DEVELOPMENT ENVIRONMENT

An integrated development environment (IDE) is an environment that contains all the features and tools needed to create, run and test your programs. It contains an editor to enter the program instructions and a compiler to run and test the program. Visual Studio 2005 is a Microsoft IDE. Included in Visual Studio 2005 are the Visual Basic 2005, Visual C++ 2005, Visual C# 2005 and Visual J# 2005 programming languages. Any of these languages can be used to create Windows-based or Web-based programs, referred to as applications.

2.1.4 TYPE OF PROGRAMMING LANGUAGE

In this course we will make use of the Visual Basic.NET computer language (VB.NET) to create windows applications. The windows operating system supports graphical user interfaces (GUI) and VB.NET provides a set of visual objects (controls or components) that can be drawn easily onto a window (called a form). These controls eliminate the need to develop the code to construct the visual interface. The layout of the user interface can easily be designed and created by dragging and dropping the controls to a position in the window. No code is needed to create the user interface.

Visual Basic.NET is an event-driven language rather than a procedure-oriented language. When a procedure-oriented program runs, it dictates the sequence of operations. In an event-driven program, the user can instruct the computer to perform whatever operation the program is capable of doing which offers flexibility to the user. However, the programmer will have to ensure that all the prerequisite actions have been done and if the user clicks on a button without entering the necessary data first, the program must take the necessary actions. The tasks to be performed in the case of an event are programmed using the same techniques

used in procedure-oriented programming and the code is contained within a procedure.

2.1.5 SOLUTIONS, PROJECTS AND FILES

As mentioned already, in this course we will create windows applications by means of VB.NET. Our applications will be very small as this is an introductory course and it will normally consist of only one project. However, one application could consist of various projects and files. A **solution** is a container to store the projects and files for an application. A **project** is also a container, but it stores files associated with one piece of the solution.

An analogy to illustrate these concepts is to think of a filing cabinet as illustrated in **Figure 2-1**. A solution could be similar to a drawer of the filing cabinet. A file is similar to a document that you store in a file folder in the drawer. There can be many projects in a solution just as there can be many file folders in a drawer of the cabinet. One file folder may also contain many documents. In a similar way, one project may consist of many files.

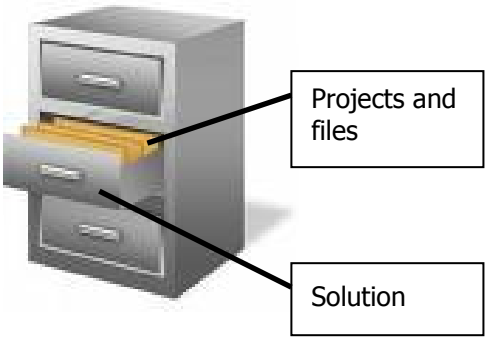
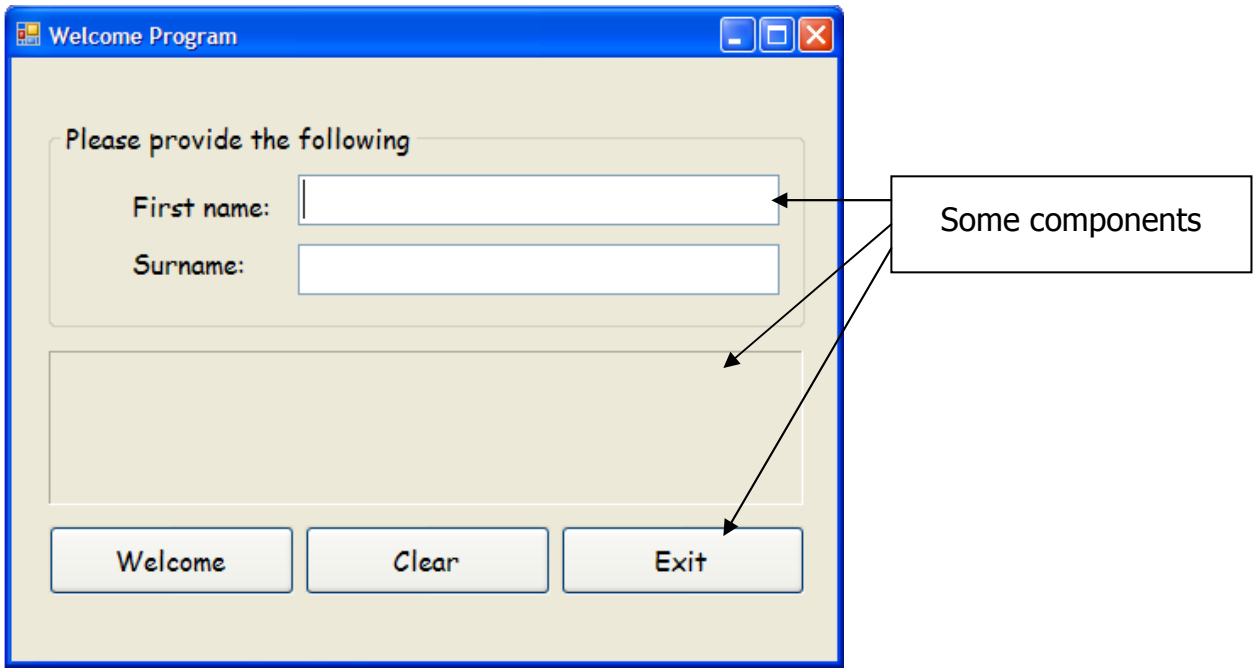


Figure 2-1

2.1.6 THE USER INTERFACE

Before we learn how to code the events, let's look at the user interface. As mentioned earlier, the user interface (sometimes referred to as a form) is what the user sees and the way with which he or she will interact with the program. Therefore it is important that it is user friendly and appealing to the eye.

For our first example, we are going to design the following form **(Figure 2-2)** through which the user can enter his or her first name and surname. The program must then display a message to welcome the specific user and expresses the wish that the user must enjoy learning Visual Basic.NET. The idea is to put every potential VB.NET learner at ease!



Any user interface consists of a number of components or controls and some of them are introduced in **Figure 2-2**.
Note: The purpose of some of the components on the form is there to display information while other components are there for the user to enter data.



Questions

- How many components can you identify on the form in **Figure 2-2**?
- How many of these components are there to enter data?
- How many components are there to display instructions and/or information to the user?
- Can you identify any other components that would not be used to enter data or to display information to the user?
- What do you think is the purpose of these components?

What will happen when the program is executed?

The user (a potential VB.NET learner) must provide his or her first name and surname. The user must then click on the Welcome button to invoke a procedure to display a personalized welcome message on the label as indicated in **Figure 2-3**.

Example:

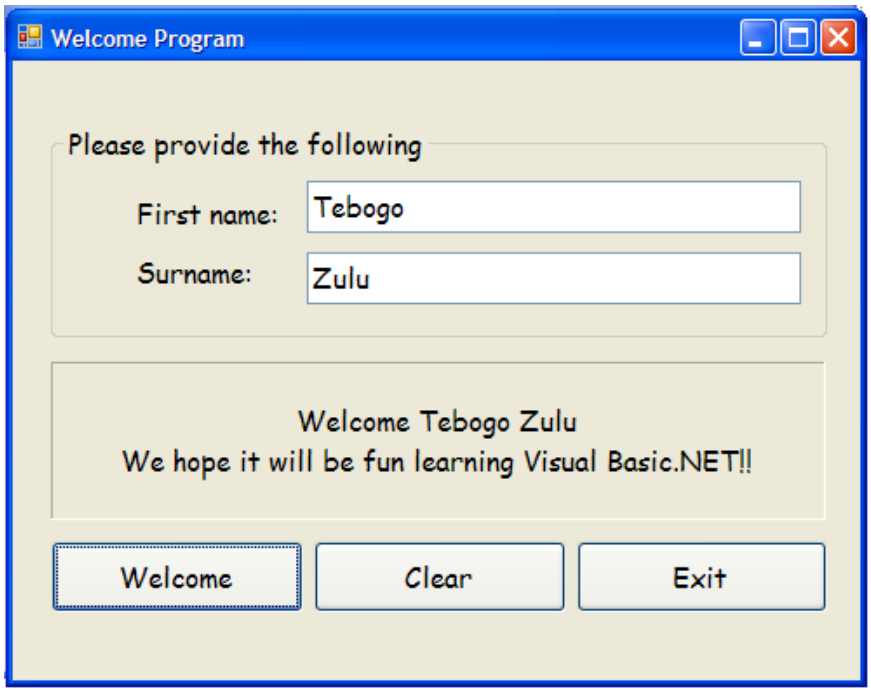


Figure 2-3

When designing a user interface, there are many guidelines to adhere to in order to make the form as user friendly as possible. In later chapters we will focus on individual controls and guidelines for each. At this stage, we can take note of the following guidelines:

- Align the borders of all controls and try to make similar components exactly the same size (width and height).
- Graphics and color should be used sparingly. If you use a graphic, use a small one and place it at a location where it will not distract the user. Limit the number of colors to three.
- Use a font that is easily readable. Keep the font and the size consistent.
- The information on a form should flow either horizontally or vertically, with the most important information at the top left corner.
- Keep margins consistent.
- Related controls are typically placed close together on the form.

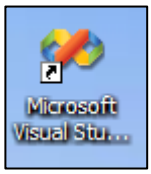
2.2 EXPLORING THE IDE (INTEGRATED DEVELOPMENT ENVIRONMENT) AND THE FORM BY DEVELOPING A SMALL APPLICATION

From the above mentioned example we can see that a programmer must first design the way the user will interact with the program. The programmer must then use the IDE to visually design this user interface.

Let's go through all the steps that the programmer must follow when the user interface in **Figure 2-2** must be designed and to add the necessary code to produce the output in **Figure 2-3**.

STEPS TO FOLLOW TO CREATE A VISUAL BASIC 2005 WINDOWS BASED APPLICATION:

Start VB.NET
Double click on the following icon on the desktop



or
Click on the start menu, select Programs and Microsoft Visual Studio 2005.

A **new application** must then be created by clicking on File and choosing **New Project** as indicated in **Figure 2-4**.

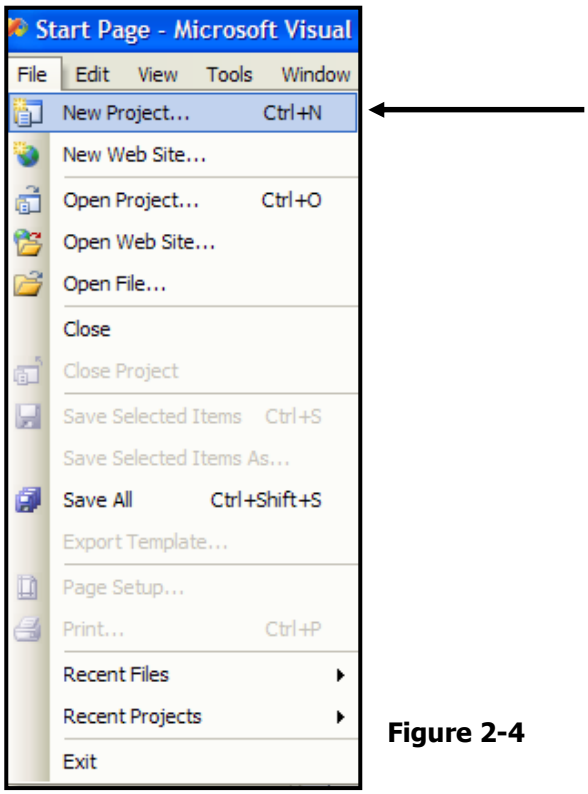


Figure 2-4

The following screen (in **Figure 2-5**) will appear. Note that the project type must be Visual Basic. (Remember the Visual Studio 2005 IDE also provides for Visual C++ 2005, Visual J# 2005 and Visual C# 2005). If the project type is not Visual Basic, position the cursor on Visual Basic and right click on it.

The default name for a new application is WindowsApplication1 (or WindowsApplication2, 3 etc. depending on previous attempts during the same session). As our first program is only going to ask a student for his/her name and is then going to welcome the student, we are going to change the name of the application to be **WelcomeProgram**, which is a more meaningful name to describe the project and to find it when trying to retrieve it at a later stage.

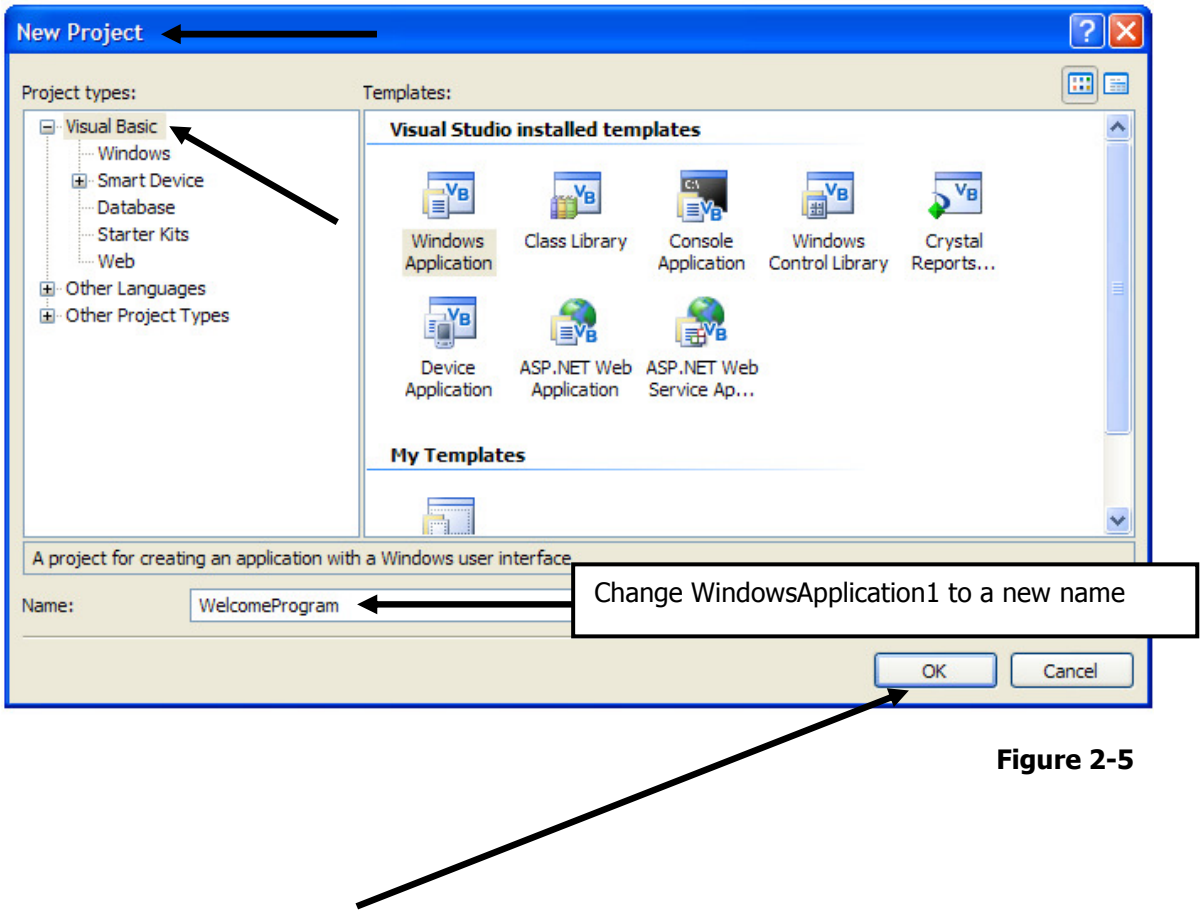


Figure 2-5

When you click on the OK button, a new form (Form1.vb) will appear in the Windows Form Designer Window. Two other windows, the Solution Explorer and Properties window, should also be open. Should one or both of these windows not be open, click on VIEW and select the required window.

The solution explorer displays a list of the projects in the current solution. It also contains the items contained in each project.

Our solution contains one project folder and a file named Form1.vb

It is good practice to give each form file a more meaningful name. Right-click on Form1.vb and select Rename as indicated in **Figure 2-6**.

According to the Hungarian naming convention, the new name **must** start with the prefix frm.

The extension of the new name **must** be .vb.

Let's choose the new name
frmWelcome.vb

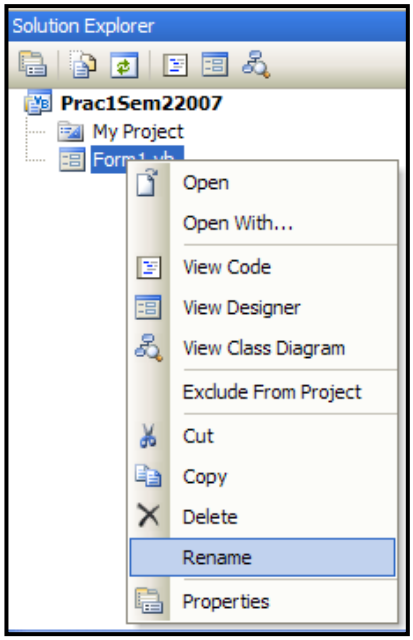


Figure 2-6

After the form has been renamed, the new name will automatically appear in the object box of the properties window as well as in the Windows Form Design window (See **Figure 2-7**):

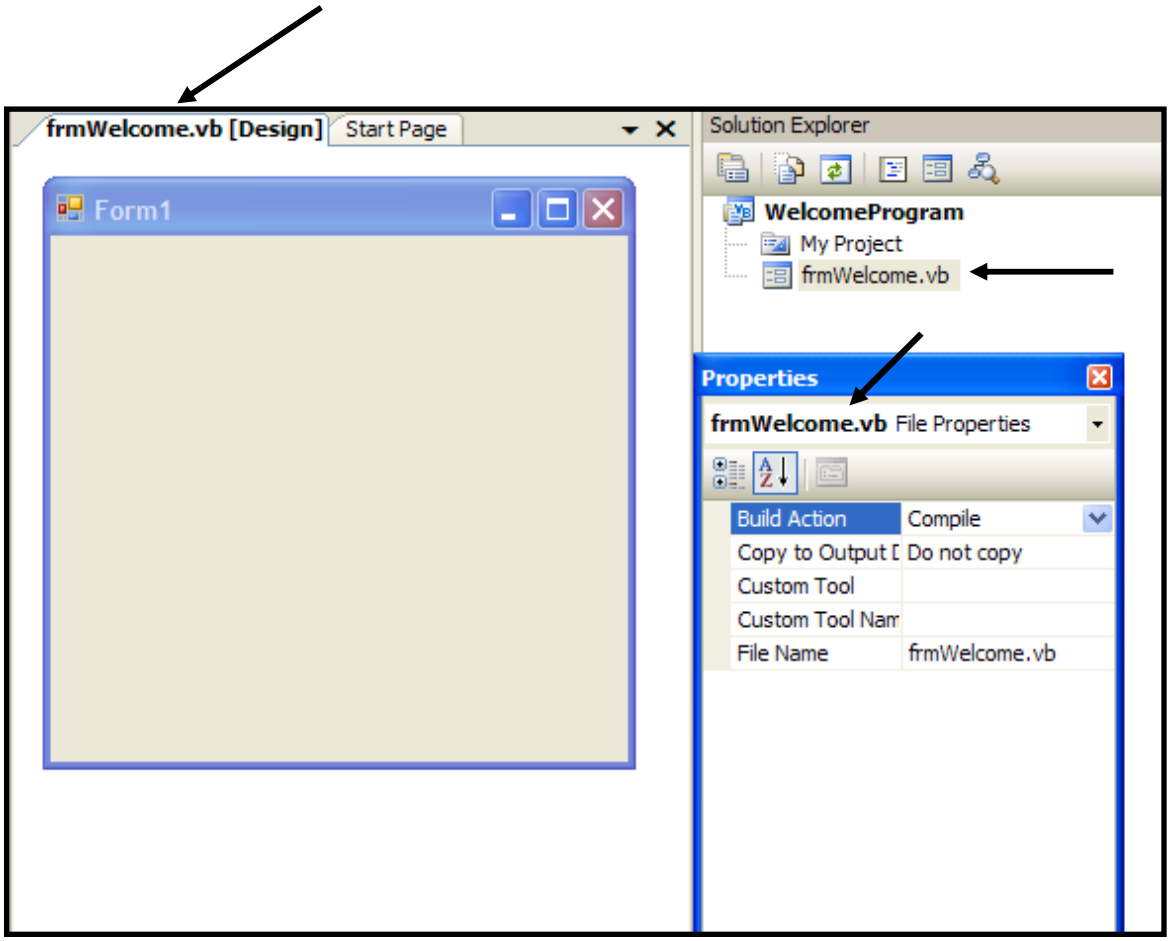


Figure 2-7

A form is an object. All objects have certain characteristics. For instance, a form has a certain colour, height and width. These are just some of the characteristics or properties of the form. Each property has a certain value. The values can be selected and changed in the properties window.

To **set the properties of a form**, the programmer must first select the form by clicking on it (See example in **Figure 2-8**). The properties will then change to reflect all possible properties that can be selected for the form.

There are many properties applicable to the form and the programmer can scroll up and down to select and change various properties according to the specific characteristics that the form must have that the programmer wants to design.

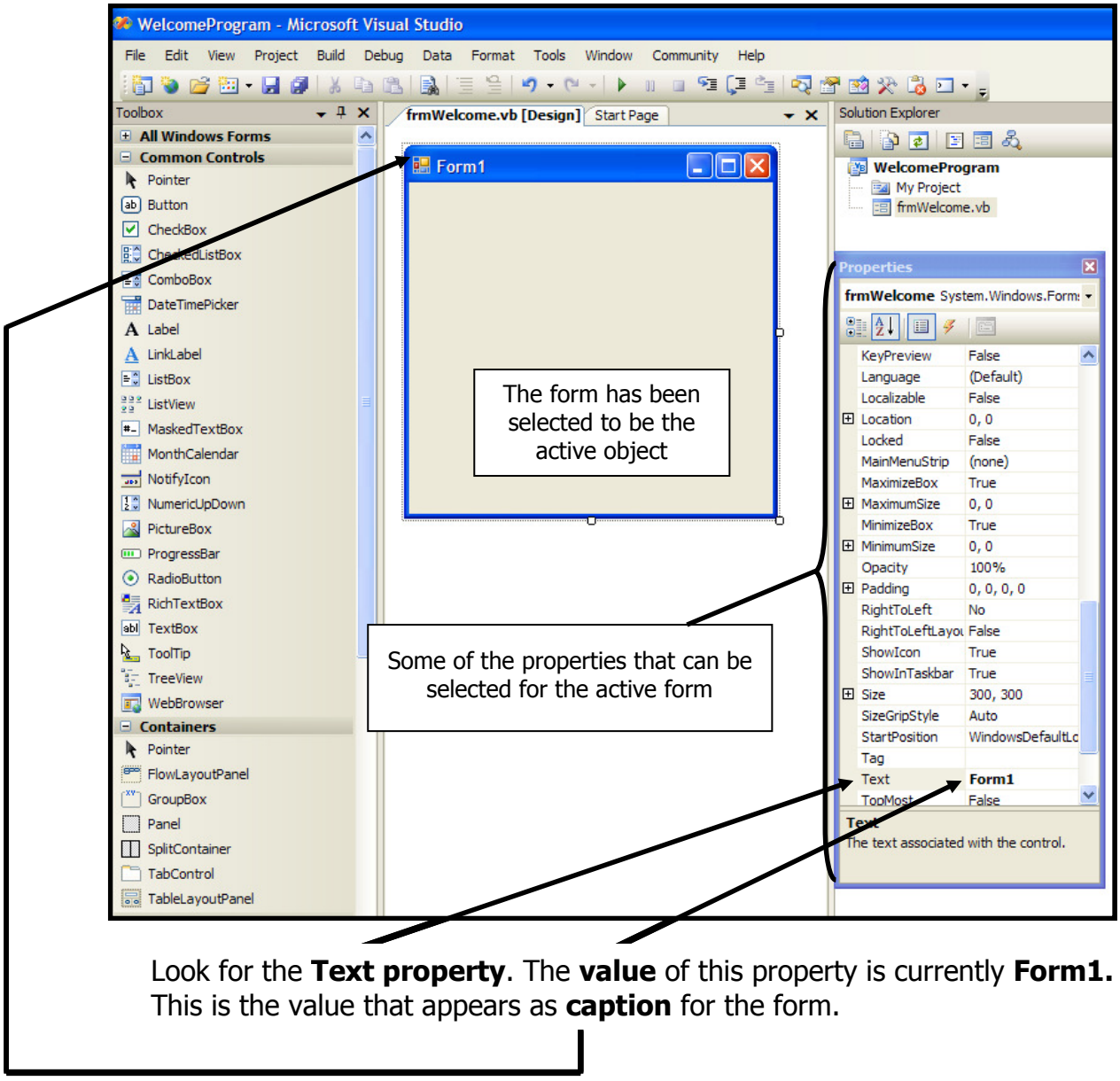


Figure 2-8

Look for the **Text property**. The **value** of this property is currently **Form1.** This is the value that appears as **caption** for the form.

If we want to change the caption on the form, we must change the value of the text property of the form. If we change the name to Welcome Program, the caption of the form will automatically change to this new value.

In a similar way we can change the font from Microsoft Sans Serif, 8.25p (the current default font and size) to Comic Sans, 12p. This will then be the default font and size applicable to all components that will be placed on the form at a later stage. It will also have an effect on the size of the form because the value of the AutoScaleMode property is Font. (See examples in **Figure 2-9 to 2-13**).

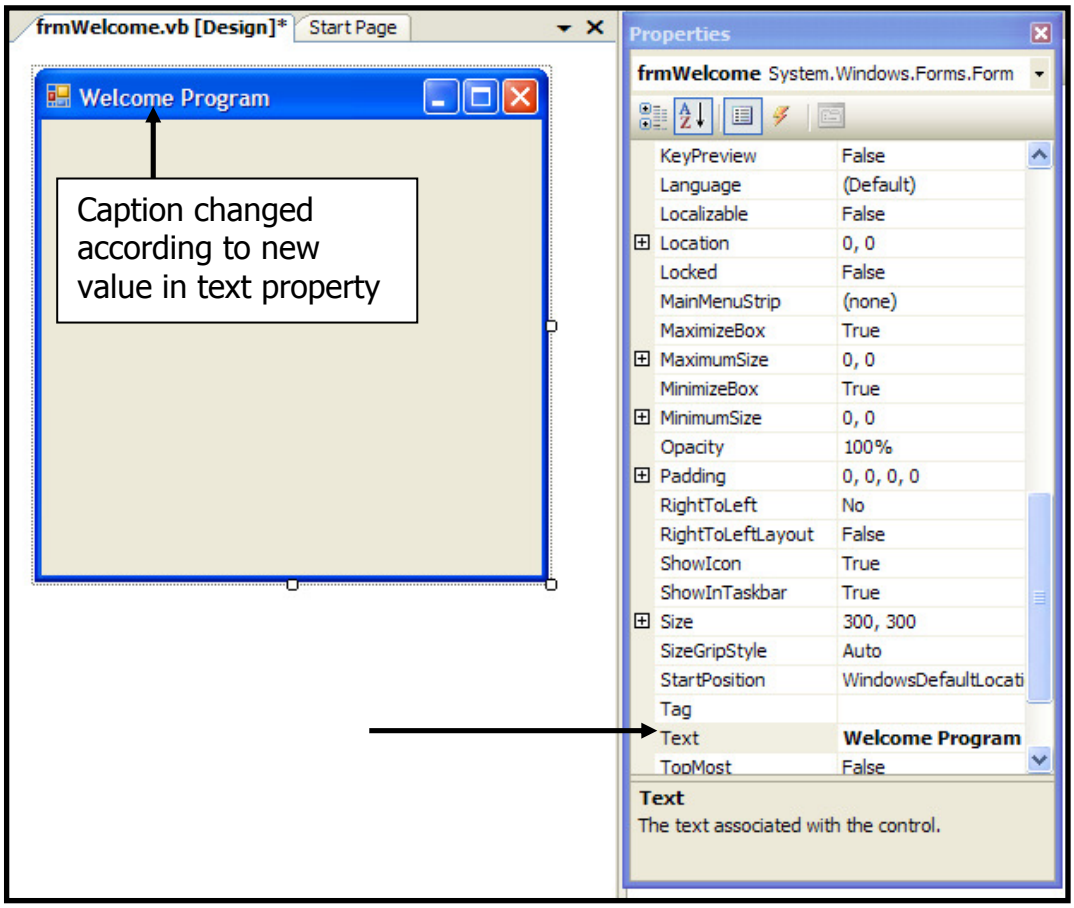


Figure 2-9

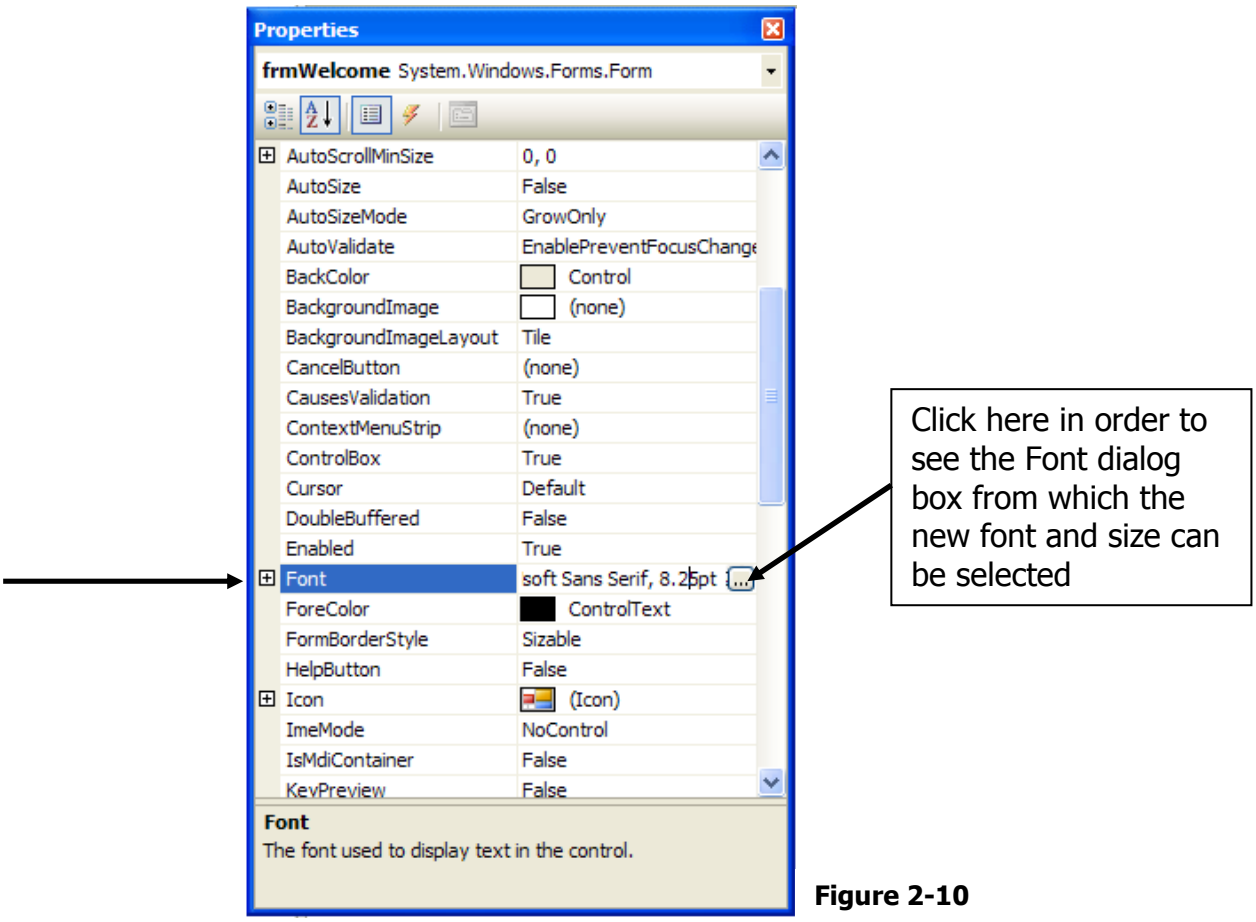


Figure 2-10

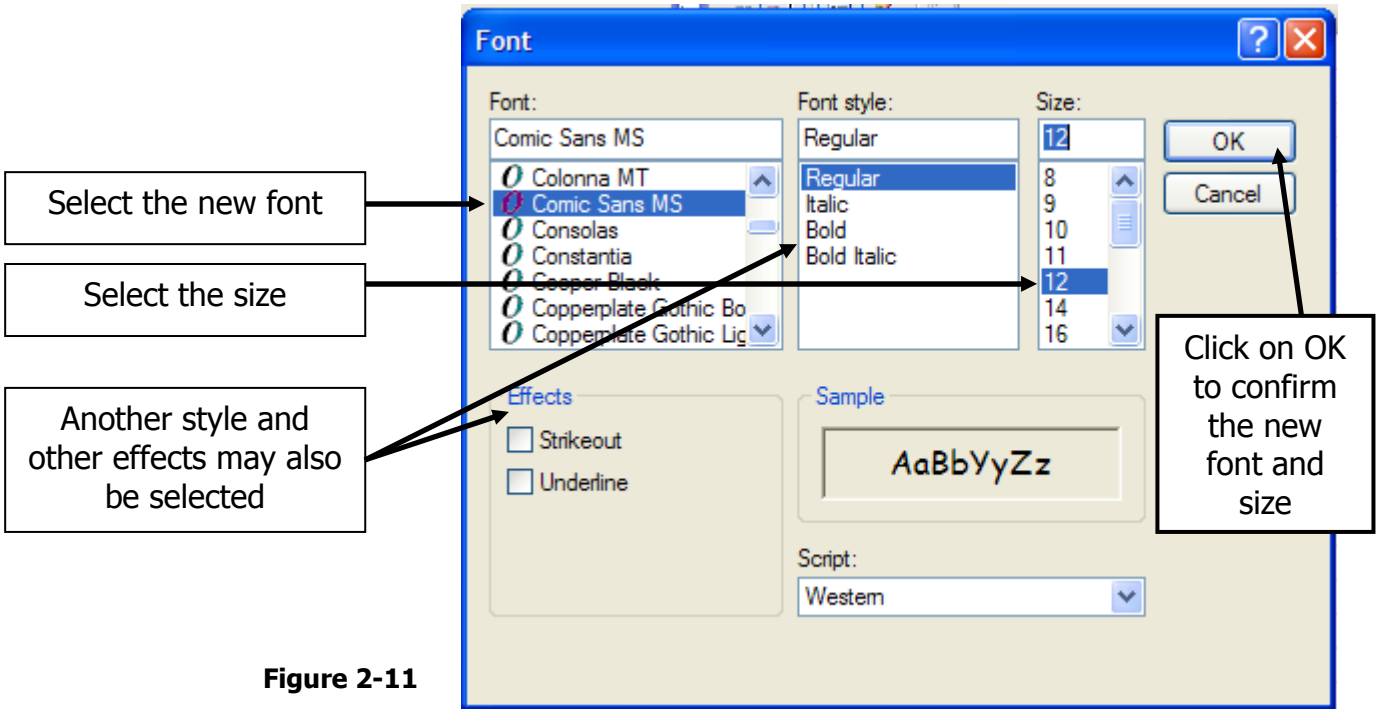


Figure 2-11

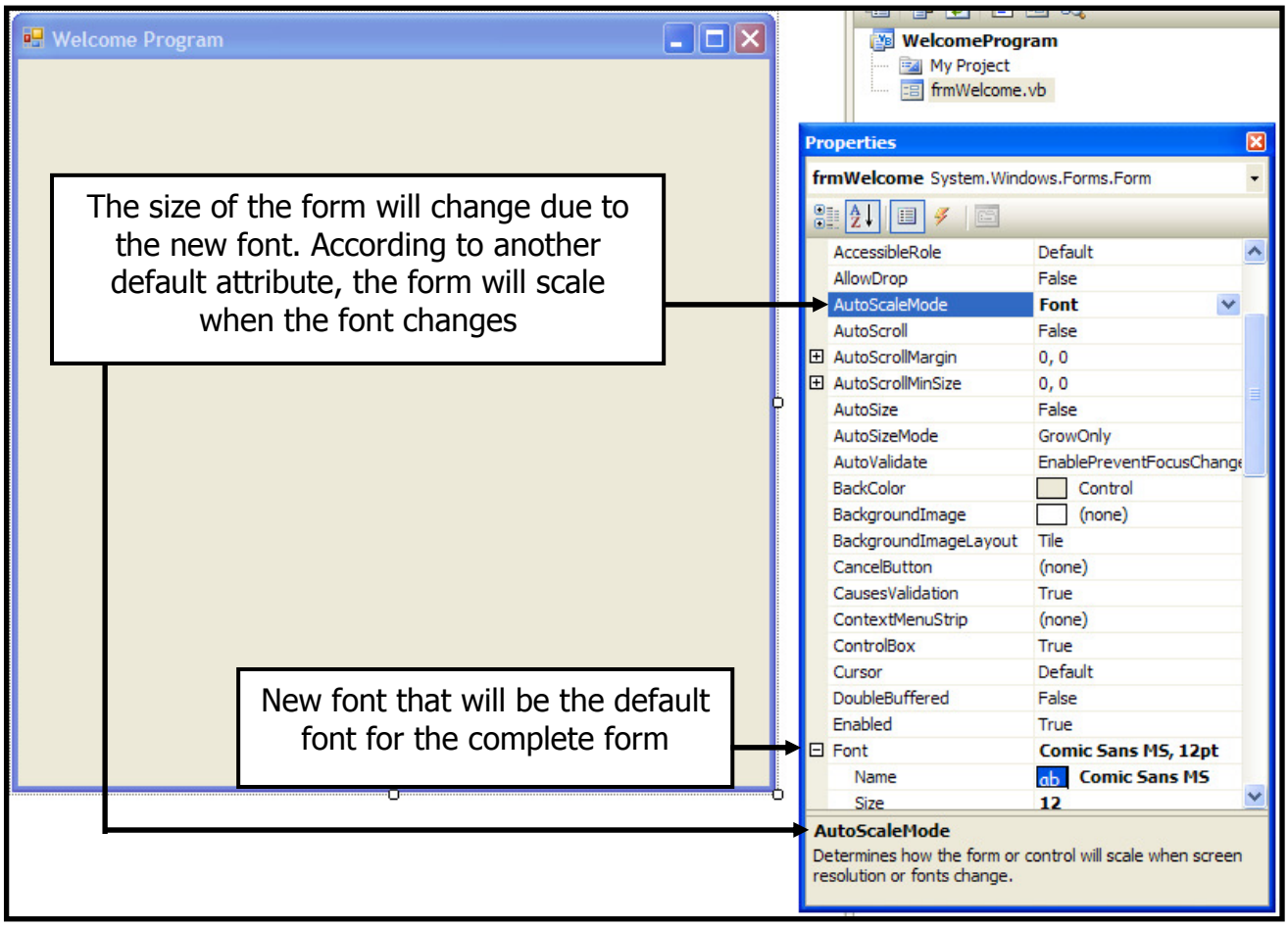


Figure 2-12

The best way to learn is to explore and to try various options. Go to different properties of the form and change it to see what the effect of the change will be.

A few suggestions:
Change the BackColor, place a certain image as background on the form via BackgroundImage, then click on + next to the size property to change the height and the width of the form.

You will also see that, when a property is selected, a description of the purpose of the property is provided as a help-facility.

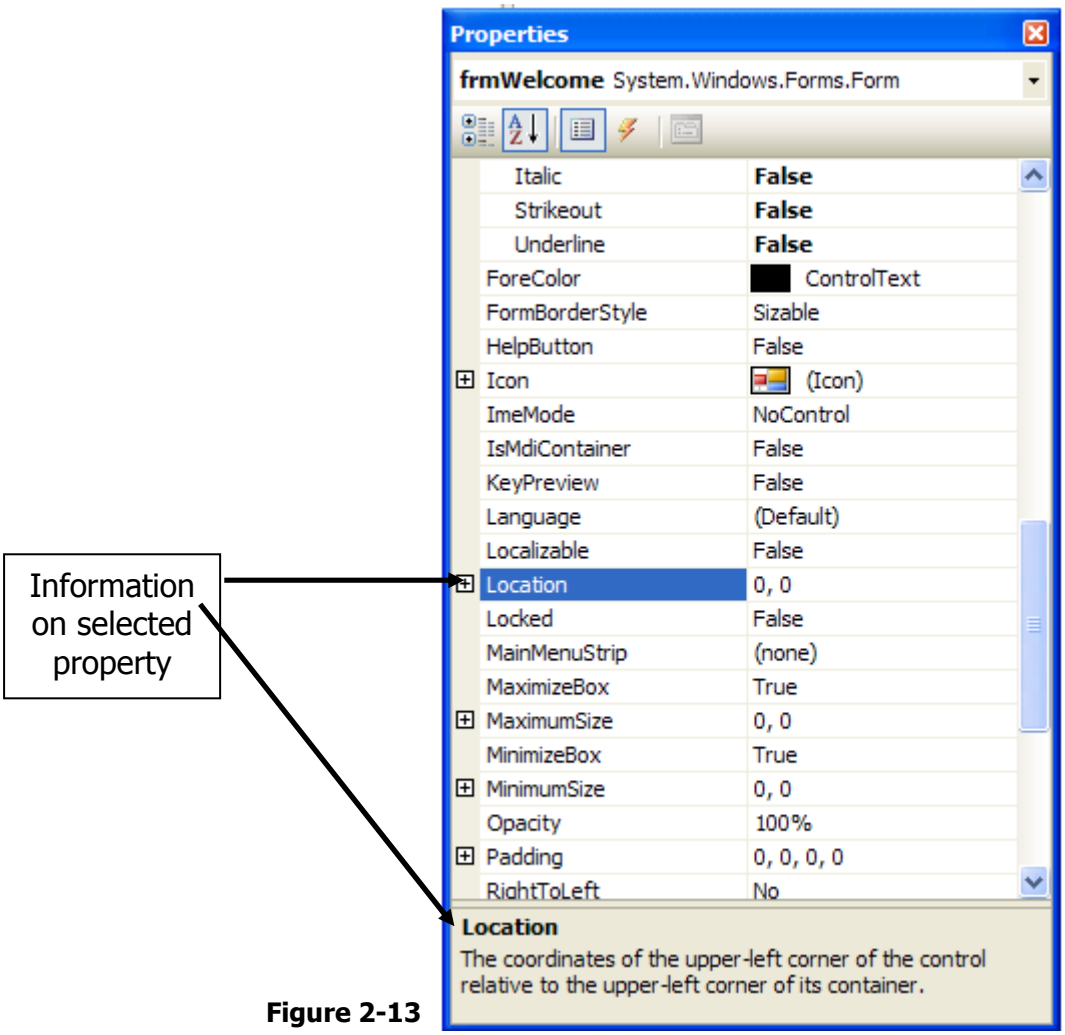


Figure 2-13

Placing components on a form:

Once the default properties for the form have been selected, the necessary components can be placed on the form. All the components can be found in the toolbox window. Remember if any part of the IDE (such as the properties window or toolbox window) is not visible, it can be activated via the view command.

In a similar way that we learned about a form, the best way to learn about a component is to select a component and place it on the form and then to explore by playing around with the different properties of that component. For our welcome program we will use labels, text boxes and buttons. We will also use a group box to group some of the components. In later chapters we will start exploring radio buttons, check boxes and list boxes. ***The purpose of the course is not to focus on the user interface and to teach you all the components but to focus on the programming logic.*** You will therefore not be taught about all the components, but you are free to experiment and play with them. Remember, to explore is the best way to learn about it.

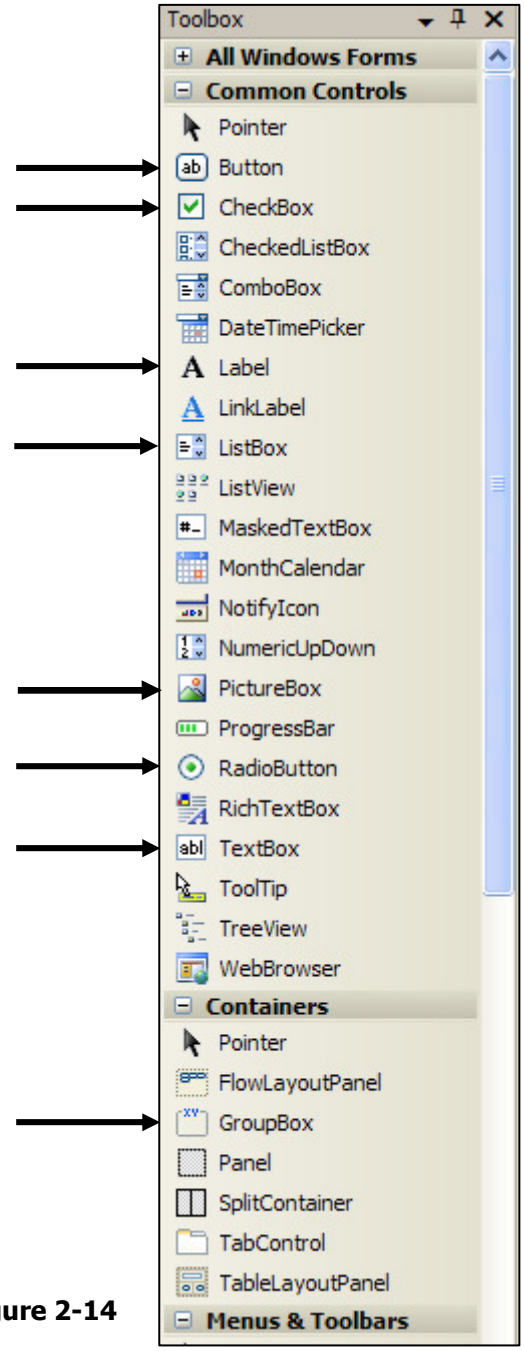


Figure 2-14

To place a component on the form, simply click on the component in the toolbox and drag it to the position on the form where you want to place it.

You may also double click on the component in the toolbox in which case it will be placed in the top left corner of the form. You may then drag is from there to the position where you want to place it.

To remove a component from the screen simply click on it and the press <delete>

Any component can be resized by clicking and dragging the handles.

The components that we will use during this course are indicated in **Figure 2-14**.

The purpose of a **label** is to provide information to a user – e.g. to display an instruction or to display an answer to the user.

The purpose of a **text box** is to provide a space where the user can enter an input value. A text box can also be used to display an answer but be careful because in such a case the user may alter the value, unless the enabled property is set to FALSE.

The purpose of a **button** is to provide a place where the user can click if he wants to activate a certain event e.g. to instruct the computer to calculate a certain answer.

Remember, at this point we just want to create the user interface in **Figure 2-2** and to add the code in order to obtain the output in **Figure 2-3**. We will discuss all the relevant objects in detail in later chapters. When each object is discussed we will also provide additional design rules to adhere to when these components are placed on a form.

Now, let's design our user interface:

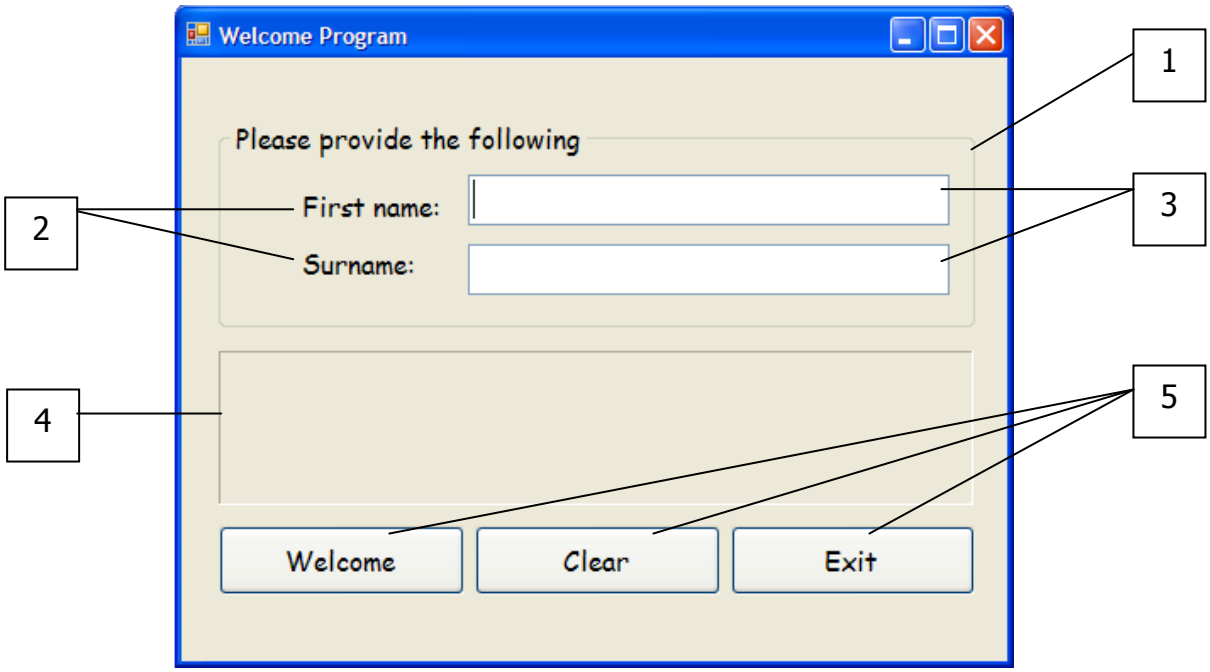


Figure 2-15

Step 1:

Place a **group box** on the form. Name the group box **grpEnter**. Change the text to the value "**Please provide the following"** and change this text in order for it to be displayed in black.

Step 2:
Place two labels on the group box. Name the first label **lblFirstName** and the second **lblSurname**. Change the text to "**First name:"** and "**Surname:"** respectively and ensure that the two labels are aligned.

Step 3:
Place a text box next to each label on the group box and ensure that they are aligned. Name the first text box **txtFirstName** and the second one **txtSurname**.

Step 4:
Add a label with a border beneath the group box. Name this label **lblWelcomeMessage** and align it with the borders of the group box by setting the **AutoSize** and **BorderStyle** properties of the label. Ensure that the message will be displayed in the centre of the label by setting the **TextAlign** property to MiddleCentre.

Step 5:
Place the following 3 buttons underneath the label (lblWelcomeMessage) and ensure that they are exactly of the same size and aligned.

Name:	btnWelcome	**Text:**	Welcome
	btnClear		Clear
	btnExit		Exit

Ensure that btnWelcome is the default button and that Exit is the cancel button. (Make the form the active object and set the **AcceptButton** and **CancelButton** properties respectively).

Step 6
Save the form by pressing the **Save All** icon or click on File and select Save All. This step must be done at regular intervals throughout any practical session and not only at this point.



When the application is saved for the first time, the following screen will appear in order for you to enter the directory where the application must be saved.

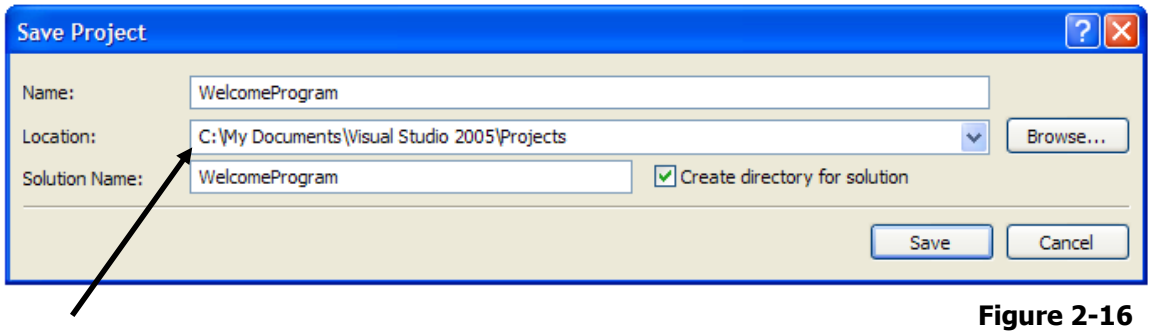


Figure 2-16

Ensure the correct folder location in your laboratory or personal computer. The application may also directly be saved to your flash disk drive. However, to develop, test and run an application directly from the flash disk could be very slow. Therefore we would rather recommend that you work on the hard drive and copy your complete application to the flash disk when you leave or when you have finished.

Adding working code to the application:

We are now going to add code in the code editor in order for the program to welcome the user in the following way:

The user must first enter his or her first name and surname and then press the Welcome button (e.g. Tebogo Zulu). You may assume that the user will enter both

fields before clicking on the Welcome button. The program must then welcome the person with the following message:

Welcome Tebogo Zulu
We hope it will be fun learning Visual Basic.NET!!

When the user clicks on the clear button, both text boxes as well as the label must be cleared in order to give another user a chance to enter his or her name and to be welcomed in the same way. When the user clicks on the exit button, the program must terminate.

Because the Welcome button and the Exit button have been activated as the default button and exit button respectively, it will also be possible for the user to enter both input fields and then press only the enter-key (instead of clicking on the Welcome button) in order to see the welcome message. In a similar way will it be possible to exit the program by pressing the escape key on the keyboard instead of clicking on the Exit button.

To enter code for the Welcome button:

Double click on the Welcome button. Control will be transferred to the code editor and a procedure header and procedure footer will automatically be generated for this click event.

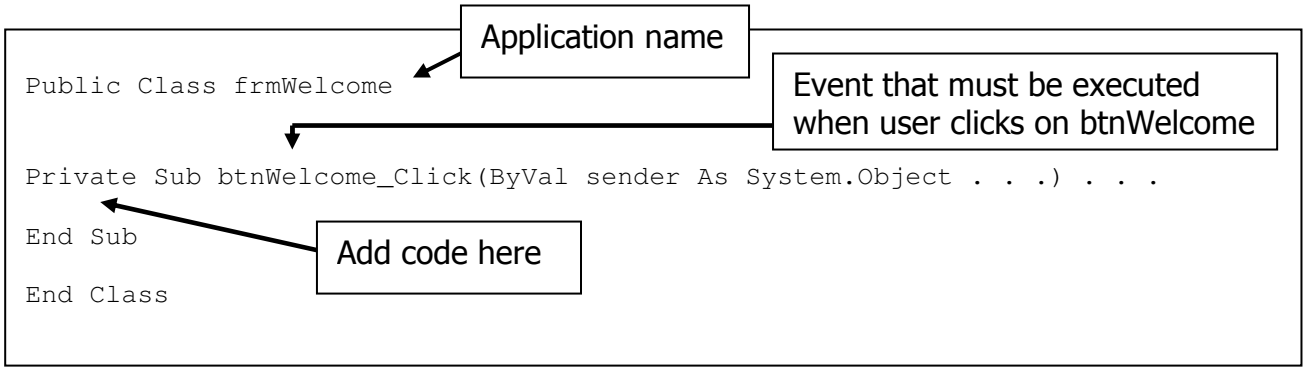


Figure 2-17

Add the following code in between the Header and the Footer that automatically appeared.

```
lblWelcomeMessage.Text = "Welcome " & txtFirstName.Text & " " & _
                  txtSurname.Text & ControlChars.NewLine & _
                  "We hope it will be fun learning Visual Basic.NET!!"
```

The & operator is a concatenation operator that will combine string items together. The word Welcome is combined with the names that the user entered in the two text boxes. A space is concatenated between the first name and surname.
The underscore character (_) indicates to the compiler that one statement continues on a next line.
ControlChars.Newline is a constant that will instruct the computer to place the next string on a new line on the label.

These operators and constants will be fully discussed in chapter 5, but at this point we are going to use them in order to run a simple application and to display output on a label on the screen.

After entering these two lines of code, the entries in the code editor look as follows:

```
Public Class frmWelcome

    Private Sub btnWelcome_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handl
        lblWelcomeMessage.Text = "Welcome " & txtFirstName.Text & " " & _
                          txtSurname.Text & ControlChars.NewLine & _
                          "We hope it will be fun learning Visual Basic.NET!!"
    End Sub
```

Figure 2-18

In a similar way, add the following code for the btnClear click event to clear the screen and to restore the focus to the text box where the user must enter the first name:

```
txtFirstName.Text = ""
txtSurname.Text = ""
lblWelcomeMessage.Text = ""
txtFirstName.Focus()
```

The btnExit click event must contain the statement Close() in order to terminate the application.

When all the code has been entered, the code editor will look as follows:

```
Public Class frmWelcome

    Private Sub btnWelcome_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handl
        lblWelcomeMessage.Text = "Welcome " & txtFirstName.Text & " " & _
                         txtSurname.Text & ControlChars.NewLine & _
                         "We hope it will be fun learning Visual Basic.NET!!"
    End Sub

    Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
        txtFirstName.Text = ""
        txtSurname.Text = ""
        lblWelcomeMessage.Text = ""
        txtFirstName.Focus()
    End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
        Close()
    End Sub

End Class
```

Figure 2-19

Now let's run the program by clicking on the debug icon as indicated in **Figure 2-20**. Assume my name is Tebogo and my surname is Zulu:

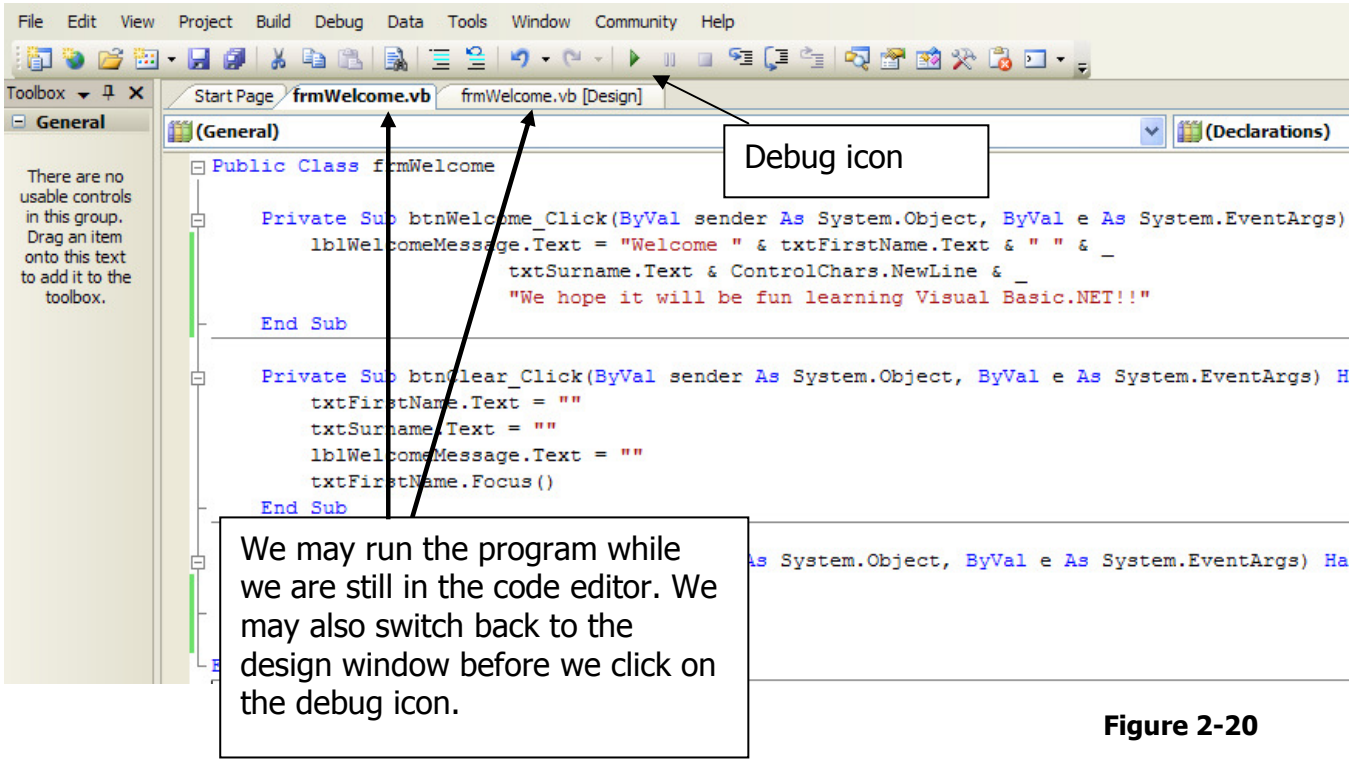


Figure 2-20

The user interface will look as follows after the user entered the values Tebogo and Zulu and then clicked on the Welcome button:

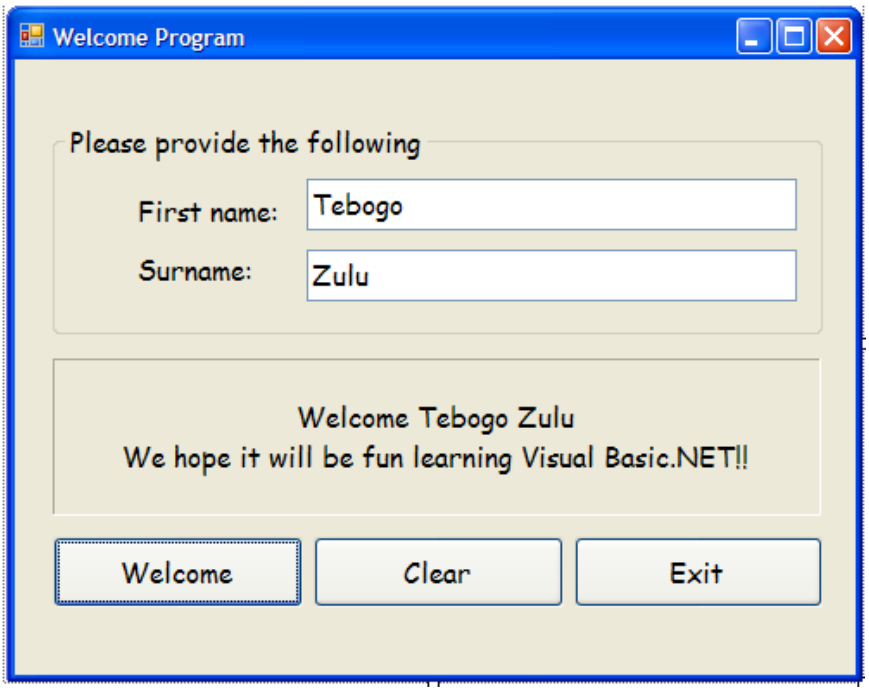
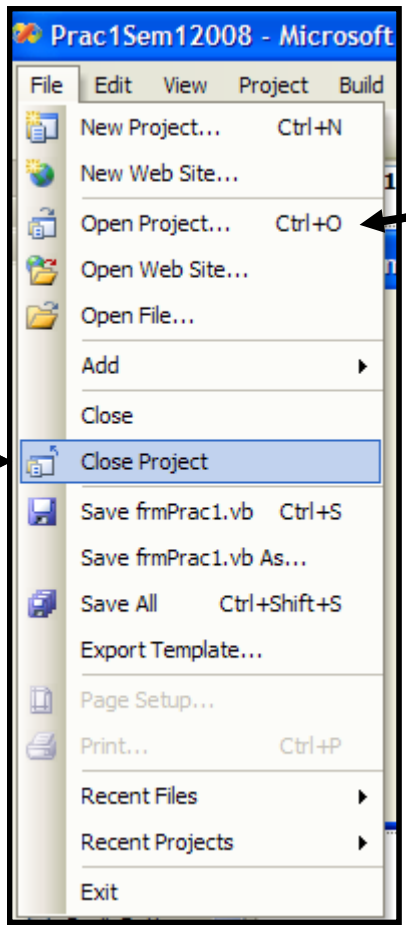


Figure 2-21



Close the application. Ensure to close the complete application or project.

To open an existing application at a later stage the option to **open a project** must be chosen, not the option to open a file. Then double click on the name of the project (folder) in the correct directory.

Figure 2-22

TO SAVE AN APPLICATION OR PROJECT ON A FLASH DISK

Look in the correct directory (folder) where the project is currently stored. Right click on the application. Send the complete folder to the Flash-drive (In this case, drive I, However it might even be E or F).

Do not use the save as option.

All the applicable parts of the project must remain in the folder.

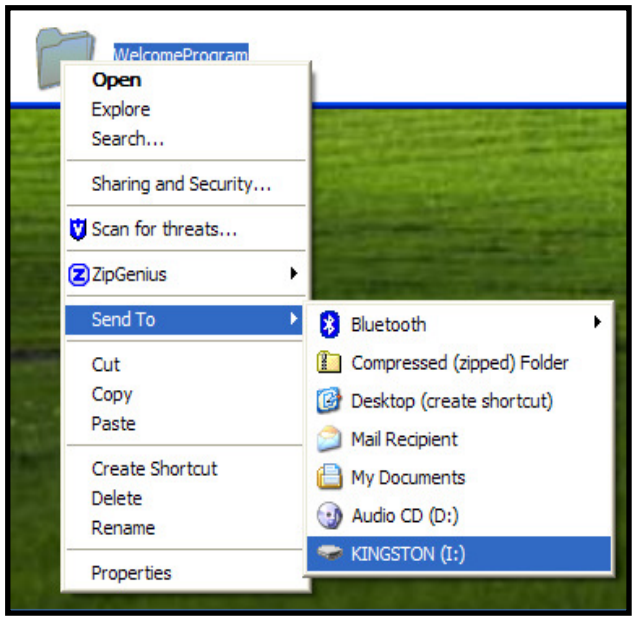


Figure 2-23

It is extremely important that these steps must always be followed in the correct way. Do not try to save the code in the code editor separately from the user interface in the design window. Remember all the files belong to the same project of the same solution and it must be saved in the same folder.

The purpose of this chapter was just to introduce you to the IDE and to get you going with a simple application. The program did not contain any variables and we will deal with that as we progress. We also assumed that the user has entered valid values before the Welcome-button was clicked. We will also deal with data validation in later chapters.

2.3 ERRORS

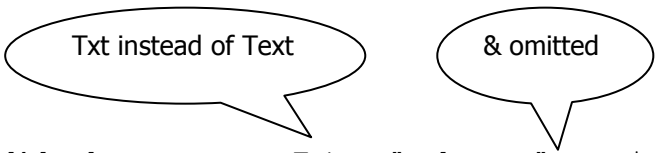
2.3.1 SYNTAX ERRORS

A syntax error is an error that occurs when a mistake has been made in the syntax of the language. In other words a rule of the language has not been followed.

Most syntax errors are typing errors made when entering the code into the code editor. Sometimes essential characters (such as a concatenation character or line continuation character) are omitted, in which case an error will also occur.

Syntax errors will be underlined in blue and the corresponding error message will be indicated in the error list window.

Suppose, in our example, we make the following mistakes when entering the statement to place the welcome message on the label.




```
lblWelcomeMessage.Txt = "Welcome " txtFirstName.Text & " " & _
        txtSurname.Text & ControlChars.NewLine & _
        "We hope it will be fun learning Visual Basic.NET!!"
```

Figure 2-24 indicates how the two errors will be underlined in the code editor and **Figure 2-25** indicates the corresponding error messages in the error list window.

```
Public Class frmWelcome

    Private Sub btnWelcome_Click(ByVal sender As System.Object, ByVal e As Syster
        lblWelcomeMessage.Txt = "Welcome " txtFirstName.Text & " " & _
                                txtSurname.Text & ControlChars.NewLine & _
                                "We hope it will be fun learning Visual Basic.NET!!"
    End Sub
```

Figure 2-24

Error List

2 Errors | 0 Warnings | 0 Messages

	Description	File	Line	Column	Project
1	'Txt' is not a member of 'System.Windows.Forms.Label'.	frmWelcome.vb	4	9	WelcomeProgram
2	End of statement expected.	frmWelcome.vb	4	44	WelcomeProgram

Figure 2-25

A syntax error is identified by means of a red circle with a cross in the middle. If you double click on the syntax error, it will take you to that specific line and error in the code editor, in order to correct is.

No program will run if there are any syntax errors present and when you try to run it, the following message will appear, as indicated in **Figure 2-26**.

Click on the No-button and correct all syntax errors before you try to run the program again.

Apart from the error message and the blue underlining of the error, the line and column where the syntax error appears are also indicated and should also be of help when trying to find all your syntax errors.

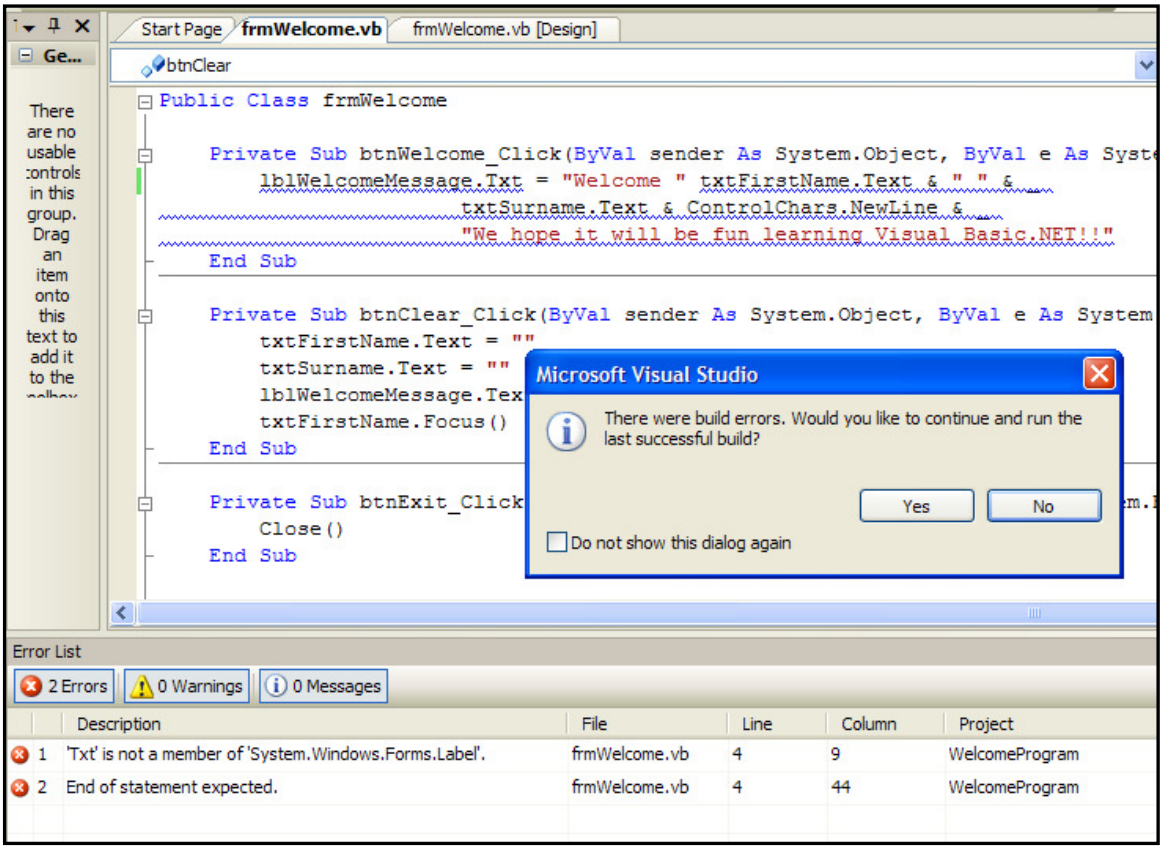


Figure 2-26

2.3.2 WARNING ERROR MESSAGES



Sometimes the code will contain warning error messages. These messages will indicate an error which is not so serious that it will prevent the program from running and giving output.

In the following example we have declared two variables respectively to hold the first name and surname entered. However, we never store the values entered in the text boxes in these variables and we use the text property of the text boxes directly in the statement to display the output welcome message on the label.

The program will warn us that we have declared variables that have never been used as indicated in **Figure 2-27**. However, it will still produce output if we click on the debug icon.

All warning error messages are indicated with an exclamation icon and the error will be underlined in green.

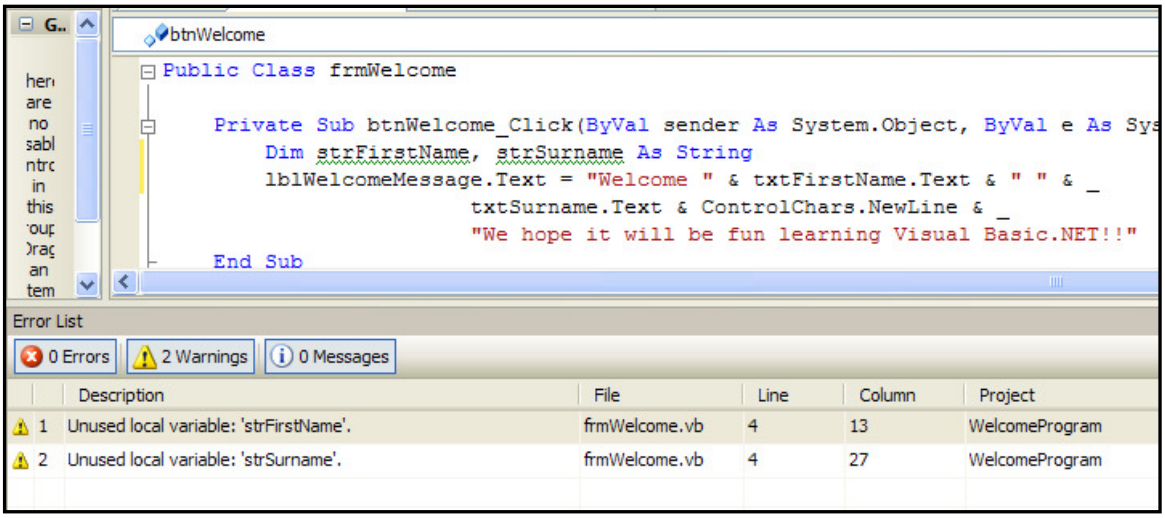


Figure 2-27

2.3.3 LOGIC ERRORS

A logic error will occur if the programming code does not produce the correct output as expected.

One of the instructions could be wrong. For example, if the programmer wants to calculate an average by means of the following statement, it will produce the wrong answer because division has a higher precedence number than addition.

```
decAverage = decNum1 + decNum2 / 2
```

If decNum1 has a value of 12.8 and decNum2 has a value of 64.8, the above statement will yield an answer of 45.2 instead of 38.8 because the number 64.8 has been divided by 2 before 12.8 has been added to it.

The program will therefore calculate and produce an answer, but the wrong answer. In order to rectify the error, brackets can be inserted to add the two numbers and then to divide the answer by 2.

```
decAverage = (decNum1 + decNum2) / 2
```

Calculations and arithmetic operators will be dealt with in chapter 3.

2.3.4 RUN TIME ERRORS

Good programs will always validate all input values entered because a wrong input value or no input value at all could cause that the program ends abruptly if calculations cannot be performed (i.e. if a character instead of a numeric value has been entered). We will deal with validation matters in chapter 6.



1. Enter all the steps as discussed in this example during the practical sessions and run the program a few times with different names in each case. Ensure you understand all steps followed.

2. Design a user interface to look exactly as the one in **Figure 2-28**. The user must be able to enter his or her name and the color and make of his/her car. The input values must then be combined into a sentence and must be displayed on the label in the following way. Provide for a default and cancel button. Once again you may assume that the user will enter valid values before the display button is clicked. This is also a very considerate programmer because, based on the message, everybody will be smart, no matter what they drive!

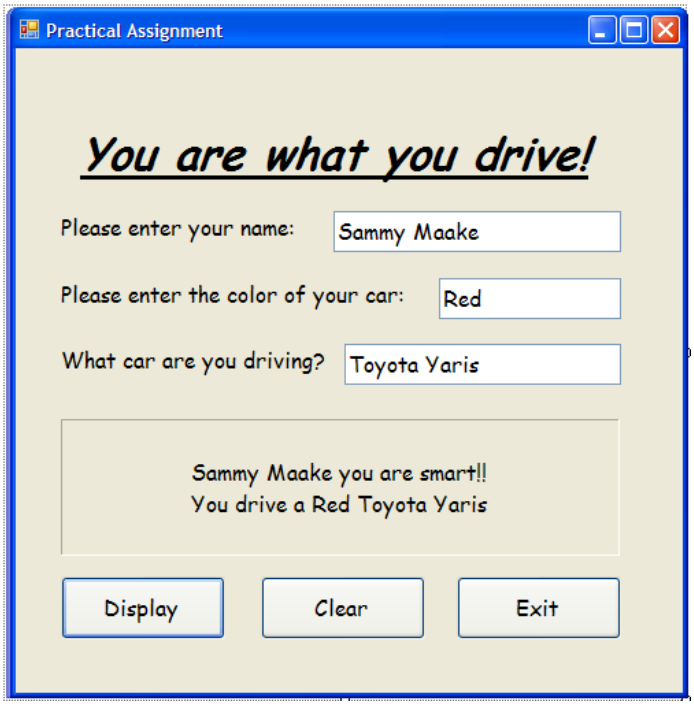


Figure 2-28

CHAPTER 3
GENERAL CONCEPTS AND ARITHMETIC

3.1 INTRODUCTION

In this chapter the beginner programmer needs to learn the basic concepts of the components of an algorithm and a computer program. In Chapter 1 we have seen that the first phase of the data processing process is the **input** to the algorithm or program. As discussed, it is also called data. The input or data must be **processed** to produce useful **output** or information. Input data entered will normally be placed in variables in memory. During the processing steps the program will then make use of these input variables to produce the required output results. The processing could also involve constant values, arithmetic calculations and certain programming structures, namely sequence, selection and iteration.

This chapter will deal with the concepts of variables and constants, as well as arithmetic.

OUTCOMES

When you have studied this chapter you should be able to:

- understand what a variable is,
- distinguish between the different data types,
- know how to declare a variable in Visual Basic.NET,
- distinguish between a variable with module scope and one with procedure scope,
- do conversions between different data types,
- know how to insert the OPTION EXPLICIT ON and OPTION STRICT ON into a program and know why it should be done,
- distinguish between a variable and a constant,
- know how to write an assignment statement,
- know how to format numeric output,
- know the difference between a format specifier and a precision specifier,
- understand basic arithmetic operations,
- know and use all arithmetic operators,
- set up and evaluate expressions and equations using variables, constants, operators and the hierarchy of operations.

3.2 VARIABLES

The programmer refers to a position or location in the memory of the computer as a variable where values can be stored. Initially, the variable need not contain a value. The word *variable* indicates that the value of the variable may vary as needed or as processing is done. You can visualise a variable as a box in memory that contains one, and only one, specific value at a given time.

An illustration is a scoreboard of a football match where the scores of the two teams are displayed. A team has only one score at a given time e.g. at the beginning the score will be zero, but after the first goal it will be one. At no time may the scoreboard contain 2 different scores for one specific team.

The same rule applies to a variable. After the statements of the algorithm are processed, the previous value of the variable is replaced by a new value. It is not possible to retain the previous value in the same variable.

3.2.1 NAME OF A VARIABLE

Every variable is given a descriptive name. There is no need for the programmer to know the exact position (address) of the variable in the memory of the computer, as the compiler will link the given name of the variable to the actual address in memory. Only one name is given to a variable and this exact name, is used throughout the algorithm or program.

When choosing the name of the variable the programmer must adhere to certain rules:

1. The name must be descriptive e.g. grade, mark.
2. It may not contain a space but more than one word may be joined e.g. bookPages, itemPrice. It may also be joined with an underscore character.
3. It may contain letters and numbers, but not only numbers e.g. department23.
4. It must begin with a letter and may not start with a number.
5. It may not contain any special characters e.g. &, #, @, etc.
6. It may not be too long. (In VB.NET, not longer than 32 characters)
7. The name cannot be a reserved word in the language (e.g. Integer)

How to name a variable:

All the characters of the name of a variable are lower case letters except where 2 or more words are joined. In such cases all the following words start with an uppercase letter. Remember that no spaces are allowed in the name of a variable.

Examples of valid variable names:

Description of variable	Possible variable name
Name of employee	empName
Price of car	carPrice
Author of book	author
Quantity in stock	quantity
Age of student	stAge
Total amount of sales	totAmtSales



Questions

The following variable names are invalid. Supply the reasons:

5thGrade _____

member of club _____

abc _____

5472 _____

theAddressOfTheCompanyInTshwane _____

grade&mark _____

3.2.2 DIFFERENT TYPES OF A VARIABLES

3.2.2.1 NUMERIC VARIABLES

Integers:

An integer variable contains a positive or negative number without a decimal part. Examples of integers are the number of students and the quantity of items in stock. It is often used for items that cannot be split, e.g. a marble. The programmer may also decide to use the type integer for items such as weight that could contain decimal positions, but the programmer may prefer to use only an integer value.

Examples: 15, -2334, 9728

Real numbers:

A real number variable contains a positive or negative number with a decimal part. Examples of real numbers are the length of a window in meters and centimetres and an amount of money in Rand and cents.

Examples: 12.47, -987.123, 17.00

Note that a pure numeric value is never enclosed in quotes.

3.2.2.2 NON-NUMERIC VARIABLES

Character:

A character variable contains a single letter, number or special character such as $, %, @, etc. The value of the character is always enclosed in quotes.

Examples: "A", "G", "*", "8".

Note that the character "8", which is declared as character, is not regarded as numeric and may not be used in arithmetic. Such a character may for instance be used as a code where arithmetic is not needed e.g. the group number is "8".

String:

A string variable consists of two or more characters and it must also be enclosed in quotes.

Examples: "32 Long Street" and a message such as "The name is found".

Boolean:

A boolean value can contain only a true or false value.

A boolean variable is used to test if a specific condition is true or false. It is also called a logical variable.

Examples: Does a person have membership of a club?
Did a student pass an exam?

3.2.3 THE VALUE OF A VARIABLE

A variable need not contain an initial value, but a value may be assigned to a variable in the beginning of the algorithm according to the type.

Study the following table to understand all the aspects of a variable:

Description	Variable name	Variable type	Possible value
Name of student	stName	string	"John Smart"
Number of books	noBooks	integer	234
Price of item	price	real	78.56
Student? (Y/N)	student	boolean	true
Code (A – C)	code	character	"B"



Exercises

Complete the following table:

Description	Variable name	Variable type	Possible value
Colour of dress			
Length of man in meters			
Adult?			
Age in years only			
Salary in R/c			
Title of book			
Player in match?			
Class code (K,L or M)			
Author of book			
Number of computers			

3.2.5 VARIABLES IN VISUAL BASIC.NET

3.2.5.1 USE THE HUNGARIAN NAMING NOTATION

We have already seen that the name of all variables must be meaningful in order to describe the purpose of the variable. In addition to that, most VB.NET programmers use the Hungarian notation when naming variables. In this notation a three-letter prefix is used as the first 3 characters of the variable name. This **three-letter prefix** must indicate the **data type of the variable**. This convention enhances readability and ensures that the programmer gets less confused during the programming process.

The different data types and the three-letter prefix for each of the variable names are summarized as follows:

Data type	Prefix	Example
Integer	int	intTotal
Short	shr	shrNumStudents
Long	lng	lngProduct
Decimal	dec	decAmount
Single	sng	sngTemp
Double	dbl	dblPaid
Character	cha	chaDept
String	str	strName
Boolean	bln	blnValueEntered

3.2.5.2 DECLARATION OF VARIABLE NAMES IN VISUAL BASIC.NET

A variable must be declared in a VB.NET program before it can be used. The name of the variable and the data type of the variable must be indicated in the declaration.

Syntax:
Accessibility variablename [As *datatype*] [=*initialvalue*]

The **accessibility** of the variable indicates the scope of the variable. The word **Dim** indicates block level or procedure level scope while the word **Private** indicates module level scope. The scope of a variable will be discussed at **point 3.10**. For now, let's use the word Dim during our examples of declarations.

After the accessibility has been indicated, a valid variable name must be given, followed by data type of this variable. An initial value is optional and may be assigned to a variable during declaration. If the data type is omitted, a default data type of Object will be assigned to the variable. However, this is not recommended.

Example:

```
Dim intNoStudents As Integer = 20
```

A memory position of 4 bytes has been allocated to the variable called intNoStudents. It may only contain integer values and it currently contains a value of 20

We can also declare a number of variables of the same type on the same line:

Example:

```
Dim intTotal, intNumber1, intNumber2 As Integer
```

intTotal, intNumber1 and intNumber2 are three variables. Memory space is allocated to them and each one may contain an integer value. In a case like this, where more than one variable are declared in one statement, no initial value may be assigned to them.

3.2.5.3 DATA TYPES IN VISUAL BASIC.NET

NUMERIC VARIABLES

As mentioned before, numeric variables are variables that can be used in calculations. They are divided into two types.

a) INTEGERS

In Visual Basic.NET there are 3 data types that can accommodate an integer value and they are summarized in the following table. The main difference between these types is the memory space each requires and the values that it may contain:

DATA TYPE	MEMORY REQUIRED	VALUES	
		FROM	TO
Integer	4 bytes	-2,147,483,648	2,147,483,648
Long	8 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,808
Short	2 bytes	-32,768	32,767

Examples of declarations:

```
Dim intNumber As Integer
Dim intNumber As Integer = 0
```

These 2 declarations are the same. If the initial value for an integer is omitted, it will automatically be initialized to 0

```
Dim shrNumber As Short
Dim shrNum As Short
Dim shrNumber, shrNum As Short
```

The memory size for each of these variables will only be 2 bytes and not 4 bytes as for a normal integer. The first 2 statements can also be combined into one statement as indicated in the third statement

```
Dim lngNum As Long = 123455666666
```

A variable of data type long with a big initial value

Remember, we will use a convention to start the names for variables with a prefix to indicate the data type. The prefixes for variables that may store integer values are as follows:

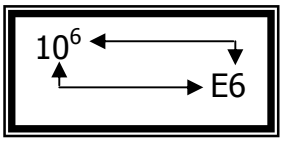
Data type	Prefix
Integer	int
Short	shr
Long	lng

b) REAL NUMBERS

A real number has a decimal position, indicated by a decimal point, not a decimal comma. Data types in Visual Basic.NET that can store a decimal position are classified as **floating point numbers** and **fixed point numbers**.

A floating point number is a number that is expressed as a multiple of some power of 10. Floating point numbers are written in exponential notation, which is similar to scientific notation.

E.g. 2,100,000 is equivalent to
2.1 x 10^6 (scientific notation) and to
2.1E6 (exponential notation)



Floating point numbers can also have a negative number after the E.

E.g. 0.0000021 is equivalent to
2.1E-6
2.1 divided by 10 to the sixth power
Floating point numbers would be used to represent both extremely small and extremely large numbers.

A fixed point number is not expressed as a multiple of some power of 10 and they store numbers with a fixed decimal point. Calculations involving fixed point numbers are not subject to the small rounding errors that may occur when using floating point numbers. Therefore the decimal data type is recommended for calculations involving money.

Data types for real numbers can be summarized in the following table:

DATA TYPE	MEMORY REQUIRED	VALUES	
		FROM	TO
Single	4 bytes	-1.401298E-45	3.402823E38
Double	8 bytes	-4.94065645841E-324	1.79769313486231E308
Decimal	16 bytes	Smallest non-zero number is +/- 0.0000000000000000000000000001	

Examples of declarations:

```
Dim decAmount As Decimal
```

A variable called decAmount may contain a decimal value and will be automatically initialized to 0

```
Dim sngValue As Single = 10
```

The variable called sngValue will contain a floating point number as a value. It will have an initial value of 10

Dim dblClassAvg As Double = 0

A variable of data type double with an initial value of 0

Dim decAmt, decNumber As Decimal

The declaration of two variables can be combined in one statement only if they are of the same data type and if no initial values are assigned to it

Naming conventions for variables declared to contain real values:

Data type	Prefix
Decimal	dec
Single	sng
Double	dbl

NON-NUMERIC VARIABLES

Variables that cannot be used in calculations can be categorised into the following types:

a) CHARACTER

A character data type can store one Unicode character. Unicode is the universal coding scheme for characters. It assigns a numeric value to each character used in the writing languages of the world. Each character is represented by 2 bytes. A character value must always be included in quotes in a statement.

Declaration example:

Dim chaDept As Char = "A"

The variable chaDept is declared to contain one character value and the initial value is "A"

In our naming convention, variables of data type character, must begin with the prefix ***cha***

b) STRING

A string data type can store text from 0 up to approximately 2 billion characters. The text property of most components (e.g. label, button, text box) is of type string.

Declaration examples:

Dim strStudentName, strStudentNumber As String

Both these variables could be of type string. Although the student number will contain a number, it is mostly not used in calculations

Dim strMessage As String = "The value entered was incorrect, please re-enter"

In our naming convention, variables of data type string, must begin with the prefix ***str***

c) BOOLEAN

A variable of data type Boolean can only store the Boolean values True or False. We will make use of Boolean variables to determine whether the user has entered a valid value or not (True or False). It can also be used to reset a radio button or to disable a button.

Declaration example:

Dim blnValidEntry As Boolean = False

A variable called blnValidEntry is initially set to False. After the user has entered a value and it is found to be correct, the value will become true

In our naming convention, variables of data type Boolean, must begin with the prefix ***bln***

OTHER DATA TYPES

Visual Basic.NET also supports the data types date, byte and object. **Date** is used to store date and time information and the **Byte** data type is used to store binary numbers. If you do not assign a specific data type to a variable, Visual Basic.NET will automatically assign the **default data type** of **Object** to it. The data type Object is flexible in the sense that it can accommodate many different types of data. At the beginning of the application a variable could have an initial value of 0 and at a later stage the same variable could have the name "Sally Smith". However, it is not recommended to use this data type as it takes up a lot of memory space. It also slows down the execution time because the computer must first determine the current data type before a statement containing this variable can be executed.



Exercises

Declare variables for the following. Each declaration must have a procedure scope. In each case choose a suitable variable name with the correct prefix:

1. A code for a class group. The initial value for this group must be a "B".
2. A wage of a cashier. Assign a value of R256.34 to the wage.
3. The number of students in a class. The number of students is 250.
4. A name of a student. The name of the first student is "John Adams".

3.3 OPTION EXPLICIT ON

By declaring a variable, a programmer has control over the data type of a variable. However, in Visual Basic.NET it is possible to use a variable that has not been declared. When the compiler detects a variable that has not been declared, it will automatically declare it and assign a data type of Object to it. As mentioned in the previous paragraph, this is not recommended.

To force a programmer to declare all variables before they are used, the statement **OPTION EXPLICIT ON** can be inserted in the general declarations section of the Code Editor Window.

If the code then contains an undeclared variable, the programmer will be informed via a syntax error and the program will not be able to execute before the variable has been correctly declared.

3.4 CONSTANTS

3.4.1 PURPOSE OF A CONSTANT

All the rules of a variable apply to a constant, except for the fact that it has a fixed value right through the entire program. Constants are used in cases where we know that the value will never vary or change during the execution of the program. A constant may be of any data type.

Examples:

There are always 24 hours in a day
The value of π is always 3.142
A message or legend to be printed may be "The name of the student is "

3.4.2 USE OF CONSTANTS IN AN ALGORITHM

In algorithms to follow it will be necessary to display a legend followed by the value of a variable as shown in the following **example**:

The statement wants to display that the percentage obtained by a student is the value that is stored in the variable called *percentage.*

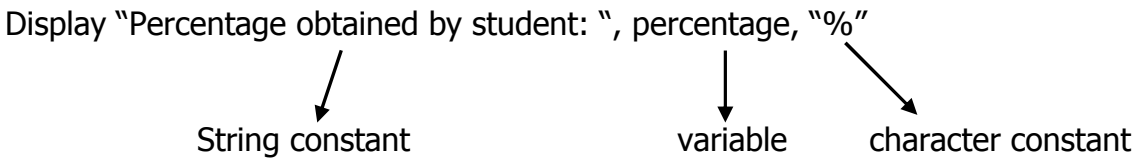
The constant (fixed) part of the display statement will be a legend that has the value "Percentage obtained by student: "
The variable *percentage* will contain the calculated percentage (example: 50) obtained by the student. The variable name *percentage* is not contained in quotes.

The following statement:

display "Percentage obtained by student: ", percentage, "%"

will produce the output

Percentage obtained by student: 50%



3.4.3 DECLARATION AND USE OF CONSTANTS IN VISUAL BASIC.NET

In Visual Basic, a constant value can be used directly in an expression or it can be declared as a constant if the value is known before the program is created.

Syntax to declare a constant:
Const *constantname* [As *datatype*] = *expression*

The declaration must start off with the word Const followed by the name of the constant. Once again, if the data type is omitted, the default type of Object will be assigned to the constant. This is not recommended. A value or expression must be provided in order to assign a value to the constant.

Examples:

Const decVat As Decimal = 0.14
Const intDistinction As Integer = 75
Const strRadioStation As String = "YFM"

3.5 USE OF A VARIABLE IN EXPRESSIONS AND ASSIGNMENTS

The assignment symbol is used to assign an initial value or an expression to a variable.

Syntax: *Variablename = Value*

or

Variablename = Expression

3.6 CONVERSIONS BETWEEN DIFFERENT DATA TYPES

The following section applies when writing programs in Visual Basic.NET:

The variable name on the left hand side of the assignment statement will be of a specific data type. The value or expression on the right hand side of the assignment operator must therefore be of the same type or of a compatible type.

If not, it must be converted to the correct type and certain methods are available for this purpose.

For example:

If numeric values are entered into text boxes it will be contained in the text property of the text box with a data type of String. (Refer to the word text box).

Study the following user interface in **Figure 3-1** where 2 integer values are supposed to be entered and added. The first output label will display the answer where the two input values are added directly to one another and will be treated as text. The second label will display the answer after the two values have been converted and treated as the data type Integer. The names for the controls are txtNumber1, txtNumber2, lblAnswer1 and lblAnswer2.

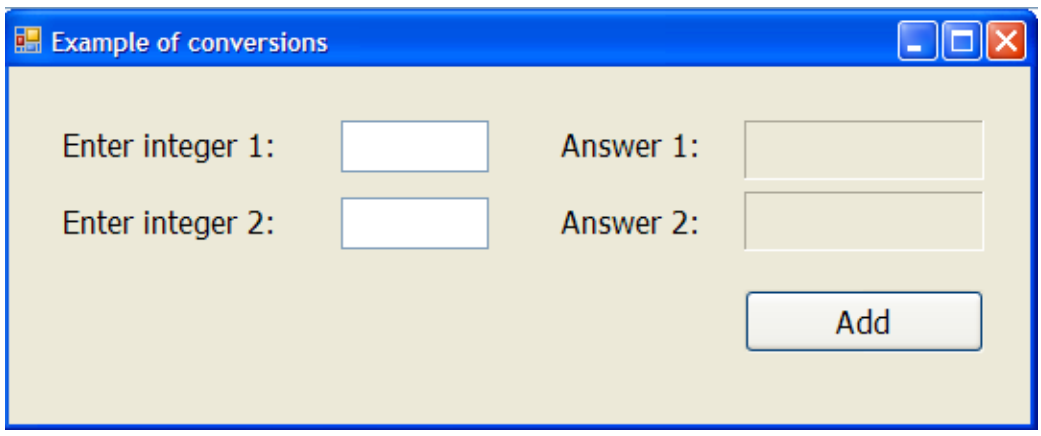


Figure 3-1

Code for the Add-button:

```
Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handl
    lblAnswer1.Text = txtNumber1.Text + txtNumber2.Text
    lblAnswer2.Text = Convert.ToInt32(txtNumber1.Text) + Convert.ToInt32(txtNumber2.Text)
End Sub
```

Answers after two integers have been entered:

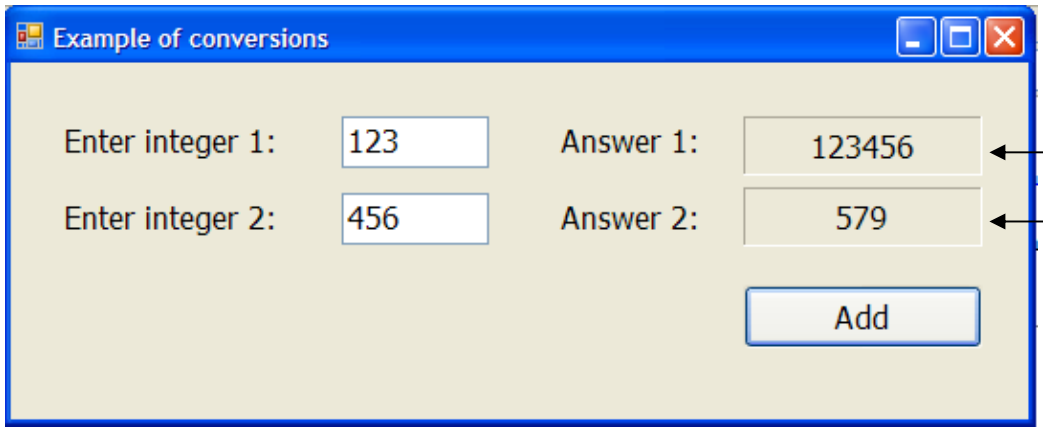


Figure 3-2

If the values in the text boxes are not converted to integer values, the + operator acts as a concatenate operator between two strings and conversions are therefore essential before any calculations can be done, especially when numeric values are to be entered into text boxes.

The conversion methods for all the data types are provided in the following table:

Method	**Convert *value* to the following data type**
Convert.ToInt32(value)	Integer
Convert.ToInt16(value)	Short
Convert.ToInt64(value)	Long
Convert.ToDecimal(value)	Decimal
Convert.ToSingle(value)	Single
Convert.ToDouble(value)	Double
Convert.ToChar(value)	Character
Convert.ToString(value)	String
Convert.ToBoolean(value)	Boolean

Tip:
It is not necessary to enter the complete method. As soon as you type the "." a list of all possible methods in the Convert class will appear. You can simply select the applicable method and press the tab key.

Therefore, if the number of students is entered into a text box called txtNum and the value must be placed in a variable (intNoStudents) of data type integer, the statement could be as follows:

```
intNoStudents = Convert.ToInt32(txtNum.Text)
```

More Conversion Examples:

```
intNumber = Convert.ToInt32(shrNum)
decAmtDue = Convert.ToDecimal(intNoBought * decPrice)
sngRate = Convert.ToSingle(txtRate.Text)
shrHours = Convert.toInt32(txtHours.Text)
chaDept = Convert.ToChar(txtDept.Text)
decAmount = Convert.ToDecimal(dblAmnt)
```

If a constant value is assigned to a variable, the following literal type characters may be used to ensure that the variable and the value are of the same data type.

Data type of variable	Literal type character
Integer	I
Short	S
Long	L
Decimal	D
Single	F
Double	R
Character	C

3.7 LOSS OF PRECISION DUE TO CONVERSION OF DATA TYPES

We know that a value or expression on the right hand side of the assignment operator must be of the same type or of a compatible type of the variable on the left hand side and if not, that the programmer must perform the necessary data type conversion.

However, a programmer must be very careful, because it is possible that a loss of precision can occur when conversions are done. This will be illustrated in the following example.

A user must enter 2 numbers via text boxes.

- The **first value** will be converted and placed in a variable of data type **double**.
- The **second value** will be converted and placed in a variable of data type **decimal**.

These two variables will then be multiplied in the following ways:

- Ensure that both variables are of data type decimal and place the answer in a variable of data type decimal
- Ensure that both variables are of data type double and place the answer in a variable of data type double
- Convert both variables to a data type single before it is multiplied. Place the answer in a variable of data type single.

These three answers will then be displayed on three labels, to a precision of 7 positions after the decimal point.

Studying the output in **Figure 3-3**, it is clear that some of the precision is lost when these two variables have been converted to a data type of single. In a case where the precision is not important, the programmer could allow conversions like this. An example where precision does not play a major role is conversion from °F to °C.

The code for our current example:

```
Private Sub btnConvert_Click(ByVal sender As System.Object, ByVal e As Sys
    Dim decNumber, decAnswer As Decimal
    Dim dblNumber, dblAnswer As Double
    Dim sngAnswer As Single
    dblNumber = Convert.ToDouble(txtNumber1.Text)
    decNumber = Convert.ToDecimal(txtNumber2.Text)
    decAnswer = decNumber * Convert.ToDecimal(dblNumber)
    dblAnswer = Convert.ToDouble(decNumber) * dblNumber
    sngAnswer = Convert.ToSingle(decNumber) * Convert.ToSingle(dblNumber)
    lblAnswer1.Text = decAnswer.ToString("F7")
    lblAnswer2.Text = dblAnswer.ToString("F7")
    lblAnswer3.Text = sngAnswer.ToString("F7")
End Sub
```

Example of two input values and the corresponding outputs:

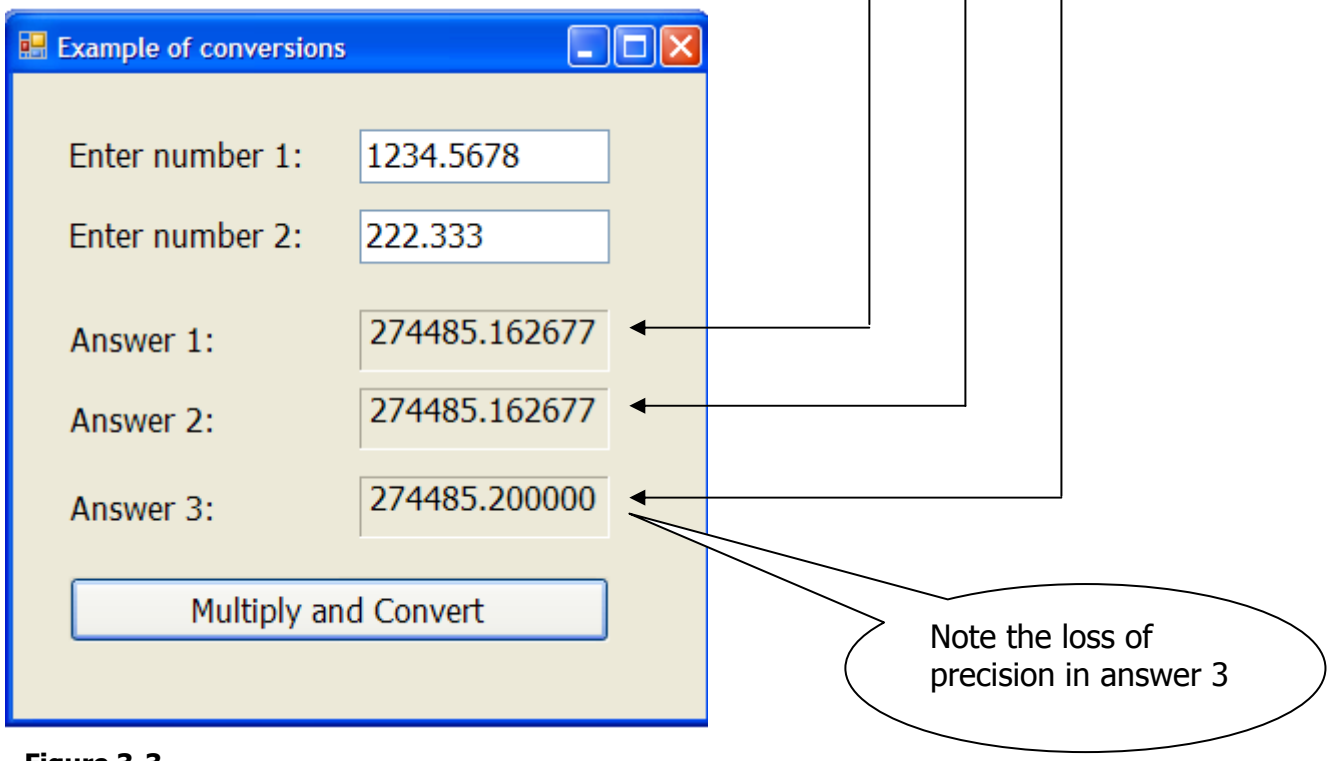


Figure 3-3

3.8 OPTION STRICT ON

It is a good programming practice to do the necessary conversions where applicable. However, if the programmer omits a conversion, Visual Basic.NET will automatically perform an implicit conversion.

We have seen that data could be lost where implicit conversions are performed on a data type with less precision or smaller capacity.

In order to force a programmer to do critical conversions to ensure correctness and preciseness of values, the statement **OPTION STRICT ON** can be inserted in the general declarations section of the Code Editor Window.

If the statement **Option Strict On** is specified, the following implicit conversions will **not** automatically be done, and must be performed by the programmer:

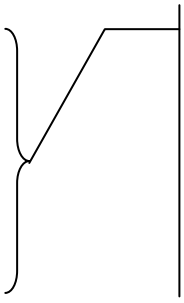
String to Decimal
String to Single
String to Double
String to Integer
String to Short
String to Long
String to Character
Single to Decimal
Double to Decimal
Double to Single

The reason why it will not automatically be done, is that the programmer must be aware of the fact that certain data losses may occur and it is as if the programmer must *give permission that it may be allowed*.

Take for example a string to be converted to a character. After the conversion process, it will contain only the first character of the string and the remaining characters in the string will be lost. If the programmer knows that the user will enter only one character in a text box (which is always of data type string), it will be valid to convert it to a character data type and the programmer can give the go-ahead!

Conversion examples where a fixed constant is assigned to a variable:

```
intNumber = 6I
shrDept = 2S
lngNoCaught = 456777L
decAmount = 456.99D
sngAve = 44.5F
dblNum = 34545.55R
chaDept = "A"C
```



Note:
If **Option Strict On** is specified, it is only necessary to insert D and C to specify these conversions. All other conversions will automatically be done and will be accurate.

3.9 ARITHMETIC EXPRESSIONS

The majority of computer programs contain arithmetic. It is now important to study how arithmetic is done when solving computer related problems. These problems may calculate average marks of students, an amount due on an invoice or given the gross salary of an employee, determine the net salary.

Arithmetic is often done by using equations e.g.

answer = 4 + 7.

The calculation to the right of the equal sign is done first and then the result is assigned to the variable *answer*. So, after the statement is executed the variable *answer* will contain a value of 11.

The equal sign (=) has two meanings:

1. It can be used to assign a value to a variable.
   - total = 0 ~ where total is a integer variable
   - noOfStudents = 50 ~ where noOfStudents is an integer
2. It can be used to replace a value of a variable
   - result = number * 2 ~ where result and number are integers
   - sum = sum + points ~ sum and points are numeric variables

The last example, sum = sum + points will add the value of *points* to the current value of *sum* to produce a new value for *sum*. The previous value of *sum* is now increased by the value of the variable *points*.

If *sum* has a value of 200 and *points* has a value of 27, the right hand side of the equation will now be 200 + 27 and the new value of *sum* will be 227.

3.9.1 ARITHMETIC OPERATORS

An operator is a symbol used within an expression or equation that tells the computer how to process the data. This symbol joins the operands to be processed.
The following operators are used to perform arithmetic operations:

Operator	Description	Example	Result	Precedence
^	Exponentiation (To the power of)	2 ^ 4	16	1
-	Negation	-TRUE	FALSE	2
*, /	Multiplication and Division	5 * 7 72 / 8	35 9	3
\	Integer division	37 \ 5	7	4
Mod	Modulus arithmetic	37 mod 5	2	5
+,-	Addition and Subtraction	4 + 7 14 - 5	11 9	6

Integer division may only be done when dividing an integer value by another integer value because it discards (drops) the decimal part and also does not round the answer e.g. 49 \ 10 = 4.

Modulus arithmetic is done when the user only wants to know what the value of the remainder is when dividing e.g. 49 mod 10 = 9.

The order of precedence of execution as shown in the table of operators is important. When the execution of an expression is evaluated, it is always done

- from left to right and
- the operator with the highest order of precedence is done before the other operators

Study the following example:

5 - 3 ^ 2 \ 8 + 17 mod 3 * 2

To find the answer of this expression we have to determine which operator to use first. The answer of the calculation done will now be printed in bold and underlined.

Studying the expression we see that the exponentiation (^) must be done first:

5 - **9** \ 8 + 17 mod 3 * 2
Next step, multiplication and division, from left to right

5 - 9 \ 8 + 17 mod **6**

Now, integer division

5 - **1** + 17 mod 6

Now, modulus arithmetic

5 - 1 + **5**

Lastly, plus and minus are done, from left to right with the minus done first because it is to the left of the plus

4 + 5

9

Parentheses are used to change the order of execution. Calculations to be done in parentheses have a higher priority than any of the operators. Within the parentheses the priority of execution of the operators applies.

Another example:

Calculate the value of the variable k where
k = a * b ^ (14 – c) mod 4 + 3 \ a where a, b and c are integer variables and have the following values: a = 3, b = 5, c = 11

Substitute the values of the variables before calculating:

k = 3 * 5 ^ (14 – 11) mod 4 + 3 \ 3
= 3 * 5 ^ **3** mod 4 + 3 \ 3
= 3 * **125** mod 4 + 3 \ 3
= **375** mod 4 + 3 \ 3
= 375 mod 4 + **1**
= **3** + 1
= **4**
k = 4

A problem statement:

Kevin works 8 hours per day at a pay rate of R11 per hour. He has to pay R12 taxi fare per day. How much money will he take home after a 5 day work week?

money = (hours * payRate – transport) * noDays
= (8 * 11 – 12) * 5
= 380

Note that we do not include the units i.e. the rand sign R in any calculation. It also applies to other units such as meters, cm, gram, kilogram, etc.

3.9.2 EXAMPLES IN VB.NET

Declare the necessary variables and constants and write arithmetic expressions for each of the following examples:

Example 1:

Thandi has 257 items and wants to pack them in packets of 14 each. How many full packets will she have?

```
Dim intItems, intFullPackets As Integer
Const intNoPacket As Integer = 14

intFullPackets = intItems \ intNoPacket
```

Example 2:

Joseph earns R5.75 per hour. He works 8 hours per day, but has to pay R3.60 bus fare daily. Determine his net income per day.

```
Const decTariff As Decimal = 5.75
Const decBus As Decimal = 3.6
Dim decNetIncome As Decimal
decNetIncome = decTariff * 8 – decBus
```

Example 3:

Bonita bought a number of items at R15.85 per item. She received a discount of 7.5%. Calculate the amount she has to pay.

```
Const decPrice As Decimal = 15.85
Const decDiscount As Decimal = 0.075
Dim decAmtDue As Decimal
Dim intNoBought As Integer
decAmtDue = Convert.ToDecimal(intNoBought) * decPrice
decAmtDue = decAmtDue – decAmtDue * decDiscount
```



Exercises

1. Calculate the value of x in each of the following exercises:

1.1 x = g – 5 + 14 mod h ^ 2 where g = 12 and h = 1
1.2 x = 27 * y \ 3 / 2 – z + 2 where y = 24 and z = 4
1.3 x = a / (b + 2 ^ 2) – 12 + c mod b –2 where a = 96, b = 2, c = 98
1.4 x = (y - 2 * z) + y / 6 * w \ 2 where w = 10, y = 12, z = 2
1.5 x = 35 mod y - (z * w + y / 3) + y * w where w = 5, y = 24, z = 3

2. Write the following in pseudocode format and calculate the results for each of the following:

2.1 Sandra bought six pairs of socks at R3.50 per pair. What is her amount due?
2.2 Vusi wrote 2 tests and obtained percentages of 75 and 63. Calculate her average mark.
2.3 Jimmy has 25 marbles and want to give each of 4 children an equal number. How many will each child receive?
2.4 Solly bought a number of items and the amount due is R245.60. He receives a discount of 5%. Calculate the new amount due.
2.5 Loveness packs baskets of biscuits containing 15 biscuits each. How many full baskets can she pack if she has 257 biscuits.

3.10 SCOPE AND LIFETIME OF A VARIABLE

The **scope** of a variable indicates where this variable may be used in the application's code. The **lifetime** of a variable will indicate how long the variable will remain in the internal memory of the computer.

Most variables have a **procedure scope**. E.g. if a variable called intSalary is declared as *Dim intSalary As Integer* in the btnCalc control's click event procedure, it means that *only* the btnCalc control's Click event procedure may use intSalary. No other procedures in the application may use the variable intSalary. Such a variable is called a procedure-level variable. The lifetime of a variable with procedure scope is the same as that of the procedure in which it is declared. In other words, when the procedure is activated, the variable will be declared and it will remain in memory while the procedure is running. The moment that the procedure ends, the variable will be removed from memory.

A variable can also be declared with **module scope**. In such a case the variable is declared in the form's declaration section, which begins with the Public Class statement and ends with the End Class statement. Such a variable is called a module-level variable and can be used by any of the procedures in the form. A module-level variable will normally start off with the word Private although it may also start with Dim. The lifetime of a module-level variable is the same as that of the application. Therefore, when the application starts, the variable will be created in memory and it will remain in the memory until the application ends.

When would we use a procedure-level variable and when would we use a module level variable?

- If a variable is required by more than one procedure, e.g. more than one button event, it is better to declare a module-level variable instead of a procedure-level variable (see example in **Figure 3-7**).

- If a variable must retain its value throughout the application, it is better to declare a module-level variable rather than a procedure level variable. Remember, a procedure level variable will be removed from memory when the procedure ends. In the case of an on click event (if the same button is clicked for the second time) the variable will be re-created with its initial value again. If totals must be accumulated, it will start off a zero every time which is not desirable. Examples of this will be shown when repetition statements and accumulators are discussed at a later stage.

It is also possible to declare a variable with a **block scope**. Such a variable is declared within a selection statement, such as an IF-THEN-ELSE statement, or within a repetition statement, such as a FOR-NEXT statement, in which case the variable may only be used within that specific statement. Variables will block scope will be discussed in Chapter 7.

3.10.1 EXAMPLE TO DEMONSTRATE PROCEDURE-LEVEL VARIABLES VERSUS MODULE-LEVEL VARIABLES

Let's design the user interface in **Figure 3-4** to represent a simple calculator that can either add 2 numbers, subtract the second number from the first, multiply the two numbers or divide the second number into the first. The user must first enter the two integer numbers via two text boxes and then press an appropriate button to perform the required action.

After the two integers have been entered they must be placed into two variables of the type integer. The answer for the addition, subtraction and multiplication will also be of type integer and the answer of the division must be of type decimal.

If we use of procedure level variables, the same two numbers will have to be declared in each button-click procedure. However, if we use of module-level variables, the variables will only have to be declared once in the form's declaration section which will be a more effective way for this application.

The conversion from the text boxes to the variables will still have to be done in each of the procedures because a user may click on the multiplication button before he or she clicks on the addition button. At a later stage, when sub procedures and functions are discussed, we will be able to avoid this duplication by placing it in one sub procedure.

Example of the user interface:

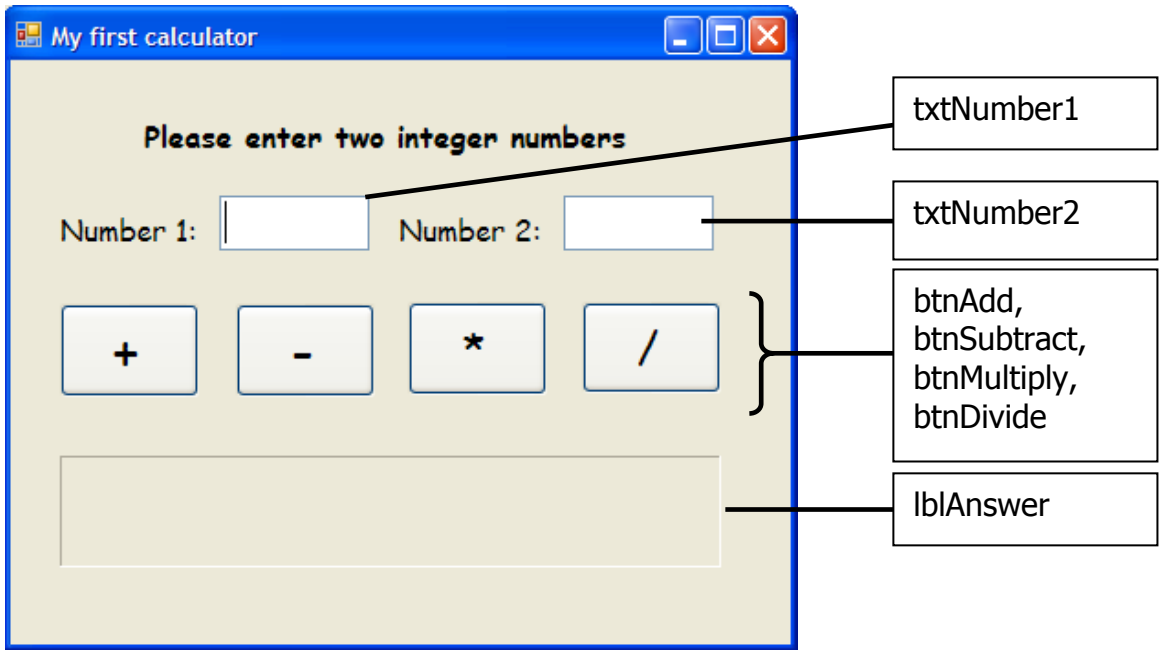


Figure 3-4

An example of input and output when the addition button was clicked is provided in **Figure 3-5**:

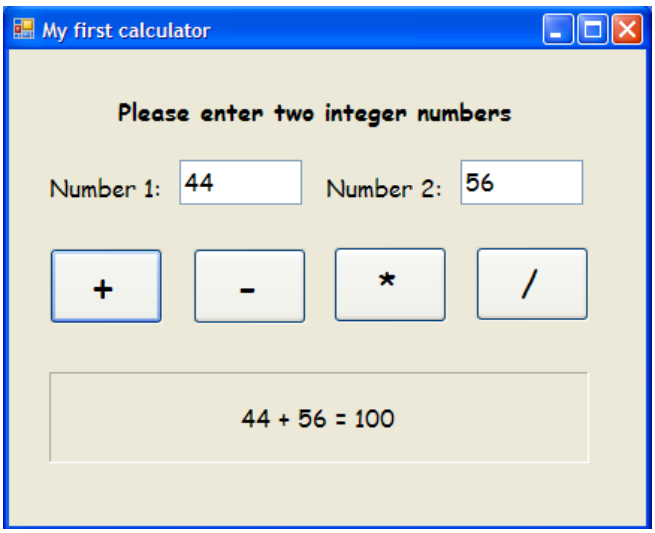


Figure 3-5

The procedures in the code editor with procedure-level variables are indicated in **Figure 3-6**. Note that there are duplication of variables because every variable may only be used in the procedure where it has been declared:

```
Public Class frmCalculator

    Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Han
        Dim intNumber1, intNumber2 As Integer
        Dim intAnswer As Integer
        intNumber1 = Convert.ToInt32(txtNumber1.text)
        intNumber2 = Convert.ToInt32(txtNumber2.text)
        intAnswer = intNumber1 + intNumber2
        lblAnswer.Text = intNumber1 & " + " & intNumber2 & " = " & intAnswer
    End Sub

    Private Sub btnSubtract_Click(ByVal sender As System.Object, ByVal e As System.EventArgs
        Dim intNumber1, intNumber2 As Integer
        Dim intAnswer As Integer
        intNumber1 = Convert.ToInt32(txtNumber1.Text)
        intNumber2 = Convert.ToInt32(txtNumber2.Text)
        intAnswer = intNumber1 - intNumber2
        lblAnswer.Text = intNumber1 & " - " & intNumber2 & " = " & intAnswer
    End Sub

    Private Sub btnMultiply_Click(ByVal sender As System.Object, ByVal e As System.EventArgs
        Dim intNumber1, intNumber2 As Integer
        Dim intAnswer As Integer
        intNumber1 = Convert.ToInt32(txtNumber1.Text)
        intNumber2 = Convert.ToInt32(txtNumber2.Text)
        intAnswer = intNumber1 * intNumber2
        lblAnswer.Text = intNumber1 & " * " & intNumber2 & " = " & intAnswer
    End Sub

    Private Sub btnDivide_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Dim intNumber1, intNumber2 As Integer
        Dim decAnswer As Decimal
        intNumber1 = Convert.ToInt32(txtNumber1.Text)
        intNumber2 = Convert.ToInt32(txtNumber2.Text)
        decAnswer = Convert.ToDecimal(intNumber1 / intNumber2)
        lblAnswer.Text = intNumber1 & " - " & intNumber2 & " = " & decAnswer.ToString("N2")
    End Sub
End Class
```

Scope of variables

Scope of variables

Scope of variables

Scope of variables

Figure 3-6

Figure 3-7 is based on the same application, but now the procedures in the code editor contain module level variables. It is now only necessary to declare the variables intNumber1, intNumber2 and intAnswer once in the declaration of the form. The variable decAnswer is still declared in the last procedure because the answer of the division may result in a decimal answer, which will be different from the answers of the first 3 procedures.

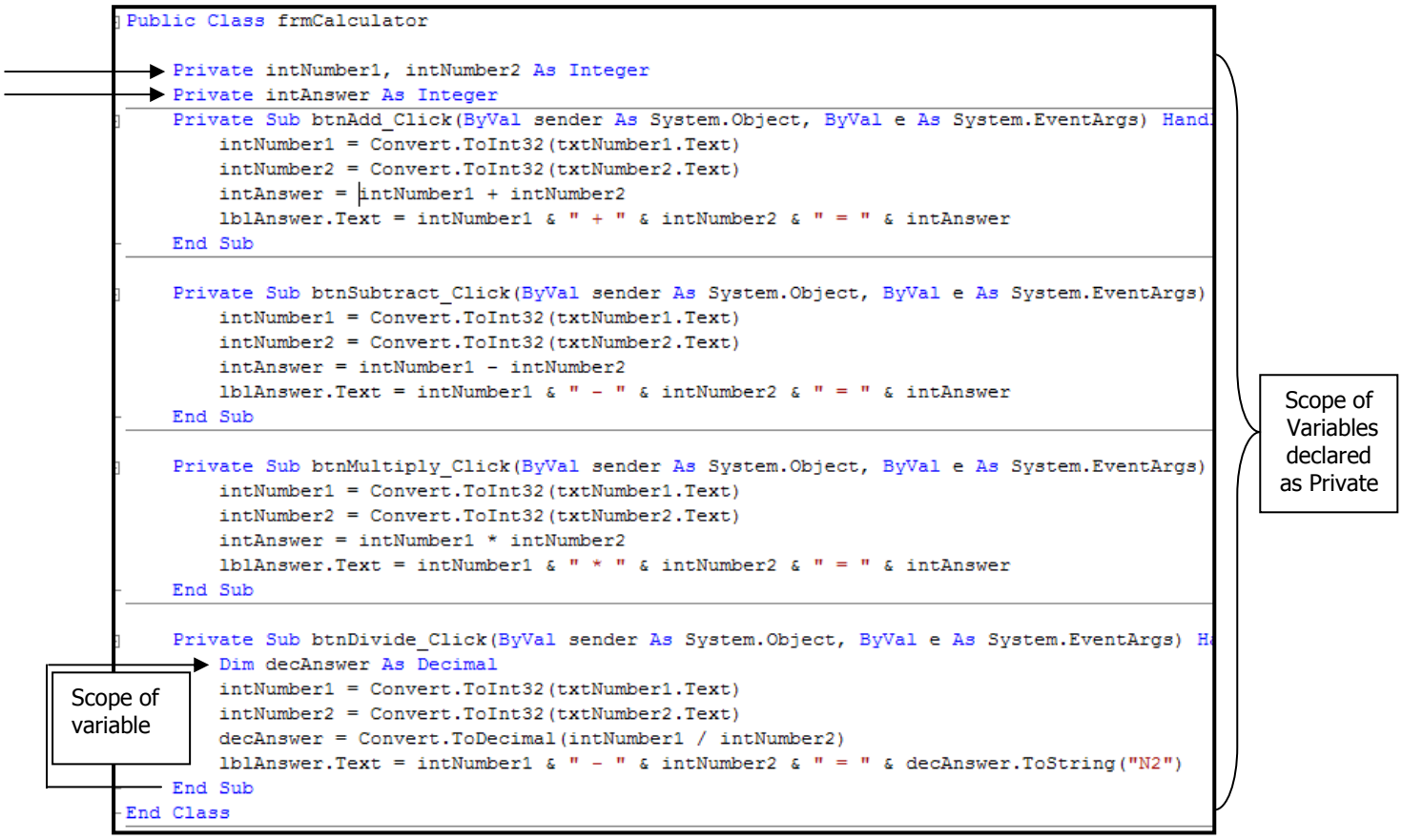


Figure 3-7

3.11 FORMATTING NUMERIC OUTPUT

By now we know that the left and right hand sides of an expression must be of the same type. We usually use text boxes to enter data and the text property of a text boxes is of type string. If we need to do calculations with the data entered, it must be converted to a numeric data type.

However, when we need to display the answer, the output will be displayed on a label and the text property of a label is once again of type string. The numeric output must therefore be converted back to a string data type.

We can then also format the output in order to display it in a meaningful format.

The concept of converting data types between input, processing and output can be summarised as indicated in **Figure 3-8**.

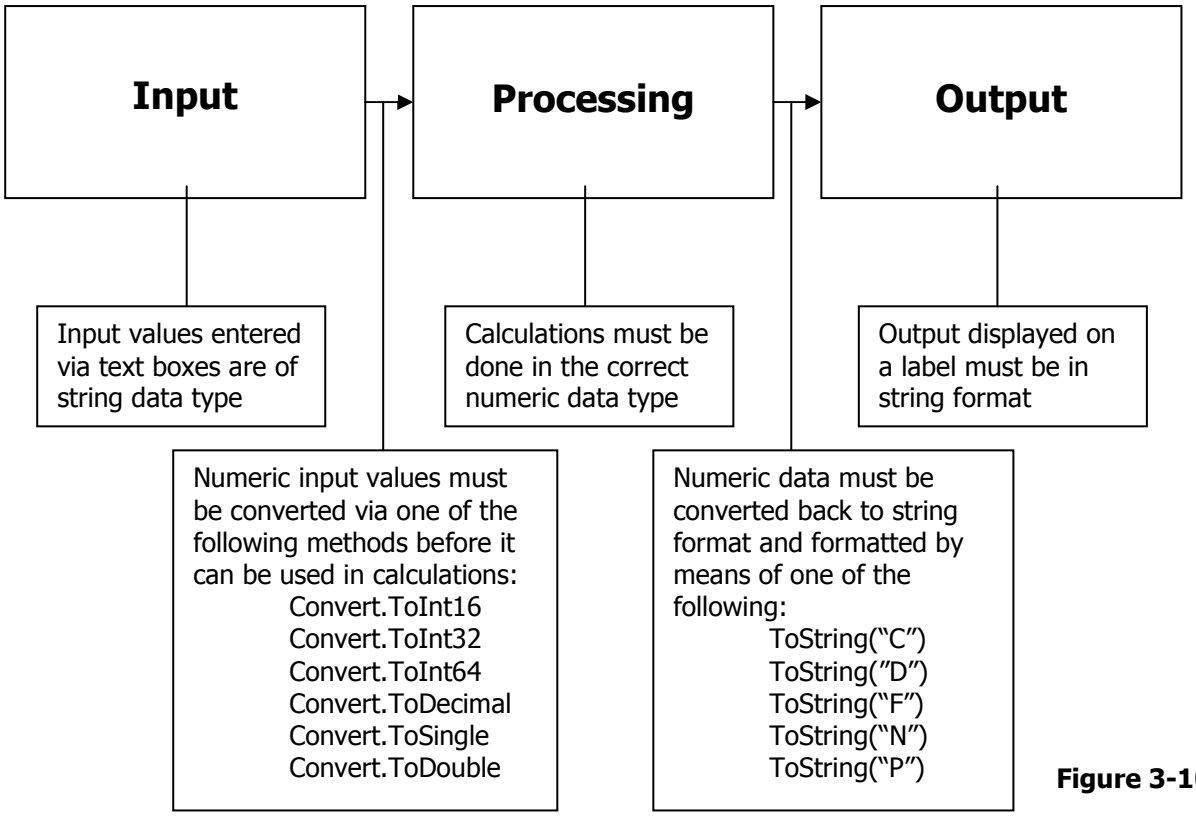


Figure 3-10

The formatting is done via a format specifier (C, D, F, N or P) and a number called the precision specifier that controls the number of digits to the right of the decimal point in the new formatted number.

The numeric variable that must be displayed on the label must be followed by the convertion *.ToString* followed by the format specifier and possible precision specifier in quotes within brackets.

Examples:

```
decAmtDue.ToString("N2")
intNumber.ToString("D")
decAmount.ToString("C0")
shrAmt.ToString("F2")
dblAverage.ToString("P1")
```

3.11.1 FORMAT SPECIFIERS

C: Currency
A dollar sign will be included in the output. A comma will be inserted as a thousands separator and negative numbers will be enclosed in brackets.

D: Decimal
This format specifier may only be used for integers. A comma will be inserted as a thousands separator and negative numbers will be preceded by a minus sign.

F: Fixed point
This format specifier is used for real numbers. There will be no thousands separator and negative numbers will be preceded by a minus sign.

N: Number
This format specifier is used for real numbers. A comma will be inserted as a thousands separator and negative numbers will be preceded by a minus sign.

P: Percent
The number will be multiplied by 100 and a percentage sign will be added to the number when it is displayed. Negative numbers will be preceded by a minus sign.

3.11.2 PRECISION SPECIFIERS

Decimal
The precision specifier will indicate the number of digits to display. If it is omitted the full value will be displayed. If necessary the number will be padded with zeros to the left e.g. if 5 digits must be displayed and the number to display is 17, it will be displayed as 00017.

Currency, Fixed point, Number and Percent
The precision specifier will indicate the number of decimal positions that must be displayed. If no precision specifier is specified, it will display 2 decimal positions.

3.11.3 EXAMPLES

The following table contains examples of formatted numeric output where A is the format specifier and xx the precision specifier.

ToString("Axx")	Value in Variable	Output displayed
D	5	5
D	-5	-5
D2	5	05
D3	5	005
C	3067.28	$3,067.28
C2	3067.28	$3,067.28
C0	3067.28	$3,067
C1	3067.28	$3,067.3
C2	-3067.28	($3,067.28)
F	3067.28	3067.28
F2	3067.28	3067.28
F0	3067.28	3067
F1	3067.28	3067.3
F2	-3067.28	-3067.28

ToString("Axx")	Value in Variable	Output displayed
N	3067.28	3,067.28
N2	3067.28	3,067.28
N0	3067.28	3,067
N1	3067.28	3,067.3
N2	-3067.28	-3,067.28
P	.0755	7.55%
P	.0750	7.50%
P2	.0750	7.50%
P1	.0750	7.5%
P0	-0.05	-5%



Design the following user interface in **Figure 3-11** and add the code provided in **Figure 3-12** for the button click event. Predict the output that will be displayed and then run the program to check your results.

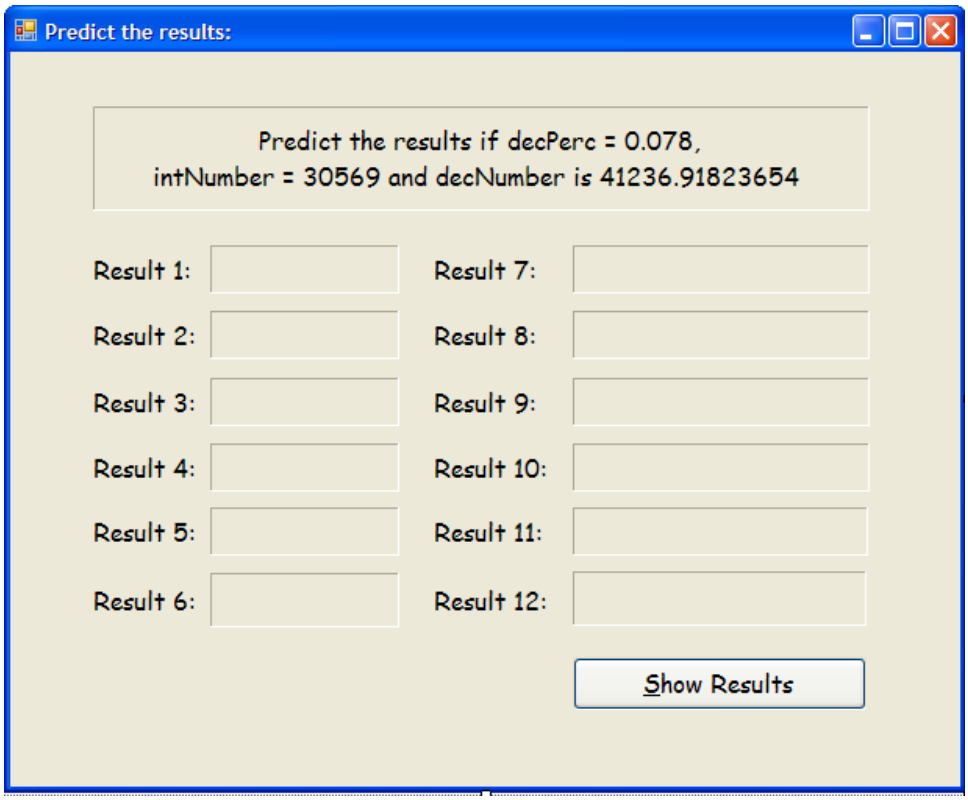


Figure 3-11

```
Private Sub btnResults_Click(ByVal sender As System.Object, ByVal e As System.
    Dim decPerc As Decimal = 0.078
    Dim intNumber As Integer = 30569
    Dim decNumber As Decimal = 41236.91823654
    Dim decAnswer As Double
    lblAnswer1.Text = intNumber.ToString("D2")
    lblAnswer2.Text = intNumber.ToString("D7")
    lblAnswer3.Text = decNumber.ToString("C")
    lblAnswer4.Text = "R" & decNumber.ToString("N2")
    lblAnswer5.Text = "R" & decNumber.ToString("F2")
    lblAnswer6.Text = decNumber.ToString("F0")
    lblAnswer7.Text = "The percentage = " & decPerc.ToString("P1")
    lblAnswer8.Text = decPerc.ToString("P4")
    lblAnswer9.Text = "Answer = " & decNumber.ToString("F5")
    decAnswer = Convert.ToDecimal(intNumber) + decNumber
    lblAnswer10.Text = decAnswer.ToString("F5")
    decAnswer = decAnswer * 0.1D
    lblAnswer10.Text = decAnswer.ToString("N3")
    decAnswer = Convert.ToDecimal(intNumber) * 0.1
    lblAnswer11.Text = decAnswer.ToString("F1")
    lblAnswer12.Text = decAnswer.ToString("F0") & decNumber.ToString("F0")
End Sub
```

Figure 3-12



Exercises

Indicate the exact value that will be displayed on the label in each of the following cases.

1. decAmount = 19613.417D
   lblAnswer.text = "The amount = R" & decAmount.ToString("N2")

2. decAmount = 19613.417D
   lblAnswer.text = "The amount = R" & decAmount.ToString("F")

3. decAmtDue = 2399.619D
   lblAnswer.text = "The amount due = " & decAmtDue.ToString("C")

4. intNoStudents = 180
   lblAnswer.text = "The number of students is " & _
   intNoStudents.ToString("D4")

5. decVat = 0.14D
   decAmt = 1000D
   decFinalamt = decAmt + decAmt * decVat
   lblAnswer.text = decVat.ToString("P0") &" VAT added to R" & _
   decAmt.ToString("F") & " = R" & decFinalamt.ToString("F2")

CHAPTER 4
CREATING A USER INTERFACE WITH BASIC CONTROLS

4.1 INTRODUCTION

In chapter 2 you were introduced to a user interface, but the idea was just to get you started. No components were discussed in detail and the application that you created did also not contain any variables in the code. This chapter will discuss labels, text boxes, group boxes, buttons and picture boxes in detail. Other controls (also called components) will be discussed in later chapters. All controls can be created or instantiated by choosing the applicable tool in the toolbox.

OUTCOMES

When you have studied this chapter you should be able to:

- name controls according to the Hungarian naming convention,
- know the purpose of a label control and how to add it to a form,
- know the purpose of a text box control and how to add it to a form,
- know the purpose of a button and how to add it to a form,
- know what access keys are and how to set it,
- set a default button and cancel button on a form,
- group certain related controls in a group box control,
- add a picture box control to a form,
- set different commonly used properties for each type of control,
- place a background image on a form,
- design user interfaces according to good GUI design guidelines.

4.2 NAMING CONVENTIONS

4.2.1 USE MEANINGFUL NAMES FOR CONTROLS

When you place four text boxes on a form they will automatically be named TextBox1, TextBox2, TextBox3 and TextBox4. If you do not rename the textboxes it could become difficult to remember the purpose of every text box. The same applies to labels, buttons and all the other components on a user interface. By giving components meaningful names, the programmer not only makes his or her own task easier, but the program will also be more user-friendly and understandable to other programmers who might need to study the code.

4.2.2 THE HUNGARIAN NAMING CONVENTION

Most VB.NET programmers use the Hungarian notation when naming components and variables. In this notation a three-letter prefix is used as the first 3 characters of the component or variable name. In the case of a component, this three-letter prefix will indicate what type of component it is. The remaining part of the component name must be descriptive to indicate the purpose of the component.

All the components that we will be using and the three-letter prefix that should be used for each one are summarized in the following table. An example is also provided in each case.

Component	Prefix	Example	Purpose
Form	frm	frmStudent	Form for a student application
Label	lbl	lblHeading	Label to display a heading on the form
Text Box	txt	txtMark1	Text box to capture test mark 1
Button	btn	btnCalc	Button to trigger calculations
Picture Box	pic	picLogo	Picture box that contains a logo
Group Box	grp	grpMarks	Group box to group all marks together
Check Box	chk	chkAbsent	Chck box to test if student was absent for a test
List Box	lst	lstResults	List box to list all the results
Radio button	rad	radFinal	Radio button to indicate if mark counts towards final mark

4.3 A LABEL CONTROL

4.3.1 PURPOSE OF A LABEL CONTROL

The purpose of a label control is to display instructions and information to the user. The user is not allowed to modify this text while the application is running.

In the user interface in **Figure 4-1**, the first label is used to display a heading to provide information about the program, the next 2 labels are used to identify the text boxes where the user must enter information and the last label is used to display the output. In each case, the user may not change the text value of any of these labels while the application is running.

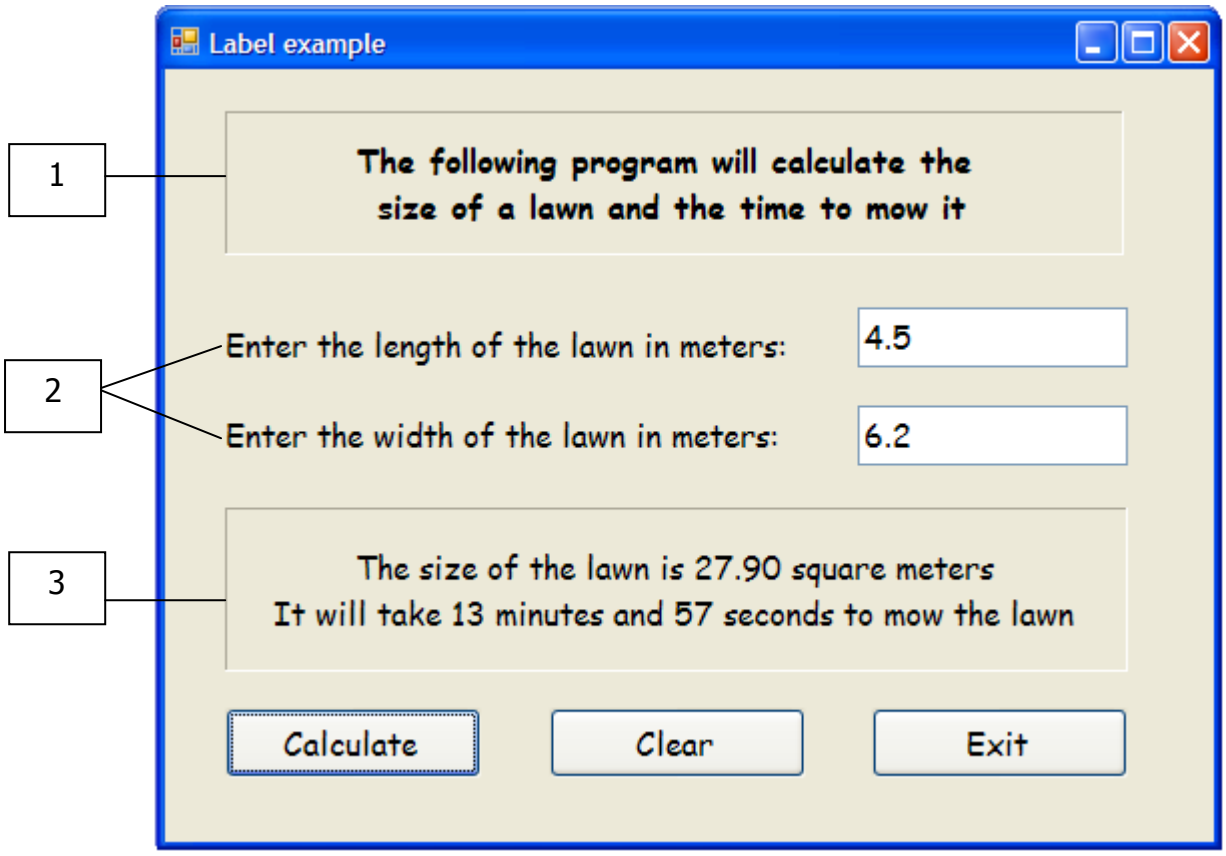


Figure 4-1

4.3.2 PROPERTIES MOST COMMONLY USED FOR A LABEL CONTROL

Every label must have a meaningful name starting with lbl according to the Hungarian notation. The first step to do after you have placed a label on the form would be to give it a meaningful name in the **name property**. Do not leave the name label1 or label2. The next step would be to replace the **text property** with the text that you want to display on the label. These 2 properties are the minimum that must be set for any label on a form. If the purpose of the label is to display output, the initial text property of the label will be an empty string.

The **text align property** will determine where in the label the text will be displayed. From the following Properties window **(Figure 4-2)** we can gather that the current active component on the form is a label named lblAnswer. It is currently empty as indicated in the text property, but when a text value is assigned to it, it will be placed in the middle, centre as indicated in the text align property.

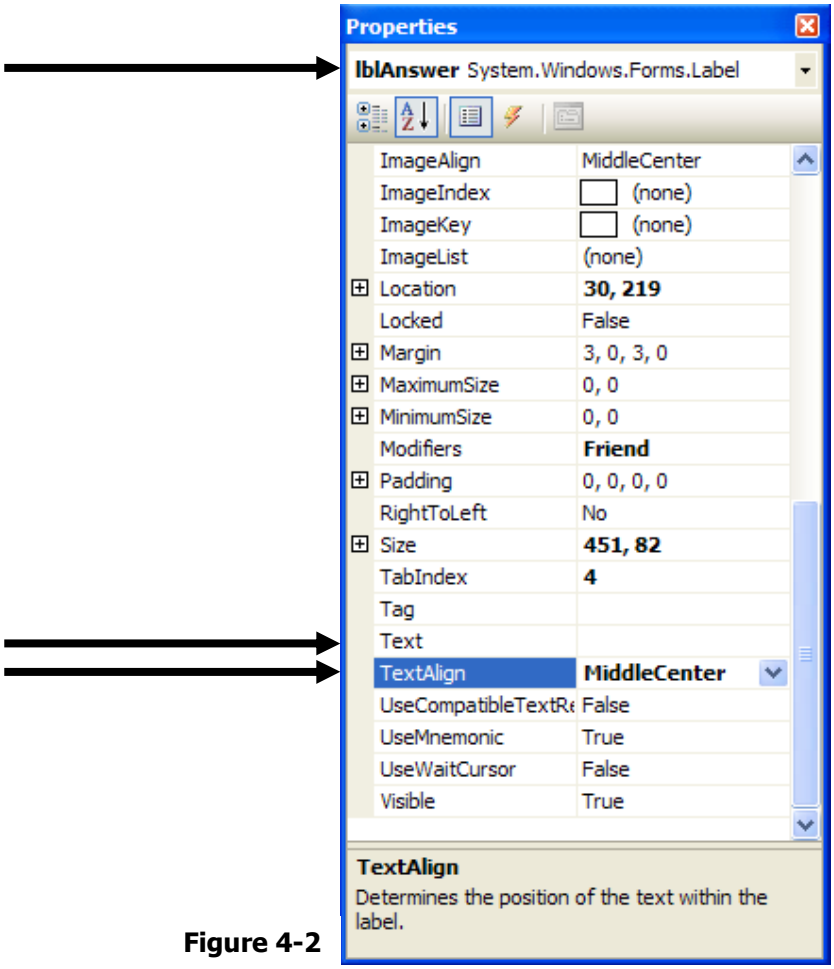


Figure 4-2

If we want to change the position of text within the label, we must click on the arrow in the text align property. The various options, as indicated in **Figure 4-3**, will then be displayed and a new position may be selected.

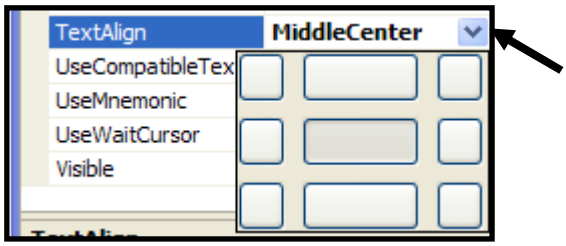


Figure 4-3

The **AutoSize property** automatically sizes the label to fit its current text property. If this property is set to "false" the size of the label can be controlled by you. Compare the two labels in the form in **Figure 4-4**.

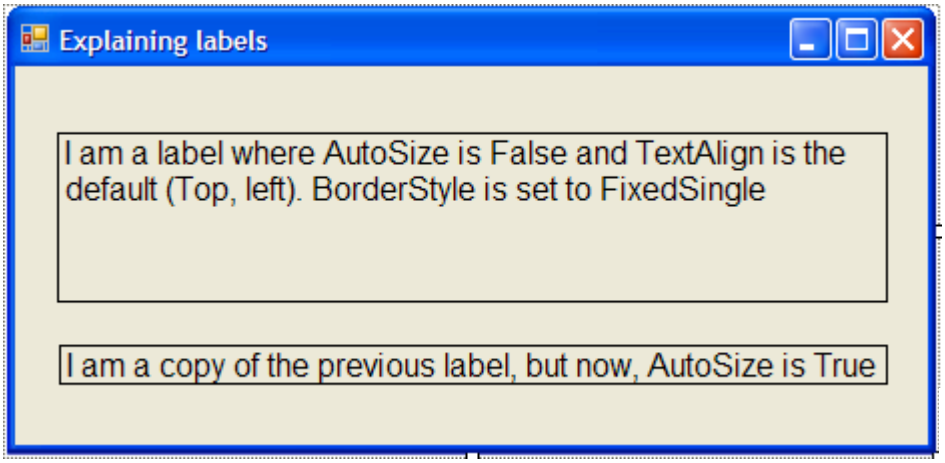


Figure 4-4

Both labels in **Figure 4-4** have a border which can be set in the **BorderStyle property**. The default is none, in which case the label will not have any border. The type of border in Figure 4-4 is FixedSingle, but a border may also be set to Fixed3D, in which case it will look like the borders of the labels in **Figure 4-1**. Once again, to change the type of border, click on the arrow next to the property value and make your choice, as indicated in **Figure 4-5**.

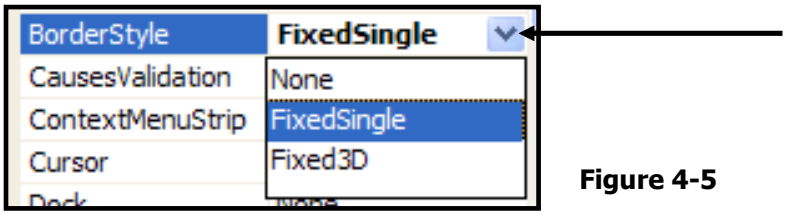


Figure 4-5

You can also set the font property which will change the font and appearance of the text that will be displayed on the label. Part of the font-property is to display the text in bold, italics, underlined etc. as indicated in **Figure 4-6**.

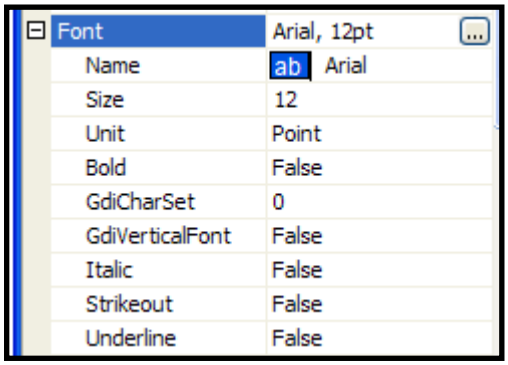


Figure 4-6

There are many other properties to set. As mentioned in chapter 2, the best way to learn is to explore and to try various options. Go to different properties of a label to see what the effect of the change will be.

4.3.3 GUI DESIGN GUIDELINES FOR A LABEL CONTROL

Keep the following in mind when placing a label onto a form and setting the properties of the label:

- Label controls must be used to display information that you don't want the user to change while the application is running.
- A label to identify a text box must be placed to the left or above the text box, should end with a colon and should be typed in sentence capitalization.
- Labels to identify text boxes should be left aligned.
- Labels that identify controls should have their BorderStyle property set to none.
- Labels that display program output, such as the answer of a calculation, normally has the BorderStyle property set to FixedSingle or Normal3D.
- Restrict the highlight of text to headings.

4.4 A TEXT BOX CONTROL

4.4.1THE PURPOSE OF A TEXT BOX CONTROL

A text box is used to provide the user an area in which to enter data. Data that a user enters in a text box will be placed in the text property of the text box. Although the value entered may be numeric, it is treated as text and when entered it is therefore not in the correct format to be used in calculations. Before calculations can be done on numeric values entered in a text box, it must first be converted to the correct format and assigned to a variable. This has been discussed in chapter 3.

A text box can also be used to display information. In such a case the enabled property must be set to FALSE, to ensure that the user may not change the information.

In the example in **Figure 4-7**, the first text box is provided to enter the length of a lawn in meters and the second text box is provided to enter the width of the lawn in meters.

Note, that although the length is entered as 4.5 it is still treated as text and it is not in the correct format to use directly in a calculation and the same applies to the width of the lawn that is entered as 6.2.

I

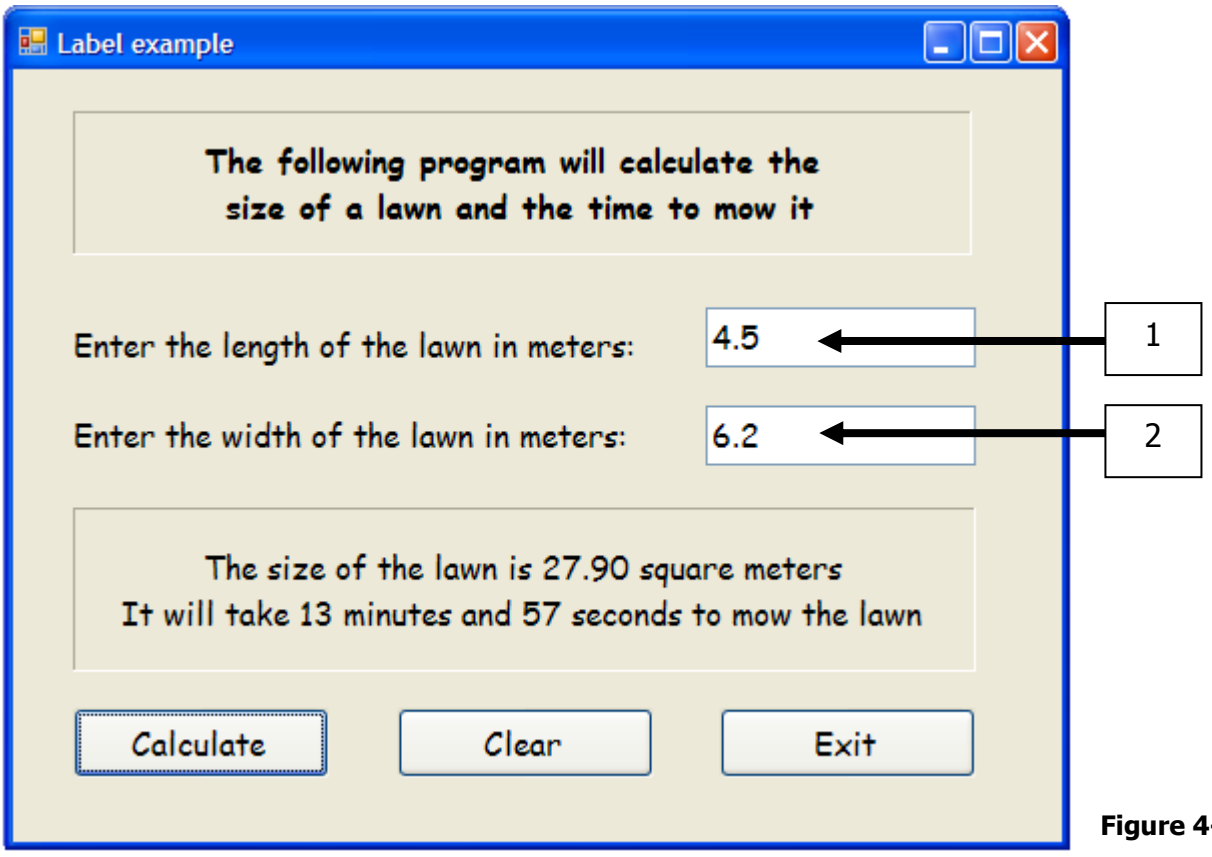


Figure 4-7

4.4.2 PROPERTIES MOST COMMONLY USED FOR A TEXT BOX CONTROL

Once again, the first property to be set for every text box is the name. Give every text box a meaningful name in the **name property** according to the Hungarian notation that starts with txt. Don't leave it as textBox1 and TextBox2. In our example meaningful names could be txtLength and txtWidth.

Other properties to set could be any of the following:

BackColor:	Specify the background color of the text box
Enabled:	If set to TRUE, the user can enter new values in the text box or modify existing values in the text box.
ForeColor:	Specify the color of the text displayed in the text box
Font:	Specify the font to use when text is entered
Text:	Specify the text that appears in the text box. Initially it will be empty
TextAlign:	Specify the position of the text within the text box
MaxLength:	Specify the maximum number of characters that can be entered in the text box

Once again, the best way to learn is to explore and try different properties in order to see the effect and result of each.

4.4.3 GUI DESIGN GUIDELINES FOR A TEXT BOX CONTROL

Keep the following in mind when placing text boxes on the form:

- Although the background and foreground color of a text box can be set, the number of colors on an interface should be limited to 3. The colors you choose should complement one another and should not distract the attention of the user.
- Related text boxes should be grouped together. A group box may be used for this purpose.
- A text box must always contain a related label in order to give clear instructions of what should be entered in the text box
- Be consistent with fonts and don't use a different font for every text box on a form

4.5 A GROUP BOX CONTROL

4.5.1 THE PURPOSE OF A GROUP BOX CONTROL

Related controls can be grouped together using a group box control. This control can be instantiated (created) by choosing the GroupBox in the toolbox window.

In the following example, **Figure 4-8** has been modified to group the two text boxes to enter the length and width of a lawn.

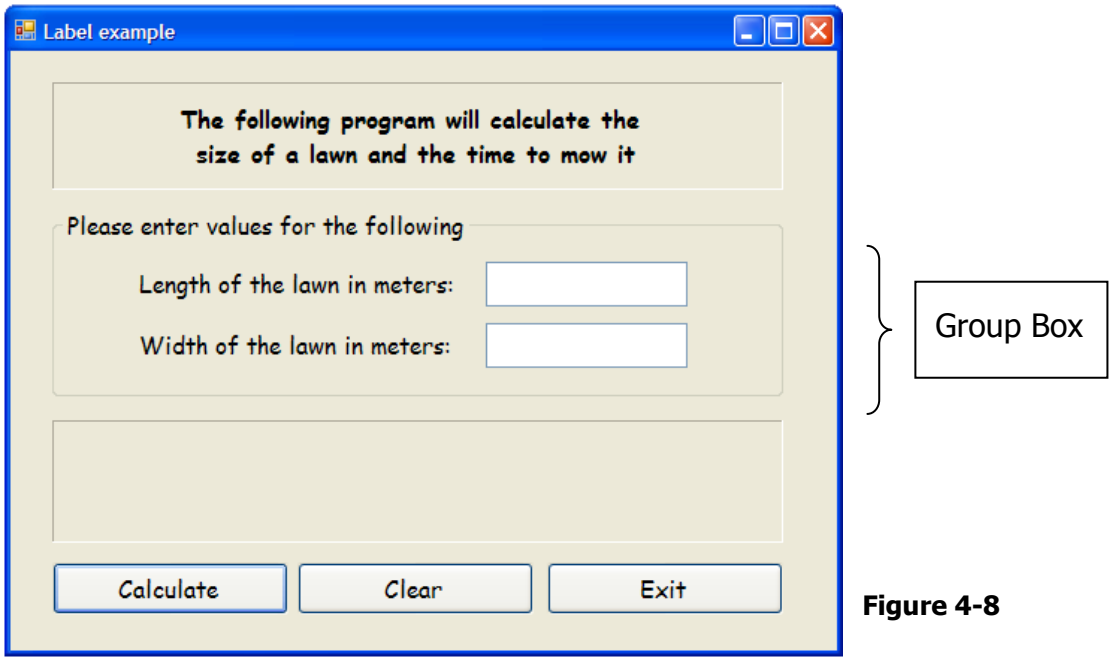


Figure 4-8

4.5.2 PROPERTIES MOST COMMONLY USED FOR A GROUP BOX CONTROL

A group box must have a meaningful name starting with the prefix grp. Don't leave it as groupbox1. Change the name of the groupbox in the **Name property**.

The text property of the groupbox must contain the value that will be displayed in the upper-left corner of the control.

To change the color of the text to black, click on the arrow next to the ForeColor property. Choose custom and select the color as indicated in **Figure 4-9**.

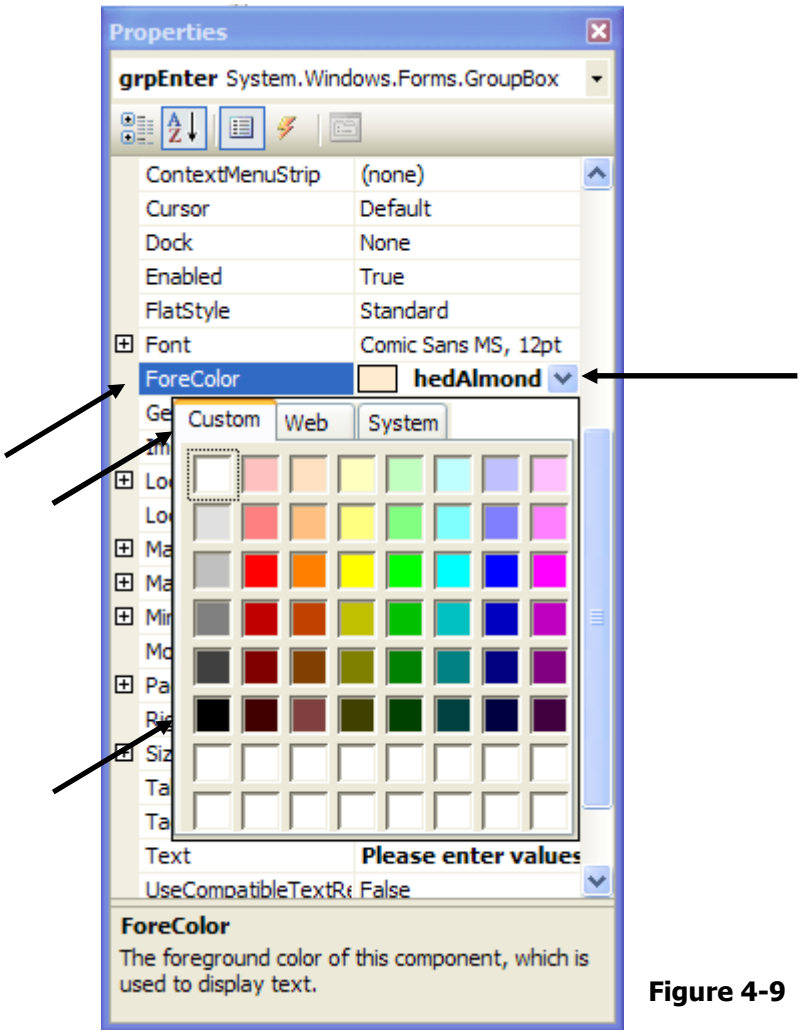


Figure 4-9

4.5.3 GUI DESIGN GUIDELINES FOR A GROUP BOX CONTROL

Keep the following in mind when placing a group box on the form:

- Use a group box to group related controls together.
- Maintain a consistent margin from the edge of the group box.
- Align the borders of the controls in the group box where possible to minimize the number of different margins within the group box.

4.6 A BUTTON CONTROL

4.6.1 THE PURPOSE OF A BUTTON CONTROL

The purpose of a button control is to perform an immediate action when clicked. Buttons are commonly used in most Windows applications. Think of a game of solitaire. After you successfully completed the game, a window will appear to ask you if you want to play another game or not. In this window, there will be 2 buttons – Yes and No. If you click on No, the game will be terminated, but if you click on Yes, a new game will be initiated.



Figure 4-10

In applications that we will write, there will normally have to be a button to click in order to start the processing after the user has entered input values. The action to perform would then be to do the necessary calculations and to display the answers as output on the screen.

There must also be a button to clear all input and output controls in order for the user to enter new input values and to possibly repeat the process.

It is not compulsory, but a good technique is to also include an exit-button to terminate the application.

There were three buttons in **Figure 4-8**. The calculate-button must be clicked after input values have been entered in which case the size of the lawn and the time it will take to mow the lawn will be calculated and displayed on the label. When the clear-button is clicked, the two text boxes and the label will be cleared and if the exit-button is clicked, the application will terminate.

4.6.2 PROPERTIES MOST COMMONLY USED FOR A BUTTON CONTROL

A button must have a meaningful name in the **name property** and according to the Hungarian notation it starts with btn. Don't leave it as Button1 or Button2. The code in the code editor is associated with a specific button and therefore it is extremely important that a button must have a meaningful name.

Other properties to set could be any of the following:

BackgroundImage:	Specify the background image displayed in the control.
Enabled:	Indicate whether the button can respond to an action (such as clicking).
Image:	Specify the image to display on the face of the control
ImageAlign:	Specify the alignment of the image displayed on the face of the control.
Font:	Specify the font to use when text is entered.
Text:	Specify the text that appears on the control.
TextAlign:	Specify the position of the text within the button.

Although a background image can be specified for a button and although an image could be displayed on the face of a button, in this course we will not design user interfaces with such buttons.

4.6.3 ACCESS KEYS

The **text** property of a button may contain an **ampersand (&)**.The character following the ampersand will be underlined when the text is displayed on the button. This character will be the access key and will enable the user to select the button by pressing the alt key as well as the specific character which is underlined.

It is not always convenient for a user to move and click the mouse in order to select a button. For instance, if a user must enter various text boxes it will be faster to keep his/her hands on the keyboard and to select the button via an access key. Access keys will also allow users with disabilities (who might not be able to use a mouse) to use the application. Lastly access keys allow a user to work with the application even if the mouse becomes inoperative.

In the following example **(Figure 4-11)**, the user interface of Figure 4-8 has been modified to change all three buttons to contain access keys. An access key is usually the first character of the text displayed on the button. However, because two of the controls start with a C, the access key of the second button is the second character. As far as an exit-button is concerned, it is normal practice to make the access key an "x" because "x" is associated with exit.

Take note that access keys are not case sensitive. Therefore alt + X will be equivalent to alt + x.

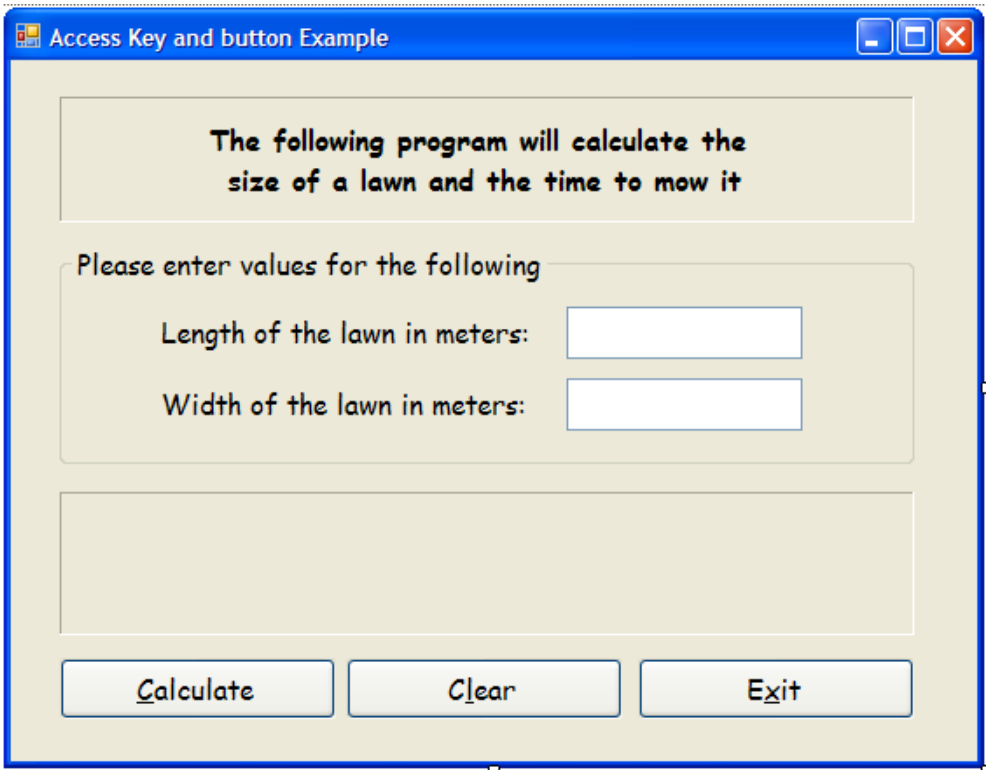


Figure 4-11

For this user interface, the following properties have been set for the calculate-button as indicated in **Figure 4-12**.

Name:	btnCalc
Text:	&Calculate
TextAlign:	MiddleCentre

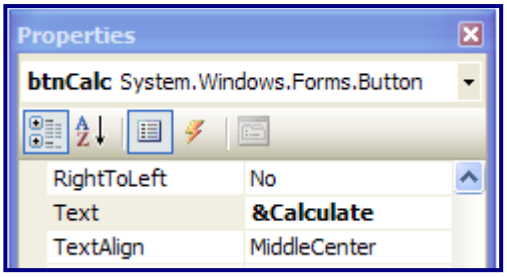


Figure 4-12

4.6.4 DEFAULT AND CANCEL BUTTONS

In 4.6.3 we mentioned that it can be faster to use an access key instead of moving the mouse to position it on the correct button control and to click the control in order to enable a certain event.

An even faster method is to specify the button that is normally clicked to perform the calculations as the default button and to specify the button that is normally clicked to terminate the application as the cancel button.

In our example, the Calculate-button could be specified as the default button and the Exit-button can be specified as the cancel button.

When a default button has been specified, the user can only press enter in order to activate it and that would be even faster than selecting an access key. To activate the cancel button, the user can only press the escape key.

A default button and cancel button is specified for the user interface and therefore it is part of the properties of the form. In our case it can be specified as indicated in **Figures 4-13** and **4-14**.

Go to the properties for the user interface

The default value for the AcceptButton property is usually none. By selecting btnCalc, this button becomes the default button for the form

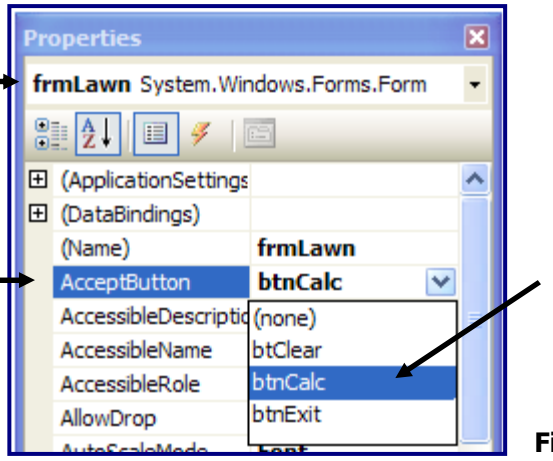


Figure 4-13

The default value for the CancelButton property is also usually none. By selecting btnExit, this button becomes the default button for the form

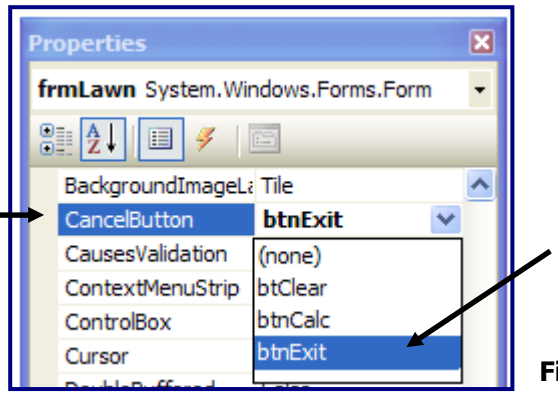
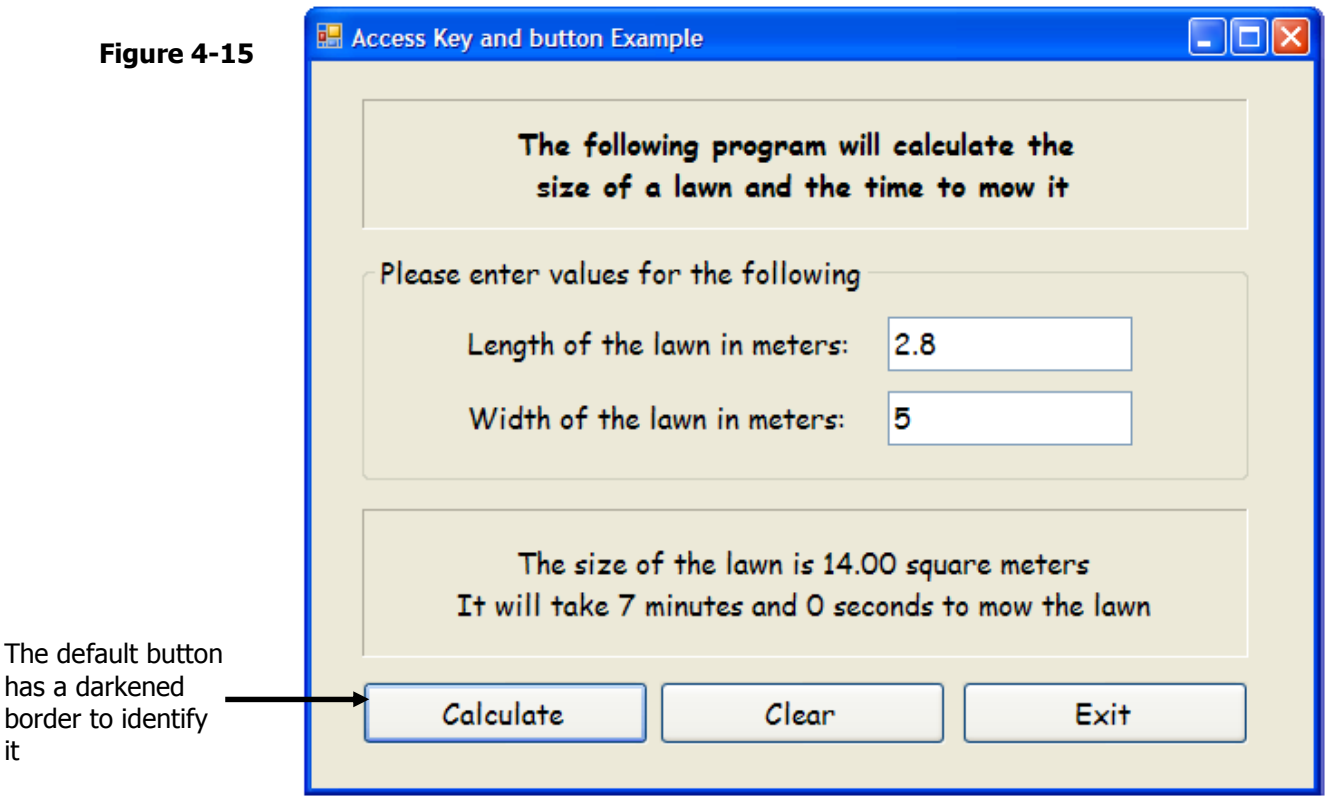


Figure 4-14

When the calculate-button has been selected as the default button, the control will be highlighted, as indicated in **Figure 4-15**.

Figure 4-15



4.6.4 GUI DESIGN GUIDELINES FOR A BUTTON CONTROL

Keep the following in mind when placing buttons on the form:

- A button control is used to perform an immediate action when clicked
- The text property of a button should be entered in book title capitalization. (Capitalize the first letter of every word)
- When buttons are positioned horizontally on the screen, all the buttons should be of the same height. It is preferred that the widths are also the same although they may vary.
- When buttons are positioned vertically on the screen, all buttons should be of the same height and width.
- The commonly used button should be placed first.
- This button that represents the user's most used action should also be the default button.
- When assigning an access key to a button use the first letter of the caption unless another letter provides a more meaningful association. If the first letter has already been used for another access key, use a distinctive consonant. Lastly use a vowel or a number.

4.6.5 CLICK EVENT PROCEDURE

When designing an application, double clicking on a button will invoke a sub procedure in the code editor. The actual coding for this button can then be entered between the header and footer of the sub procedure.

If we double click on the Exit-button, the following empty sub procedure, which corresponds to this button, will appear in the code editor **(Figure 4-16)**.

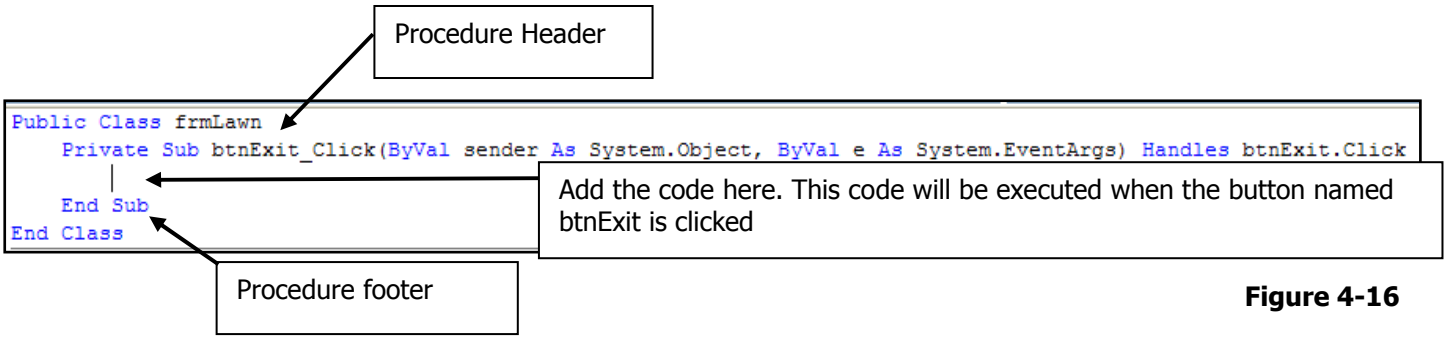


Figure 4-16

Figure 4-17 shows the sub procedure for the btnExit click event after the code has been entered. This event only contains the statement *Close()* in order to terminate the application when the button is clicked.

```
Public Class frmLawn
    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Click
        Close()
    End Sub
End Class
```

Figure 4-17

4.7 A PICTURE BOX CONTROL

4.7.1 THE PURPOSE OF A PICTURE BOX

Sometimes it might be necessary to display a company's logo as part of the header of the user interface or a diagram in order to highlight or explain something to the user.
In such a case an image can be added to the user interface by placing it on a Picture Box control. This control can be instantiated by selecting the PictureBox control from the toolbox window.

Assume the user interface in **Figure 4-18** has been created to handle orders for the Delight Chicken Farm that sells whole chickens, chicken portions and eggs. You want to place an image of a chicken in the spot next to the name of the farm.



Figure 4-18

Currently the Solution Explorer for this application looks as follows **(Figure 4-19).**

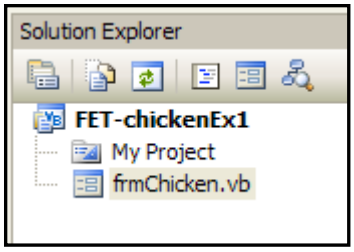


Figure 4-19

When the picture box control has been added to the user interface, click on the image property as indicated in **Figure 4-20** in order to place an image on it.

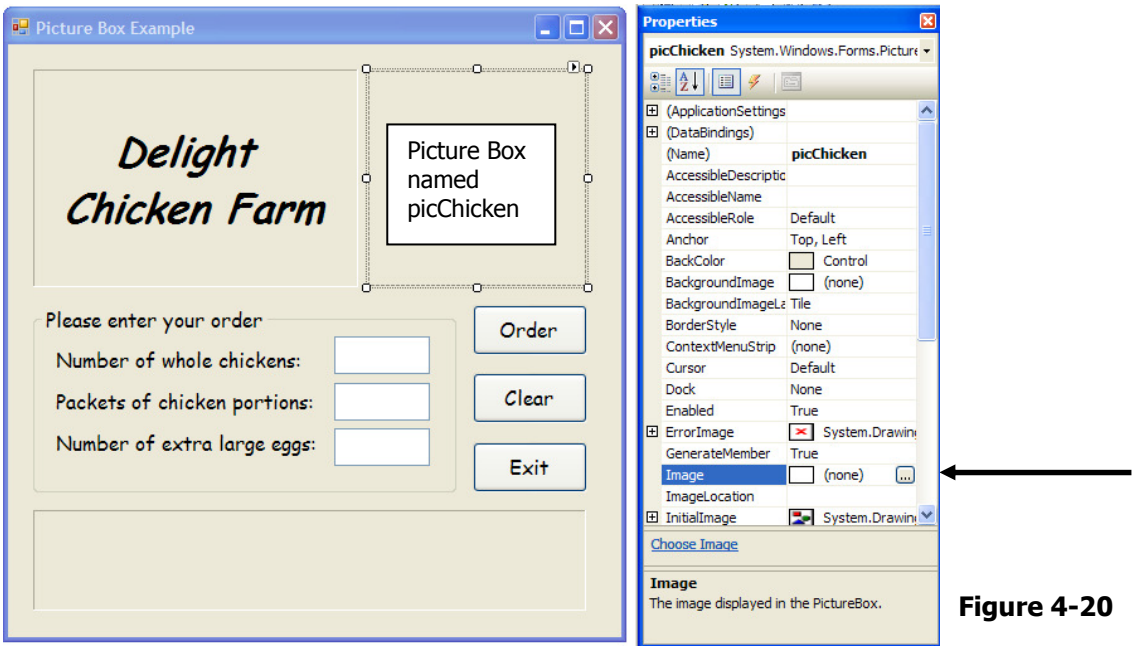


Figure 4-20

When the image property has been selected, the following dialog box, as indicated in **Figure 4-21** will appear.

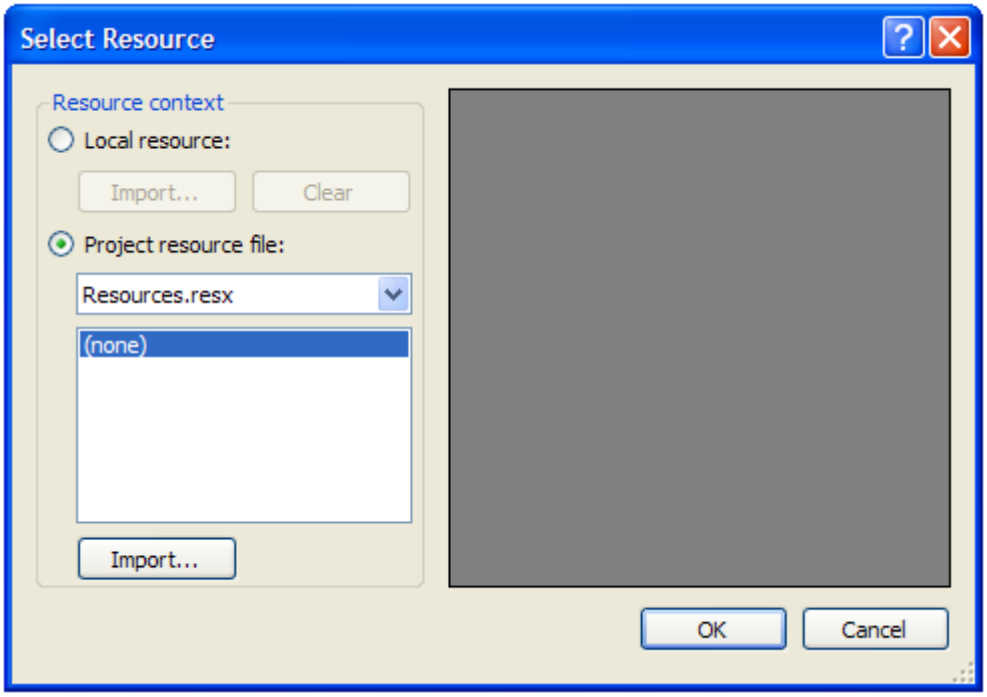


Figure 4-21

Click in Import in order to select the picture file that must be placed in the Picture box. Once a file has been selected, the file will be added to the list and the image will appear in the dialog box as indicated in **Figure 4-22**. Once you click on OK, the picture will be placed on the control.

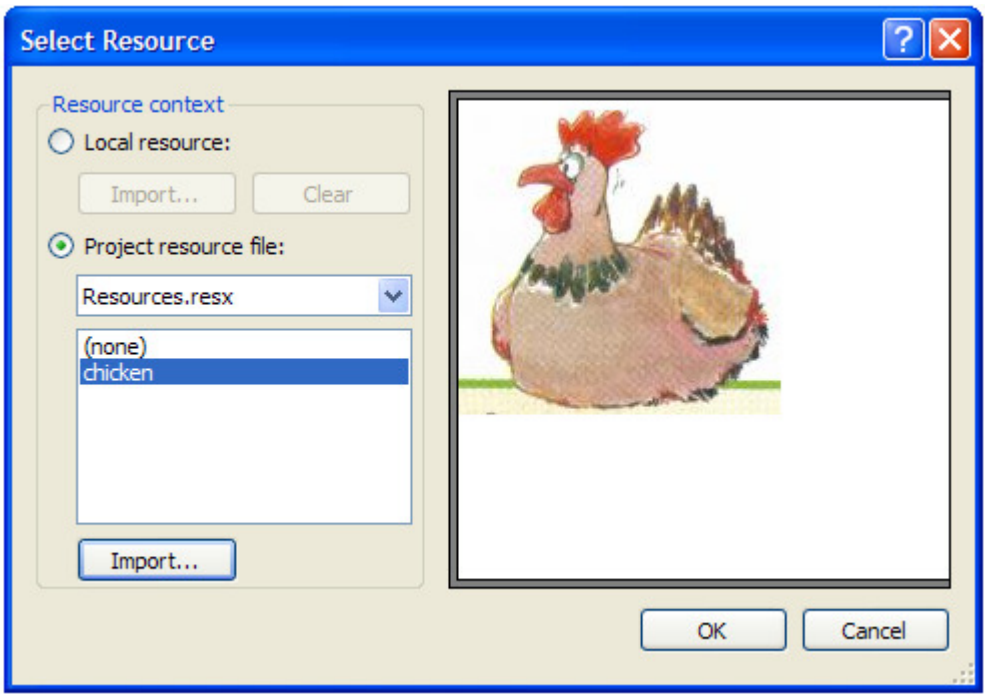


Figure 4-22

Figure 4-23 indicates the final result and **Figure 4-24** indicates the new contents of the Solution Explorer in order to indicate that a new file now forms part of this solution.

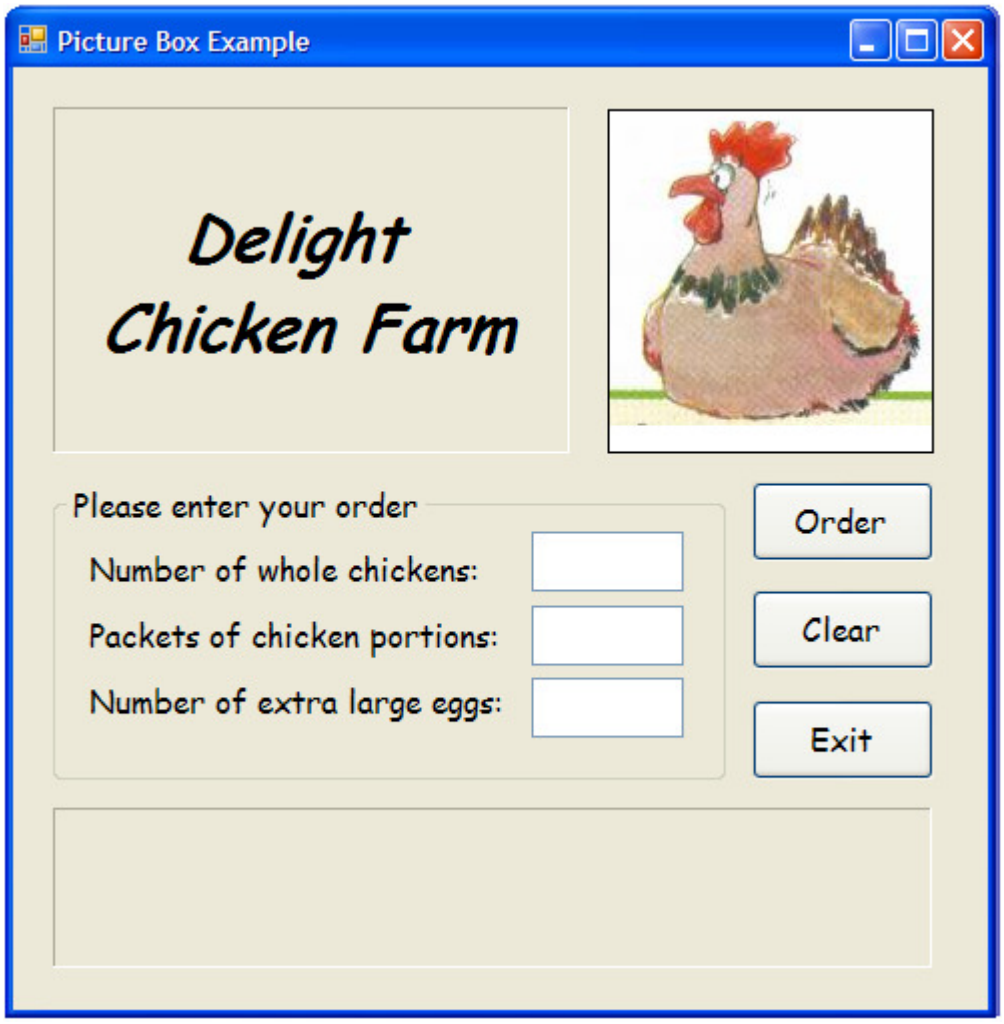


Figure 4-23

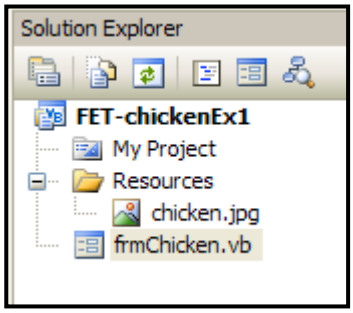


Figure 4-24

Remember, the Solution Explorer displays the projects in the current solution as well as the items in each project.

Our solution is named FET-chickenEx1 and it consists of one project folder. It also contains the files frmChicken.vb and now also the file chicken.jpg.

4.7.2 PROPERTIES MOST COMMONLY USED FOR A PICTURE BOX CONTROL

Once again, the first property to be set for every picture box is the name. Give every picture box a meaningful name in the **name property** according to the Hungarian notation that starts with pic.

Other properties to set could be any of the following:

Image: The image displayed in the picture box

BorderStyle: Controls what type of border the picture box should have

SizMode: Controls how the picture box will handle image placement and control sizing

The following five options are possible:

- StretchImage
- AutoSize
- CentreImage
- Zoom
- Normal

Once again, the best way to learn is to explore and to try out the different properties in order to see the effect and result of each.

4.7.3 GUI DESIGN GUIDELINES FOR A PICTURE BOX CONTROL

Keep the following in mind when placing picture boxes on the form:

- Graphics and color should be used sparingly in an interface.
- If you use an image in the interface, use a small one and place it in a location that will not distract the user.

4.8 A USER INTERFACE WITH A PICTURE AS A BACKGROUND

A user interface may contain a picture as a background. However, remember that a picture can be disturbing and can distract the user. The user interface should always be simple, user friendly and fulfill the purpose for the user to enter input data, to click on a button to activate an event to display certain output to the user.

In the following example we will just illustrate how you can insert a background image on a user interface and how the labels can be placed onto that in a transparent way.

The BackgroundImage property is currenty set to the default (none) as indicated in **Figure 4-25.**

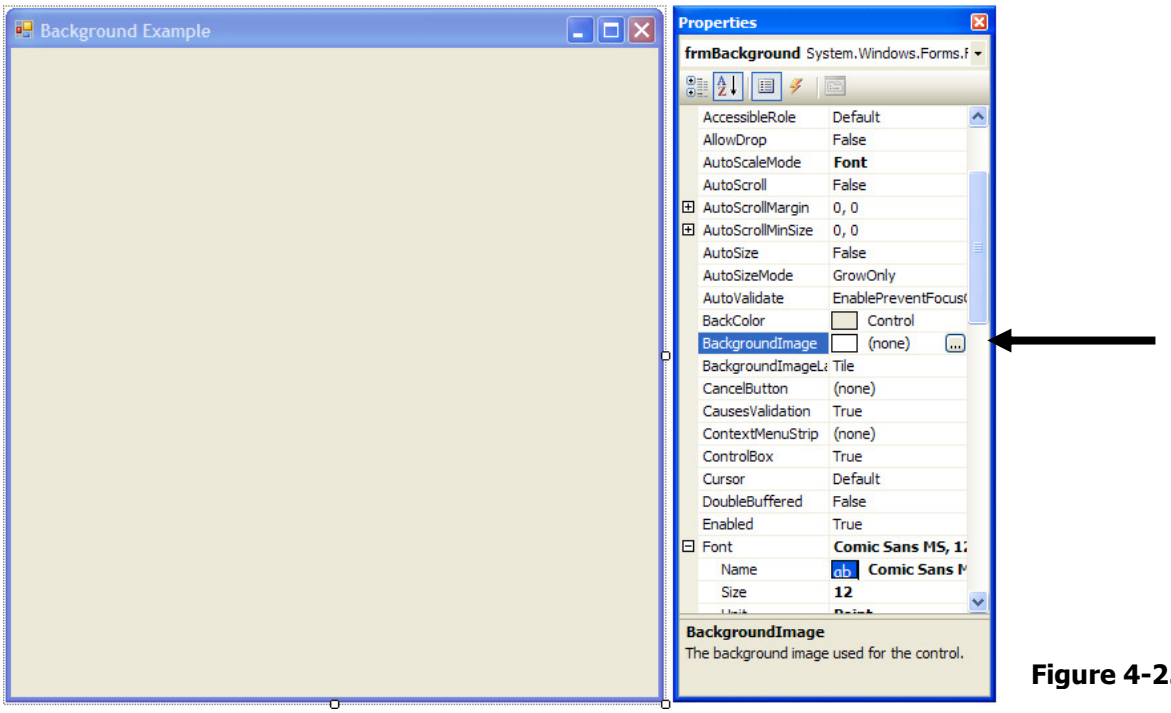


Figure 4-25

If we click on the BackgroundImage property as indicated in Figure 4-25, the Select Resource dialog box **(in Figure 4-26)** will appear. The file to be used as a background, can be selected in a similar way as the file that has been selected to be placed on the picture box in our previous example.

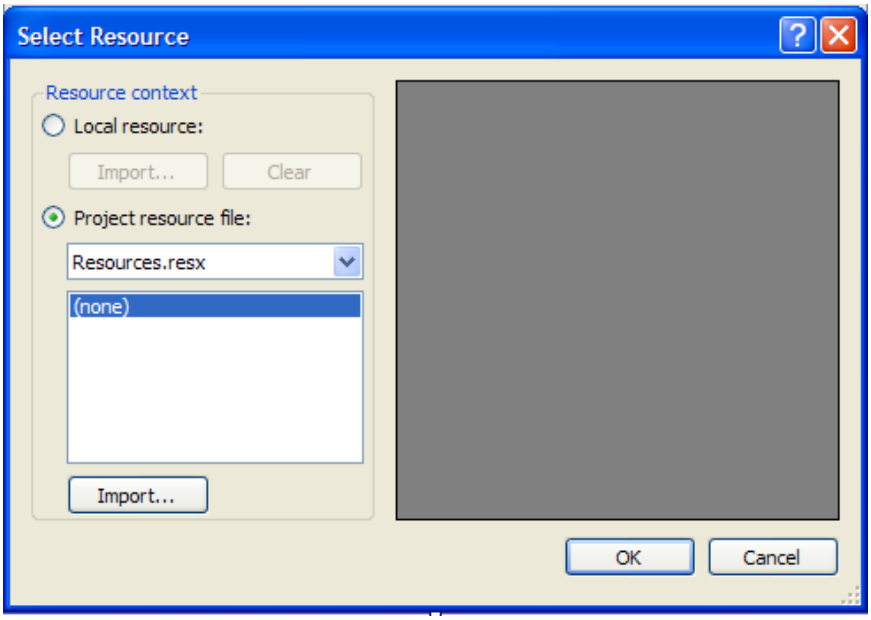


Figure 4-26

Figure 4-27 indicates how the file P1030043.JPG has been selected as background image for the user interface and this file has also been added to the Solution Explorer.

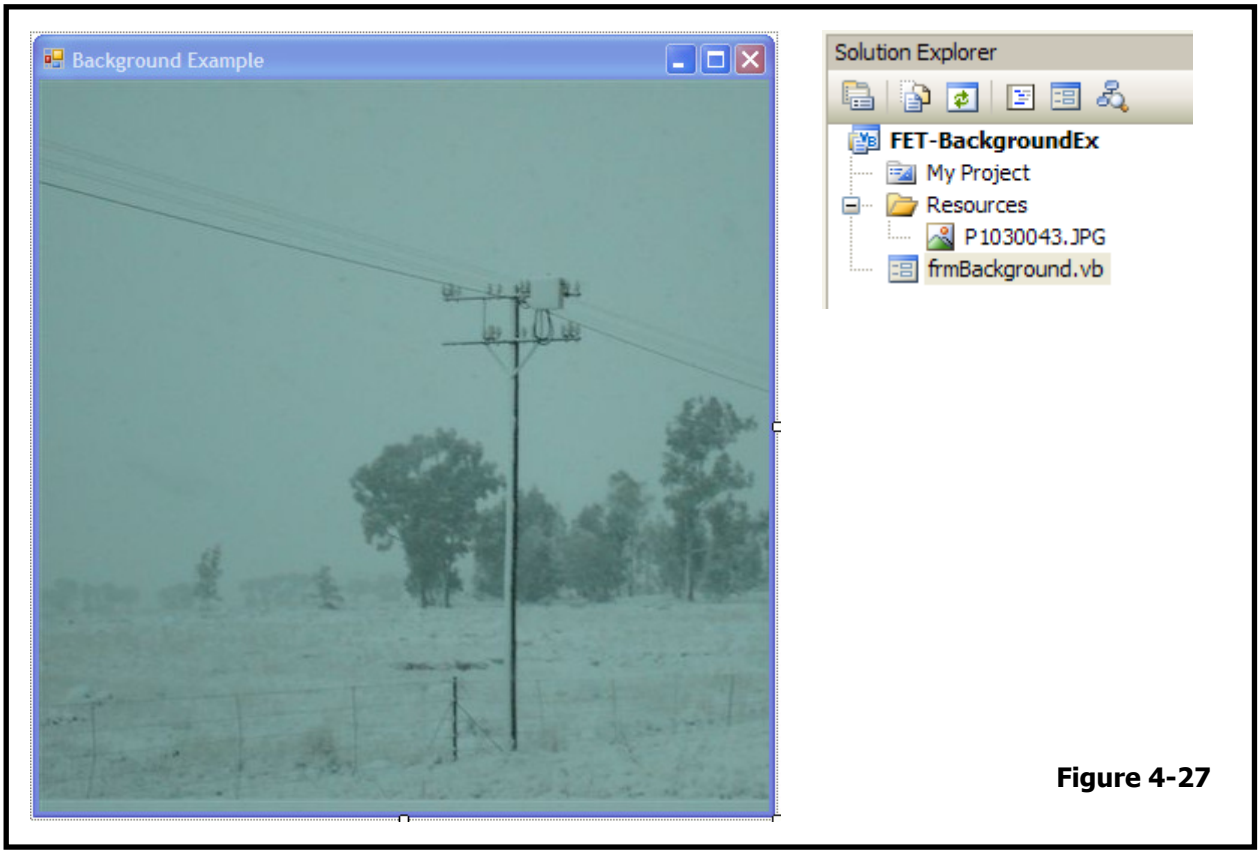


Figure 4-27

Any label placed on the form will have a border as indicated in **Figure 4-28**. If you don't want this effect on your form, you must set the BackColor property of the label to transparent after the Web option has been selected.
This is indicated in **Figure 4-29** and the result is indicated in **Figure 4-30**.



Figure 4-28

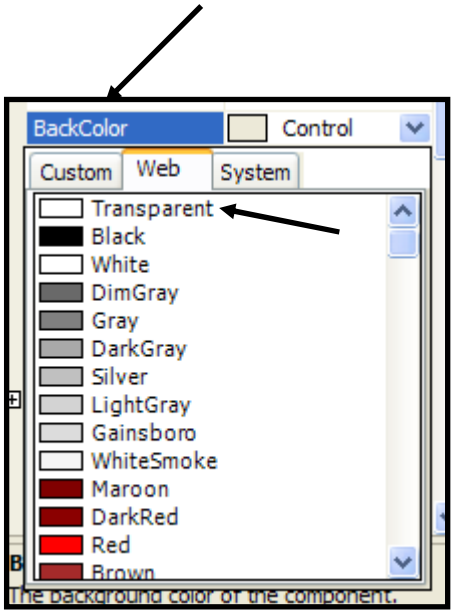


Figure 4-29

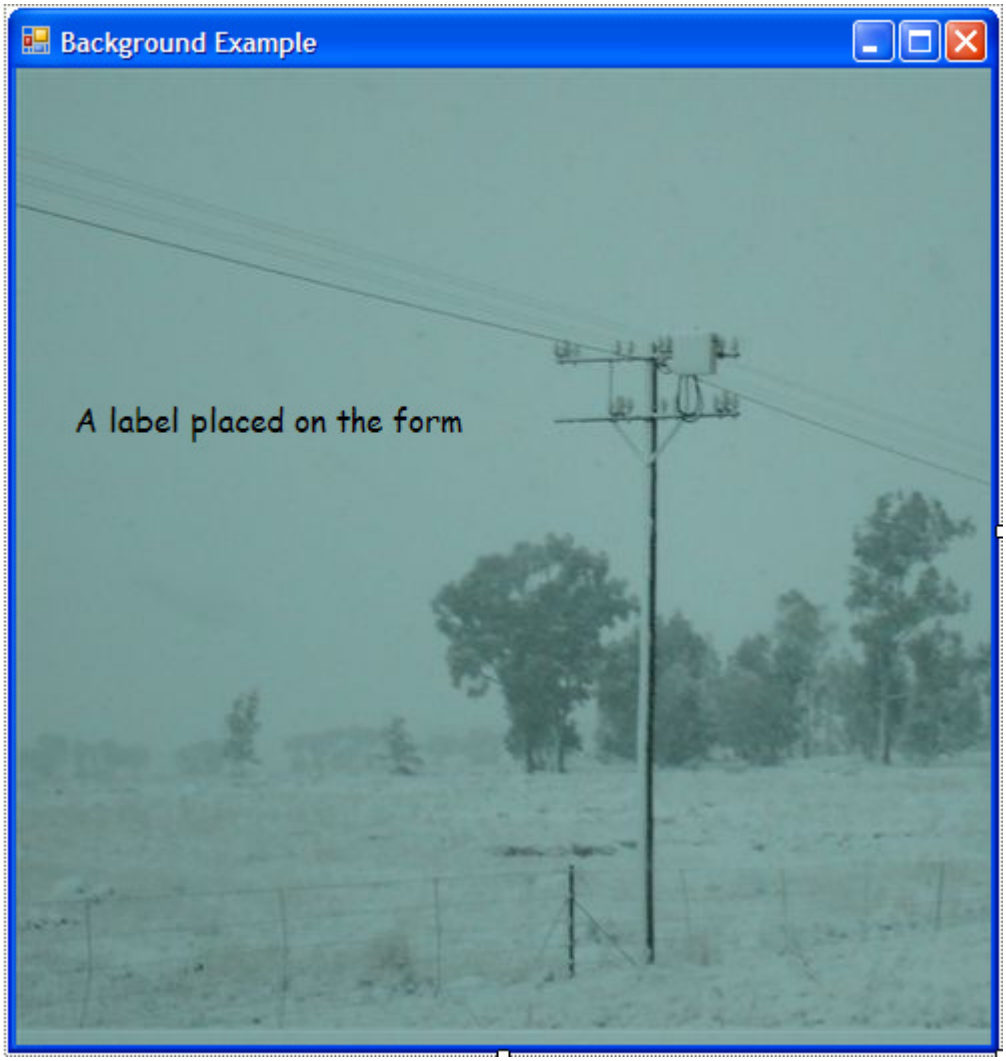


Figure 4-30

The back color of the label is now transparent and therefore no border for the label is shown.



Questions

Answer the following questions, based on the the user interface provided in **Figure 4-31**. This user interface is designed to enter the minimum and maximum temperature for a day and to display the average temperature.

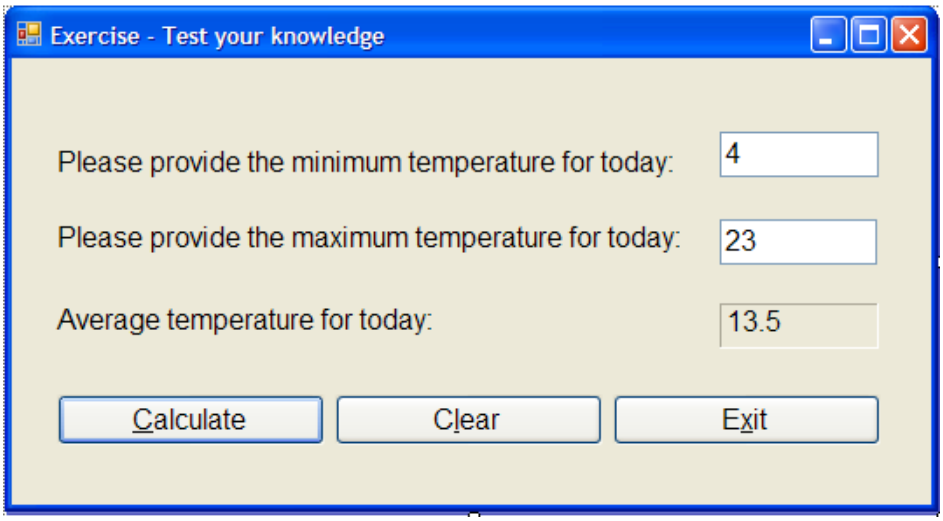


Figure 4-31

1. Another name for a user interface is a _____

a. Frame
b. Form
c. Facing
d. Feature
e. None of the above

2. This user interface contains the following number of components

a. 6
b. 9
c. 3
d. 4
e. None of the above

3. This user interface must contain code for _____ events

a. 6
b. 9
c. 3
d. 4
e. None of the above

4. In this user interface, the Calculate button is the _____ button

a. Accept button
b. Cancel button
c. Default button
d. First button
e. None of the above

5. In the given user interface, the value of the text property of the Clear-button is _____

a. Clear
b. 'Clear'
c. &Clear
d. C&lear
e. Clear

6. The text align property of all the buttons on the user interface is set to _____

a. Middle
b. Middle Centre
c. Middle Right
d. Middle Left
e. Centre

7. On the user interface provided, the following controls are used to enter the minimum and maximum temperatures

a. Text boxes
b. Labels
c. Buttons
d. Edit box
e. None of the above

8. Provide the value of the text property of the given user interface

_____

9. The average temperature is displayed on a _____ called lblAve.

a. Text box
b. Label
c. Button
d. Group Box
e. None of the above

10. True or False? Provide a reason for your answer. When the user has entered the minimum and maximum temperatures, the only way to display the average temperature is for the user to click on the Calculate-button.

_____

_____

_____

11. To add a control to the user interface, the _____ must be used.

a. Properties Window
b. Solution Explorer
c. Toolbox
d. View
e. All of the above

12. The following code is needed for the click event of the Exit button on the user interface

a. Close
b. Exit
c. Close()
d. Exit()
e. Quit()

13. If the Exit button has been selected as the cancel button of the user interface, the following actions will cause the application to terminate

a. Clicking on the Exit button
b. Pressing the Escape key
c. Pressing the alt key and the x-key
d. Any of the above
e. None of the above

14. True or False? Provide a reason for your answer. The user would be able to successfully enter a minimum and a maximum temperature of 0?

_____

_____

_____

15. Which one of the following variable names must be used in order to complete the following statement to display the answer?

lblAnswer.text = _____.ToString("N1")

a. ntAverage
b. strAverage
c. chaAverage
d. decAverage
e. a. or d.

CHAPTER 5
WRITE ALGORITHMS AND PROGRAMS IN SEQUENTIAL STEPS

5.1 INTRODUCTION

Any program can be written by using a combination of only three basic control structures i.e. sequence, selection and iteration.

When using the first of these structures i.e. the **sequence structure** all the steps are executed in sequence. Every step is executed once in sequential order. No step is left out and no step is repeated. Once all the steps are executed, the problem should be solved. But, if the problem is not solved, it indicates a logical error in the program. It may be that a step is written incorrectly or a step is left out.

OUTCOMES

When you have studied this chapter you should be able to do the following for a problem that can be solved with steps to be executed in sequential order:

- Identify all available input data and output needed.
- Draw an IPO-chart for the problem statement.
- Write an algorithm in pseusocode to solve the problem.
- Test the logic of the algorithm with test data in order to predict what the output will be and to see whether the output will be correct.
- Design a user interface that corresponds to the program planning.
- Code the program logic that corresponds to the algorithm for one of the buttons on the user interface.
- Add additional code for a button to clear all applicable controls on the user interface and code to terminate the application.
- Debug, test and run every program to see whether the output corresponds to the correct output that was prepared when the algorithm was tested with test data.

You should also be able to

- Use the concatenation character to combine output on a label
- Place output on two lines on a label by means of the ControlChars.Newline constant

5.2 PROGRAM PLANNING

As indicated before, we cannot start to write any program in a programming language unless we understand the problem. We must plan how to solve it by identifying the **input** and **output** and plan the **processing logic** via an algorithm.

In this section we will discuss the planning of various problems that may be solved by using steps to be executed in sequence.

5.2.1 FIRST EXAMPLE

Let's look at the following example.

Problem:

Enter the name and age of a learner on the keyboard and show the name and age, with a legend, on the screen of the computer.

The first step is to read and understand the problem very well and then identify

- the available data (input)
- the required results (output)

Draw the following table:

	Description	Type	Name of variable
Input:	name of learner	string	name
	age of learner	integer	age
Output:	name of learner	string	name (same as input)
	age of learner	integer	age

To do better and more comprehensive planning to enable the programmer to write a complete algorithm without errors, an **IPO-chart** (Input, Processing, Output) is completed. The IPO chart includes an overview of the processing steps.

I	P	O
name age (use names of variables)	Prompt to read input fields Enter input fields Show name and age on the screen	name age

The preparation is done to write a detailed algorithm that contains every step to solve the problem:

Algorithm:

ShowDetails

```
~       This program will enter the name and the age of the learner and
~       show it on the screen

        display  "Provide the name of the learner:"     ~display on new line
        enter name
        display "Provide the age of the learner:"       ~display on new line
        enter age

~       Show the details on the screen with appropriate legends

        display "The name of the learner is ", name     ~display on new line
        display "The age of the learner is ", age       ~display on new line
end
```

The algorithm will provide the following output on the screen:

```
Provide the name of the learner:  John
Provide the age of the learner:   18
The name of the learner is John
The age of the learner is 18
```

5.2.2 MORE EXAMPLES

5.2.2.1 EXAMPLE 2

Problem:

Enter the distance between two trees in kilometers. Display the kilometers and then calculate and display the distance in meters as well as the distance in centimeters.

	Description	Type	Name of variable
Input:	distance in km.	integer	kilometer
Output:	distance in km.	integer	kilometer
	distance in m.	integer	meter
	distance in cm.	integer	centimeter

I	P	O
kilometer	Prompt to read kilometer Enter kilometer Calculate meter and centimeter Display output fields	kilometer meter centimeter

Algorithm:

```
ConvertDistance
~       Convert km to meter en centimeter
        ~       Enter input data
        display "Please provide the distance in km."      ~ display on new line
        enter kilometer
        ~       Do the conversions
        meter = kilometer * 1000
        centimeter = meter * 100
        ~       Display the output
        display kilometer, " km is equivalent to ",
                meter, " m and to ", centimeter, " cm"   ~ display all on one line
end
```

Output:

```
Please provide the distance in km.  3
3 km is equivalent to 3000 m and to 300000 cm
```

If the last display statement is replaced by the following 3 display statements, the output will change as follows:

```
display "Distance = ", kilometer, " km"                    ~ display on a new line
display "Equivalent distance in cm = ", centimeter, " cm"      ~ on a new line
display "Equivalent distance in meter = ", meter, " m"         ~ on a new line
```

New output:

```
Please provide the distance in km.  3
Distance = 3 km
Equivalent distance in cm = 300000 cm
Equivalent distance in meter = 3000 m
```

5.2.2.2 EXAMPLE 3

Problem:

Enter the length and the width of a tennis court in meters and then calculate and display the circumference and the area of the tennis court.

	Description	Type	Name of variable
Input:	length of court	real	length
	width of court	real	width
Output:	circumference of court	real	circumf
	area of court	real	area

I	P	O
length width	Prompt for input fields Enter input fields Calculate circumference and area Display output fields	circumf area

Algorithm:

CalcTennisCourt

```
~       Calculate circumference and area of tennis court
        display "Please enter length:"              ~ display on new line
        enter length
        display "Please enter width:"               ~ display on new line
        enter width
        ~       Do calculations and display answers
        circumf = 2 * (length + width)
        area = length * width
        display  "The circumference of the tennis court is " , circumf ,
                        " meters"                   ~ display on new line
        display  "The area of the tennis court is " , area ,
                        "square meters"             ~ display on new line
end
```

Output:

```
Please enter length:   30
Please enter width:    10
The circumference of the tennis court is 80 meters
The area of the tennis court is 300 square meters
```

5.2.2.3 EXAMPLE 4

Problem:

Enter two integers on the keyboard and calculate and show the average of these two integers on the screen.

	Description	Type	Name of variable
Input:	first number	integer	num1
	second number	integer	num2
Output:	average	real	average

I	P	O
num1 num2	Prompt for num1 and num2 Enter num1 and num2 Calculate average Show average on the screen	average

Algorithm:

```
CalculateAverage

~ Prompt for and enter two integers
        display "Enter first number:"          ~ display on new line
        enter num1
        display "Enter second number:"         ~ display on new line
        enter num2

~ Calculate average
        average = (num1 + num2) / 2

~ Show average on the screen
        display "The average of the 2 numbers is ", average
                                               ~ display on new line
end
```

Output:

```
Enter first number: 24
Enter second number: 41
The average of the 2 numbers is 32.5
```

5.2.2.4 EXAMPLE 5

Problem:

Jason works as a part time assistant in a book shop and receives an hourly wage. Enter the number of hours he worked for a specific week, as well as the pay rate per hour. He has to pay R8 per week towards the recreation club. Calculate and display how much he has earned for the week.

	Description	Type	Name of variable
Input:	hours worked	integer	hours
	rate of pay	real	rate
Output:	weekly wage	real	wage

I	P	O
hours rate	Prompt for input fields Enter hours and rate Calculate wage Display wage on the screen	wage

Algorithm:

```
CalcWeeklyWage

~       Calculate and display the weekly wage for Jason
        ~       Enter input data
        display "Provide number of hours Jason worked:"     ~display on new line
        enter hours
        display "Provide rate of pay:"                      ~display on new line
        enter rate

        ~       Do calculations and display wage
        wage = hours * rate – 8
        display "Jason earned R", wage, " for the week"     ~display on new line
end
```

Output:

```
Provide number of hours Jason worked:  24
Provide rate of page:  12.00
Jason earned R280.00 for the week
```

5.2.2.5 EXAMPLE 6

Problem:

Tebogo earns a monthly salary and always divides it in the beginning of the month into different amounts according to her needs. She pays R450 for the rent of her room and the remainder of the money is divided as follows: 50% for food , 20% for clothes, 15% for transport, 5% for charity and the remaining amount for pocket money. Enter her monthly salary that is never less than R1800, on the keyboard and then calculate and display how much money she has available for the different categories.

	Description	Type	Name of variable
Input:	monthly salary	real	salary
Output:	room money	real	room
	food money	real	food
	clothes money	real	clothes
	transport money	real	transport
	charity money	real	charity
	pocket money	real	pocket

I	P	O
salary	Prompt for salary Enter salary Calculate different categories Display output fields	room food clothes transport charity pocket

Algorithm:

```
CalculateMoney
~       Divide Tebogo's money into different categories
        display "Provide Tebogo's monthly salary:"          ~ display on new line
        enter salary
        ~       do calculations
        remainder = salary – 450
        food = remainder * 0.5
        clothes = remainder * 0.2
        transport = remainder * 0.15
        charity = remainder * 0.05
        pocket = remainder – (food + clothes + transport + charity)
        ~       display results
        display "Tebogo divided her salary as follows:"     ~ display on new line
        display  "Room          R450.00"                    ~ display on new line
        display  "Food          R", food                    ~ display on new line
        display  "Clothes       R", clothes                 ~ display on new line
```

```
        display  "Transport        R", transport        ~ display on new line
        display  "Charity          R", charity          ~ display on new line
        display  "Pocket money     R", pocket           ~ display on new line
end
```

Output:

```
Provide Tebogo's monthly salary:  2000.00
Tebogo divided her salary as follows:
Room            R450.00
Food            R775.00
Clothes         R310.00
Transport       R232.50
Charity         R77.50
Pocket money    R155.00
```



Exercises

In each case, draw an IPO-chart and write an algorithm to solve the problem.

1. Enter the name and price of an item in a shop on the keyboard and then show the information on the screen of the computer with an appropriate legend.

2. Enter two integers. Calculate and show the sum, difference and product of these two integers on the screen.

3. Enter the number of rows of chairs in the city hall as well as the number of chairs per row. Calculate and display the total number of chairs in the hall.

4. Enter Kevin's length in feet and inches. Calculate and display his length in inches only (1 foot = 12 inches).

5. Sandra bought a number of the same items at a shop. Enter the number of items she bought and the unit price of an item. Calculate the amount due by multiplying the number by the unit price and then add 14% sales tax to the amount. Display the final amount she owes on the screen of the computer.

6. In a packet of sweets of the same kind, but of different colors, the following can be found: 35% of the sweets will be red; 25% will be yellow, 15% will be blue, 5% will be purple and the remaining sweets will be green. Enter the number of sweets (which will be a multiple of 20) in a big packet on the keyboard. Calculate and display how many of each of the color sweets can be found in this packet.

7. Jane is a typist who works freelance and she charges the following rates:

   A page typed single spacing R4.75
   A page typed double spacing R3.50

   The user must enter the number of pages she typed in single spacing and the number of pages she typed in double spacing. Calculate the amount due. She saves 15% of this amount to buy supplies. Display the nett amount she earned for her work.

5.3 SOLVE PROBLEMS IN VB.NET BY MAKING USE OF THE PROGRAM PLANNING

From the program planning in 5.2 we have learned that any computer application consist of input, processing and output. Any computer program needs input or data that the user must enter. These input values are needed to produce the required output. Without input data, the program will not have data to perform any calculations or to make any decisions in order to produce the correct output. (Think of using a calculator: no input = no answer).

In an **algorithm** we use a **display statement** to instruct the user that we need certain input data and then we use an **enter statement** to enter a value which must be placed in a variable. After the processing statements have been executed, the output is once again provided by means of a **display statement.**

From previous chapters we have discovered that in Visual Basic.NET we need to create a **user-interface** to communicate with the user. By means of the user interface we can instruct the user in a clear and user-friendly way what input values are needed and how it must be entered. A **label** is the component by which we can provide a user with instructions and information. The common input component on a user interface is a **text box**. After the user has entered all the input values, the values must be placed in specific variables in memory in order for the program to use these values during calculations and other processing steps.

From chapter 2 we have also learnt that Visual Basic.NET is an event-driven language. Therefore an **event** must be triggered before the output will be displayed. The most common way to trigger an event is by placing a **button** on the form and to provide logic via the code editor that will be executed every time the button is clicked by the user. A **label** is also used as an output component in order to display answers to the user. In a similar way that a legend appears in a display statement in an algorithm to identify the output, it can be placed on a label in VB.NET to identify output.

In order to implement any program planning in VB.NET, we must therefore first build an appropriate user interface for the problem. On each interface we must have a button to trigger an event to execute the program logic of the algorithm. However, in most cases there will also be a button to clear all components on a form to prepare it for new input as well as a button to terminate the application. There may be more buttons to trigger more events. However, in this early stage all our applications will contain only these 3 buttons.

In the design of the user interface, we can use various ways to display the output. One way is to provide a separate label for every output value. Alternatively, we can combine various output values onto one label. These methods will be illustrated.

5.3.1 FIRST VB.NET EXAMPLE

For this example, the problem planning in **example 5.2.2.5** will be used and the user interface will be designed to place every output value on a separate label.

5.3.1.1 USER INTERFACE WITH NAMED INPUT AND OUTPUT CONTROLS:

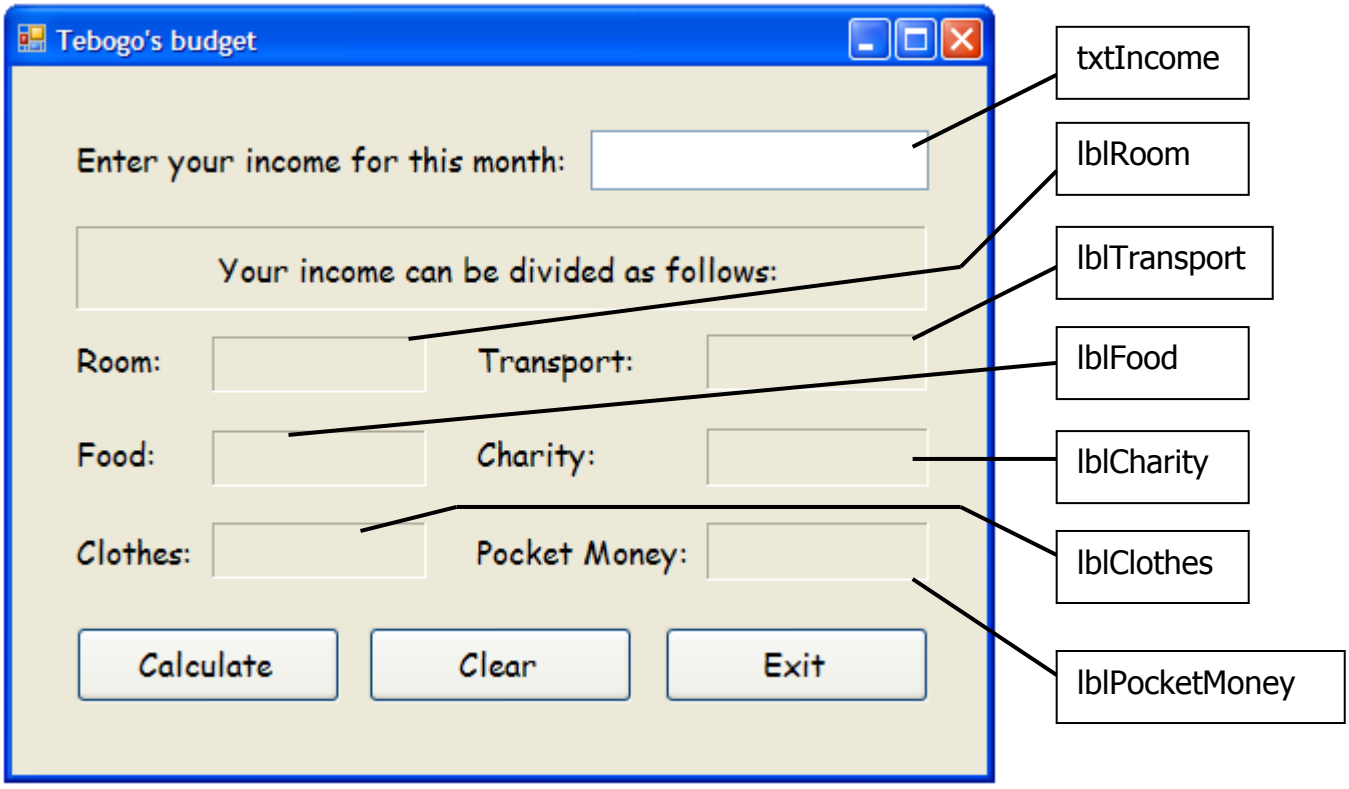


Figure 5-1

The names for the 3 buttons are btnCalc, btnClear and btnExit respectively and the name for the form is frmBudget.
When the user clicks on the button named btnCalc, the code in the code editor to be executed, must correspond to the logic planned in the algorithm. When the user

clicks on the button named btnClear, the text box to accept an input amount must be cleared and all the labels to display output answers must be cleared. When the button called btnExit is clicked, the application must be terminated.

5.3.1.2 CODE IN THE CODE EDITOR

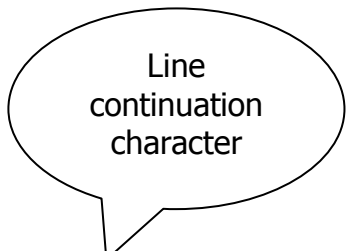
For this example we will assume that the user will enter valid input data – therefore a numeric value greater than or equal to 1800.00.

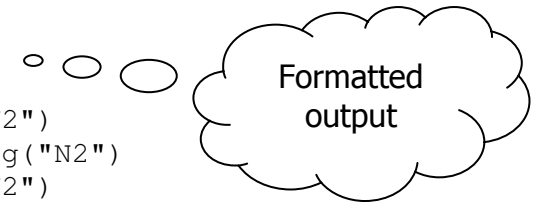
Take note of the use of the **line continuation character**. If one statement in a sub procedure is too long for one line and must continue to the next line, the first line must be ended with an underscore character (line continuation character).

Our example includes comments. **Comments** in a VB.NET program start with a single quote. The output values have also been **formatted** to display 2 numeric positions in each case as well as thousands separators, should that be necessary.

```
Public Class frmBudget

Private Sub btnCalc_Click(. . .) Handles btnCalc.Click
    'Declare input variable as decimal
     Dim decIncome As Decimal
    'Declare output variables as decimal
     Dim decRoom As Decimal
     Dim decFood As Decimal
     Dim decClothes As Decimal
     Dim decTransport As Decimal
     Dim decCharity As Decimal
     Dim decPocketMoney As Decimal
    'Declare intermediate variables
     Dim decRemainder As Decimal
    'Convert the input entered to the correct format and
    'store it in  the input variable
     decIncome = Convert.ToDecimal(txtIncome.text)
    'Sequential Processing to calculate output
     decRoom = 450
     decRemainder = decIncome - 450
     decFood = decRemainder * 0.5
     decClothes = decRemainder * 0.2
     decTransport = decRemainder * 0.15
     decCharity = decRemainder * 0.05
     decPocketMoney = decRemainder - (decFood + decClothes _
                                + decTransport + decCharity)
    'Display the output results on labels on the user
    'interface
     lblRoom.Text = decRoom.ToString("N2")
     lblFood.Text = decFood.ToString("N2")
     lblClothes.Text = decClothes.ToString("N2")
     lblTransport.Text = decTransport.ToString("N2")
     lblCharity.Text = decCharity.ToString("N2")
     lblPocketMoney.Text = decPocketMoney.ToString("N2")
End Sub
```





```
Private Sub btnClear_Click(. . .) Handles btnClear.Click
    txtIncome.Text = ""
    lblRoom.Text = ""
    lblFood.Text = ""
    lblClothes.Text = ""
    lblTransport.Text = ""
    lblCharity.Text = ""
    txtIncome.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
    Close()
End Sub
End Class
```

5.3.1.3 OUTPUT FOR THIS EXAMPLE IF TEBOGO EARNED R2366.55

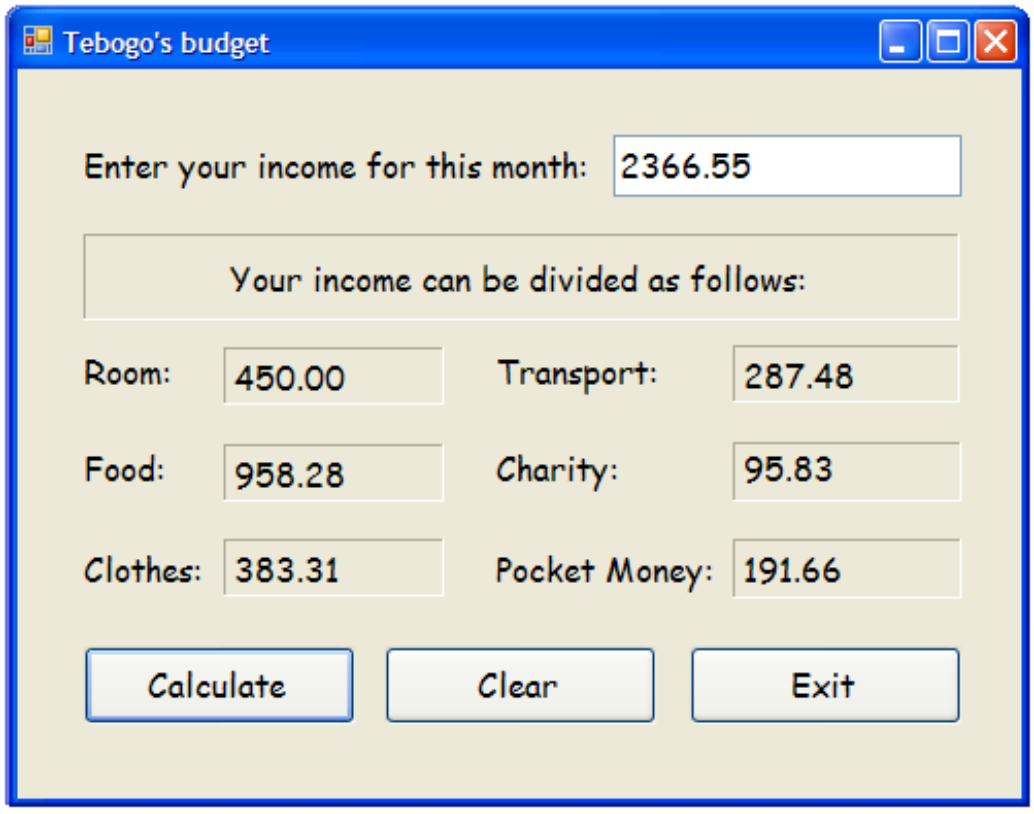


Figure 5-2

5.3.2 SECOND VB.NET EXAMPLE

For this example, the problem planning in **example 5.2.2.1** will be used and the user interface will be designed to combine all the output values on a single label.

5.3.2.1 USER INTERFACE WITH NAMED CONTROLS

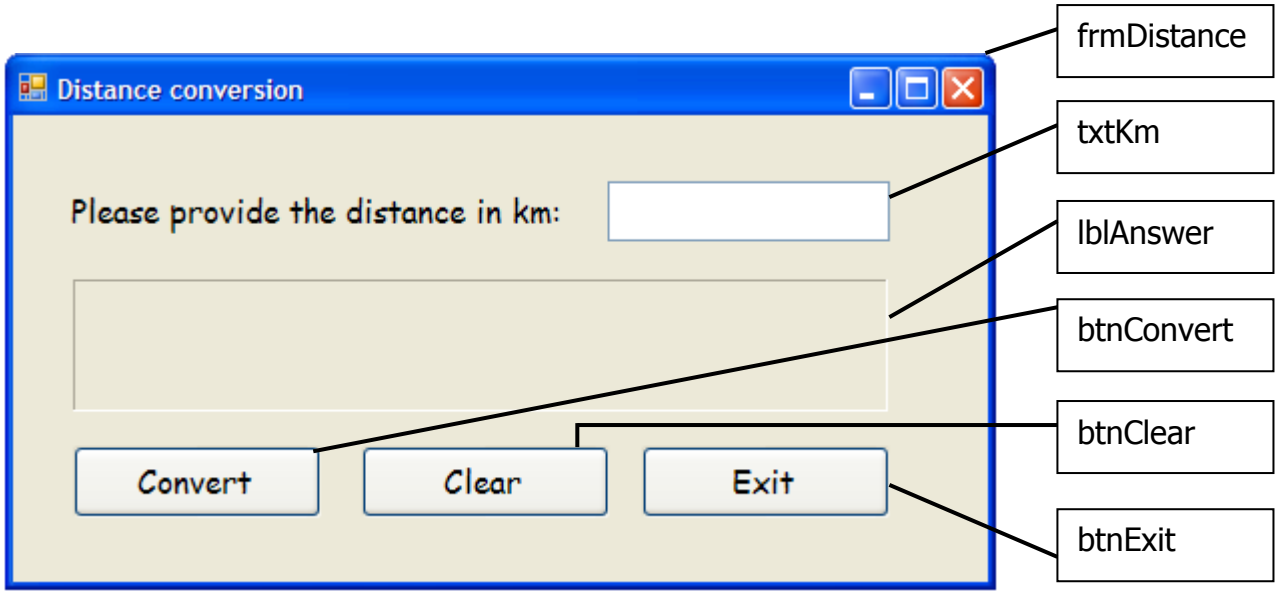


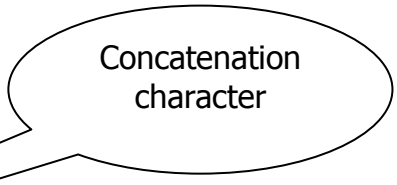
Figure 5-3

5.3.2.2 CODE IN THE CODE EDITOR

Once again, we will assume that the user will enter valid input data – therefore a numeric integer value.

In order to combine all three output values (the value entered in km and converted equivalent values in cm and meters) onto one label, a **concatenation character** is needed. The **ampersand character (&)** is used in VB.NET to act as a concatenation character. It will combine text and numeric output into one string output to be placed onto a label. The values to be concatenated can be either variables or literal strings. A **line continuation character** is once again needed in the statement to combine all the output fields onto the label.

```
Public Class frmDistance
Private Sub btnConvert_Click(. . .) Handles btnConvert.Click
    'Declare input variable, it will also be used as an
    'output variable
     Dim intKm As Integer
    'Declare output variables
     Dim intCm, intM As Integer
    'INPUT
     intKm = Convert.ToInt32(txtKm.Text)
    'PROCESSING
     intM = intKm * 1000
     intCm = intM * 100
    'OUTPUT
     lblAnswer.Text = intKm & " km is equivalent to " & _
                      intCm & " cm and to " & intM & " m"
End Sub
```



```
Private Sub btnClear_Click(. . .) Handles btnClear.Click
    txtKm.Text = ""
    lblAnswer.Text = ""
    txtKm.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
    Close()
End Sub

End Class
```

5.3.2.3 OUTPUT FOR THIS PROGRAM IF A DISTANCE OF 3 KM IS ENTERED

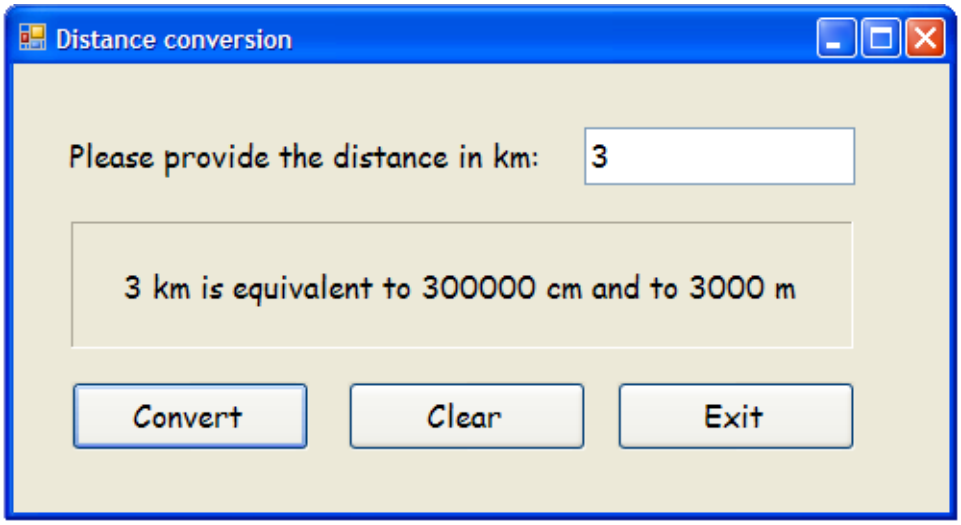


Figure 5-4

5.3.2.4 A different format for the output

It could be necessary that the output on the label must be displayed on more than one line. In such a case the **ControlChars.Newline** constant must be used. It will place the next output to be concatenated onto the label, on a new line on the label.

```
lblAnswer.Text = intKm & " km is equivalent to " & _
                 intCm & " cm and to " & intM & " m"
```

rewritten as:

```
lblAnswer.Text = "Distance = " & intKm & " km" & _
    ControlChars.Newline & "Equivalent distance in cm = " & _
    intCm & " cm" & ControlChars.Newline & _
    "Equivalent distance in meters = " & intM & " m"
```

will produce the following new output as indicated in Figure 5-5:

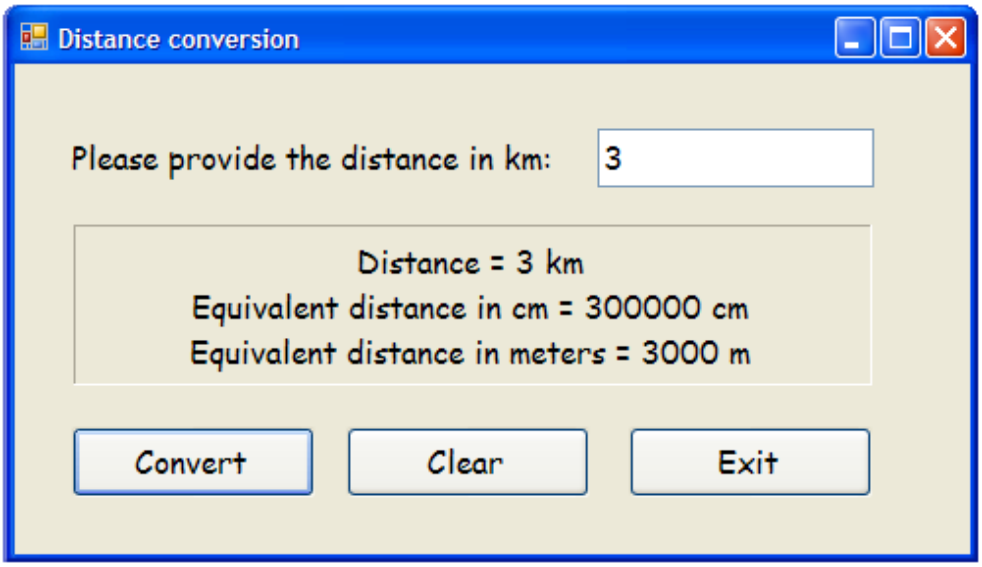


Figure 5-5



Exercises

1. In each of the following cases, choose a valid name for the variable and write a statement to declare this variable. Then write a statement to convert the value entered in the text box to the correct format and to store it in the declared variable.

1.1 Name of text box: txtName
Variable: Student name

1.2 Name of text box: txtAmount
Variable: An amount of an item in a shop

1.3 Name of text box: txtDept
Variable A department code – A, B or C

1.4 Name of text box: txtNumber
Variable Number of students in a class

2. Study the following declarations in a VB.Net application:

```
Dim intNum1 As Integer = 77
Dim intNum2 As Integer = 100
Dim intNum3 As Integer =  133
Dim decAverage As Decimal
```

Write the statement to display the following output on a label called lblOutput for **Figure 5-6** and **Figure 5-7**.

2.1

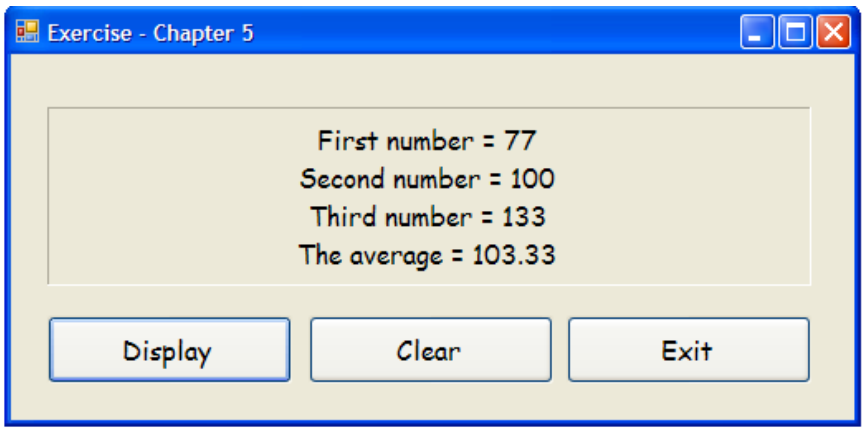


Figure 5-6

2.2

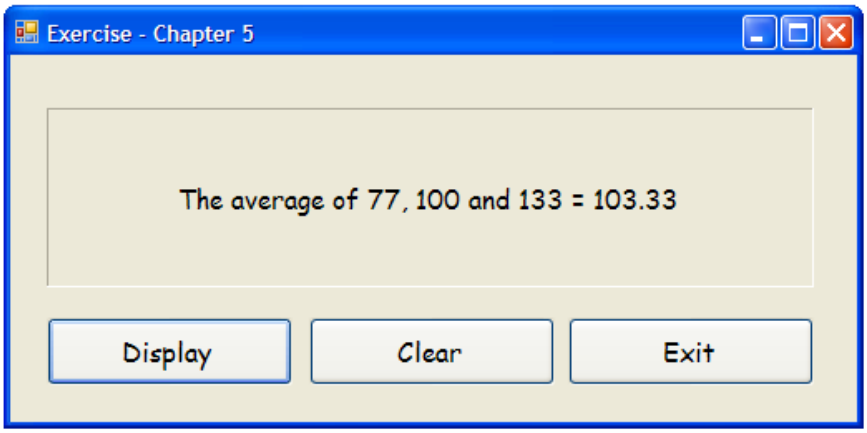


Figure 5-7



Write a VB.NET solution for each of the exercises 1 to 7 on page 96 and 97. In each case design your own user friendly interface. Use the IPO-chart and algorithm that you have designed in each case.

CHAPTER 6
THE SELECTION CONTROL STRUCTURE - PART 1

6.1 INTRODUCTION

The complexity of a problem may be of such a nature that it cannot be solved by simply applying statements in sequence. An algorithm may therefore include the possibility of providing for different options that may change the course of the flow of the statements in the algorithm. To enable the programmer to include this type of logic in an algorithm, a selection (decision) control statement may be included in the algorithm which is based on relational and logical operators.

OUTCOMES

When you have studied this chapter you should be able to

- determine the results of relational conditions,
- evaluate expressions containing logical operators,
- write a simple if-statement in pseudocode and in VB.NET in order to test a condition,
- write an if-then-else-statement in pseudocode and in VB.NET,
- prepare test data to test all possible conditions for a program that contains decision structures,
- write algorithms and corresponding VB.NET solutions containing
  - simple if-statement,
  - if-then-else-statements,
  - compound if-statements,
  - Boolean variables,
- know the difference between the AndAlso and And logical operators and between the OrElse and Or logical operators in VB.NET,
- know how to add a checkbox control to a form and how to use it in decision statements in the code editor,
- code defensively and to do data validation wherever possible,
- use the TryParse method in VB.NET to test for possible input and conversion errors,
- use the String.IsNullOrEmpty method to test for possible input errors,
- use a MessageBox.Show method to display error messages and other possible output messages,
- take actions after a message box has been displayed.

6.2 RELATIONAL OPERATIONS

Computers have the ability to compare two values and expressions that compare operands are called relational expressions. The outcome or result of such a comparison always yields a boolean value i.e. true or false.

Examples:

5 > 3 result: true
16 = 5 result: false

The following table contains the operands used in relational comparisons:

PSEUDOCODE SYMBOL	DESCRIPTION	EXAMPLE
=	Equal to	A = 5
<	Less than	number < 97
>	Greater than	totalAmount > 100
<=	Less or equal to	noStudents <= 50
>=	Greater or equal to	weight >= 75
<>	Not equal to	quantity <> 17

All these relational operators have an equal priority, but the mathematical operators have a higher priority than the relational operators.

Examples:

Determine whether the following expression is true or false:

1. b <= a – 3 where b = 5 and a = 7
   substitute:
   5 <= 7 - 3
   5 <= **4** false

2. m + 7 > n – 3 * p where m = 4, n = 12 and p = 5
   substitute:
   4 + 7 > 12 – 3 * 5
   4 + 7 > 12 - **15**
   11 > **-3** true



Exercises

Evaluate the following expressions:

1. a + t <> (a + b) * 4 where a = 4, b = 17 and t = 20
2. k ^ 2 <= m mod 5 + 29 – n where k = 5, m = 63, n = 7
3. (x \ 8 –2) + 3 > y / 12 mod 2 where x = 59, y = 96
4. d + 4 * g / 2 < g * 3 + 14 where d = 3, g = 24

6.3 LOGICAL OPERATIONS

A logical operator joins two Boolean expressions to yield a Boolean answer, either true or false.

Pseudocode Symbol	Description	Example	Result	Precedence
NOT	Yields the opposite	NOT True NOT False	False True	1
AND	Both expressions must be TRUE to yield TRUE	True AND True True AND False False AND True False AND False	True False False False	2
OR	Only one expression must be TRUE to yield TRUE	True OR True True OR False False OR True False OR False	True True True False	3

The order of precedence is indicated in the last column i.e. processing is done in the following order:

- NOT
- AND
- OR

When the same operator appears more than once in one expression, it is evaluated from left to right. When different operators are combined in one expression, all the mathematical operators will be processed first, then the relational operators and lastly the logical operators.

Examples:

Evaluate the following expressions:

1. D OR K AND NOT S where D = TRUE, K = FALSE, S = TRUE
   substitute:
   TRUE OR FALSE AND NOT TRUE
   TRUE OR FALSE AND **FALSE**
   TRUE OR **FALSE**
   TRUE

2. M > 7 AND D < S ^ 2 where M = 7, D = 8, S = 4
   substitute:
   7 > 7 AND 8 < 4 ^ 2
   7 > 7 AND 8 < **16**
   FALSE AND 8 < 16
   FALSE AND **TRUE**
   FALSE

3. D < (E + 7) AND G OR (A * C) >= (D – E + C)
   where A = 5, C = 12, D = -15, E = -17, G = FALSE
   substitute:
   -15 < (-17 + 7) AND FALSE OR (5 * 12) >= (-15 –(-17) + 12)
   -15 < **-10** AND FALSE OR (5 * 12) >= (-15 –(-17) + 12)
   -15 < -10 AND FALSE OR **60** >= (-15 –(-17) + 12)
   -15 < -10 AND FALSE OR 60 >= (-15 **+ 17** + 12)
   -15 < -10 AND FALSE OR 60 >= (**2** + 12)
   -15 < -10 AND FALSE OR 60 >= **14**
   TRUE AND FALSE OR 60 >= 14
   TRUE AND FALSE OR **TRUE**
   FALSE OR TRUE
   TRUE



Exercises

1. Evaluate the following expressions where A = 5, B = 7, C = 12, D = -4, E = TRUE and F = FALSE:

1.1 A * 3 >= B – 14 \ 3 AND F OR C <= D + 3
1.2 NOT F OR E AND D + C = A
1.3 (C – D) * 2 <> A + B OR E AND F OR B > D + B ^ 2
1.4 D = C MOD 5 AND NOT F OR NOT(A + D <= A – D)

2. Choose appropriate names for the variables used and write equations to calculate the required result:

2.1 The total of the 5 test marks of Cindy is stored in the variable totMark. Calculate the average test mark.
2.2 The area and length of a carpet is known. Calculate the width.
2.3 Determine the difference between the 2 integers stored in the variables numOne and numTwo.
2.4 Annah obtained a test mark stored in the variable test1 out of a possible 60. Convert this mark to a percentage.

6.4 THE SIMPLE IF-STATEMENT

This option can be explained by studying the following example:

If an employee has worked more than 45 hours during the past week, a bonus of R300 must be added to his wage.

Pseudocode:

```
~       test if the employee worked overtime
if hours > 45 then
        wage = wage + 300        ~  executed only if the condition is true
endif
```

Explanation of the above 3 lines:

```
if hours > 45  then
        This statement tests if the hours worked are more than 45
wage = wage + 300
        If the test is true (the employee has worked more than 45 hours),
        the previous wage is replaced by an increased wage of R300 more
endif
        This line ends the if-structure and the execution of statements will
        continue after the endif-statement regardless of the outcome of the if
        test
```

In the case where the employee has not worked for more than 45 hours, the execution of statements will continue after the *endif*. It would have skipped the statement to increase the wage.

The syntax of the if-statement in an algorithm:

```
if condition then
        statement(s)
endif
```

Example:

```
if mark >= 50 then
        display "Student has passed"
endif
```

Please note:

- There may be any number of statements in the *body* of the if-statement (between the **if** and the **endif**).
- The statements will only execute if the condition is TRUE
- The statement(s) within the if-statement is/are always indented to enhance the readability of the algorithm.
- The **if** and the **endif** are written in the same column while the body of the if statement is indented.
- The if-statement always renders a Boolean value i.e. the result is always TRUE or FALSE.

It is also possible to test whether a specific condition is not true by inserting the word NOT before the condition.

```
if NOT condition then
        statement 1
        :
        statement n
endif
```

Example:

```
if mark NOT < 50 then
        display "Student has passed"
endif
```

6.5 TESTING OF A PROGRAM

After a program has been written, the programmer may not assume that it is error-free and produces the desired results in a manner that is easy to understand and to read. It is very important to test this program for correctness and readability.

In order to accomplish the desired and correct results, the programmer must compose a set of test data to test the program. The test data must include all possibilities of correct and incorrect data. It must be compiled in such a way that it resembles the possible real data. At this stage we are only going to use simple test data, but more complicated programs need more test data to ensure that no errors occur.

The program can be tested by using a trace table (desk checking) and then also by processing the program using the selected test data.

Study the following program:

Problem:

Lerato bakes cakes that she sells in slices. She cuts 10 slices per cake. She calculates the amount spent on the ingredients, electricity and packaging material. She adds 30% to the amount spent and then she calculates the price of a slice of cake. The user will enter the amount spent per cake and then the program will calculate and display the price of a slice. She always sells the slices in full rand

amounts i.e. if the calculated price is R3.25, then the selling price is R4. In other words, cents are rounded up to the next rand.

	Description	Type	Name of variable
Input:	cost of cake	real	expense
Output:	price of slice	integer	slicePrice

I	P	O
expense	Prompt for expense Enter expense Calculate slicePrice Display slicePrice	slicePrice

Algorithm:

```
CalcCakePrice

~       Calculate the selling price of cake

        ~       enter input data
        display "Provide the cost of the cake:"          ~ display on new line
        enter expense

        ~       add profit and calculate price per slice
        cakePrice = expense + expense * 0.3    ~ intermediate var cakePrice
        slicePrice = cakePrice \ 10
        rem = cakePrice mod 10          ~ calc if any cents
        if rem > 0  then                ~ test if any cents
                slicePrice = slicePrice + 1
        endif

        ~       display the price of a slice of cake
        display "The price of a slice of cake is R" , slicePrice
                                                         ~ display on new line
end
```

Test the program by using a manual calculation:

Use expense = R18.60
Increase by 30%: R24.18
Gross price / slice R2.418

Integer division results in R2 per slice of cake
Because there is a remainder, R1 will be added to the result to give the selling price for a slice of cake.

Selling price / slice R3

Trace table:

The instruction is listed in the leftmost column, followed by the variables used, the outcome of the if-statement and lastly the output as shown in the following table:

Instruction	expense	cakePrice	slicePrice	rem	Outcome of if	Output
display						Provide the cost of the cake
enter	18.60					
calculate		24.18				
calculate			2			
calculate				.418		
if					TRUE	
calculate			3			
display						The price of a slice of cake is R3

Output:

```
Provide the cost of the cake: 18.60
The price of a slice of cake is R3
```

Whenever an if-statement is involved, test data should be prepared in such a way to test if all possible outcomes of the if-statement work in a correct manner.

6.6 EXAMPLES OF SIMPLE IF-STATEMENTS

In this section we will discuss the planning of various problems that may be solved by using simple if-statements.

6.6.1 EXAMPLE 1

Problem:

John bought a number of pencils at a given price. The user must be asked to enter values for these variables. If he buys 25 or more pencils, he receives a discount of 7%. Calculate and display the amount due by John.

	Description	Type	Name of variable
Input:	number of pencils	integer	number
	price of one pencil	real	price
Output:	amount due	real	amount

I	P	O
number price	Prompt to read input fields Enter input fields Calculate amount Display amount	amount

Algorithm:

CalcAmount

~ Calculate the amount due for pencils

~ Input
display "Provide the number of pencils bought:" ~ display on new line
enter number
display "Provide price of one pencil:" ~ display on new line
enter price

~ Processing
amount = number * price

~ test if John bought 25 or more pencils
if number >= 25 then
amount = amount – (amount * 0.075)
~ the calculation could also have been amount = amount * 0.925
endif

~ Output
display "The amount due is R" , amount ~ display on new line
end

Test the program twice:

Number = 16, price = R3.50 (less than 25 pencils)
Number = 30, price = R2.25 (equal or more than 25 pencils)

Output:

```
Provide the number of pencils bought: 16
Provide price of one pencil: 3.50
The amount due is R56.00

Provide the number of pencils bought: 30
Provide price of one pencil: 2.25
The amount due is R62.44
```

6.6.2 EXAMPLE 2

Problem:

The mark obtained by a student is entered. If the student has passed (50 or more marks), the mark must be displayed and a message that indicates that the student passed. But if the mark is less than 50, the mark and a message that indicates that the student has failed, must be displayed.

	Description	Type	Name of variable
Input:	Mark obtained	integer	mark
Output:	Mark obtained	integer	mark
	Outcome of if	string	message

I	P	O
mark	Prompt to read mark Enter mark Determine message Display output fields	mark message

Algorithm:

PrepareResult

```
~       Prepare and display result of student
        display "Provide mark obtained: "                   ~ display on new line
        enter mark
        ~       initialize message
        message  =  "pass"
        ~       test result
        if mark < 50  then                  ~ could also have tested mark <= 49
                message = "fail"
        endif
        ~       display result
        display "The mark is ", mark, "%  Result = ", message ~ display on new line
end
```

Test the program twice with a mark of 45 and 69 respectively.

Output:

```
Provide mark obtained: 45
The mark is 45%  Result = fail

Provide mark obtained: 69
The mark is 69%  Result = pass
```



Exercises

1. Henry rents a trailer to move his furniture to his new house. The basic cost is R200 per day and he also has to pay a specific amount that must be entered by the user for the kilometers traveled. The user also has to enter the kilometers. If he traveled for less than 50 kilometers an additional surcharge of 5% is payable on the amount due for the distance. However, if he used the trailer for more than 400 kilometers, he receives a discount of 12% on the amount due for the distance. Write an algorithm to calculate and display the amount due. (Hint: use 2 different if-statements in your algorithm).

2. Write an algorithm for the following: Jessie and Sally want to buy a large pizza to share. Enter the cost of the pizza and the amounts that Jessie and Sally each have in their purses. If they have enough money, display a message that indicates that they can buy the large pizza. However, if the do not have enough money, display a message that they have to buy a small pizza.

Prepare test data for both examples to test all possibilities, and draw a trace table for each.

6.7 THE IF-STATEMENT IN VB.NET

In VB.NET a simple if-then-statement can be coded as follows:

```
If condition Then
     Statements
End If
```

Eg.

```
If intNumber > 20 Then
     decDiscount = 0.05D
End If
```

Once the condition is entered and the enter key pressed, the compiler will insert the **Then** and the **End If** automatically. It is therefore only necessary to include the statement or statements for the body of the if-statement. The compiler will also automatically indent the code properly.

6.8 A SOLUTION IN VB.NET WITH AN IF-STATEMENT

For this example, the problem planning in **example 6.6.1** will be used.

6.8.1 USER INTERFACE WITH NAMED CONTROLS

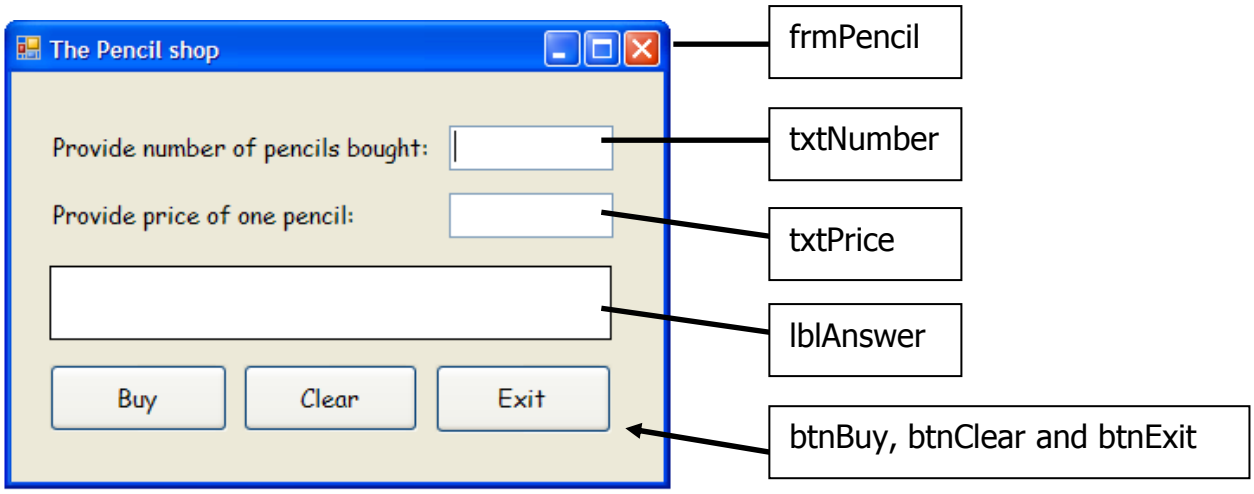


Figure 6-1

6.8.2 CODE IN THE CODE EDITOR

```
Option Explicit On
Option Strict On

Public Class frmPencil

Private Sub btnBuy_Click(. . .) Handles btnBuy.Click
   Dim intNumber As Integer
   Dim decPrice As Decimal
   Dim decAmount As Decimal
   intNumber = Convert.ToInt32(txtNumber.Text)
   decPrice = Convert.ToDecimal(txtPrice.Text)
   decAmount = intNumber * decPrice
   If intNumber >= 25 Then
      decAmount = decAmount - (decAmount * 0.075D)
   End If
   lblAnswer.Text = "The amount due = R" & _
                    decAmount.ToString("N2")
End Sub

Private Sub btnClear_Click(. . .) Handles btnClear.Click
   txtNumber.Text = ""
   txtPrice.Text = ""
   lblAnswer.Text = ""
   txtNumber.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
   Close()
End Sub

End Class
```

6.8.3 OUTPUT

Figure 6.2 shows the output if 16 pencils were bought at R3.50 each (in this case the outcome of the if-statement was false and no discount was granted)

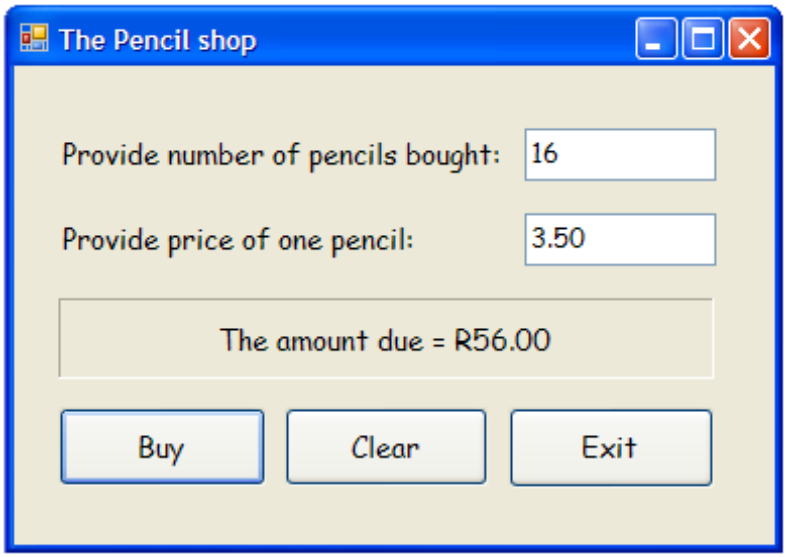


Figure 6-2

Figure 6.3 shows the output if 30 pencils were bought at R2.25 each (in this case the outcome of the if-statement was true and a discount of 7.5% was granted)

30 x 2.25 = 67.50
7.5% of 67.50 = 5.06
67.50 – 5.06 = 62.44

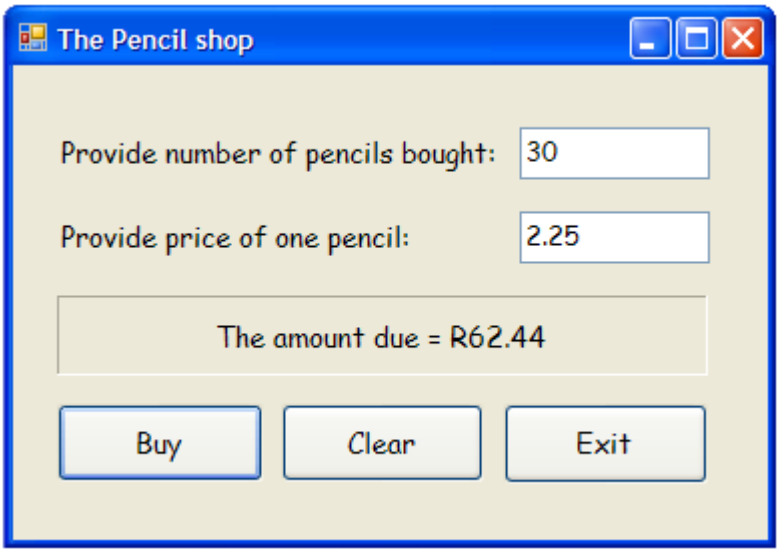


Figure 6-3

Figure 6.4 shows the same outcome, but the user interface was designed with a background image of a pencil. Although the background is not that dominant, it could be distracting. Decide for yourself.

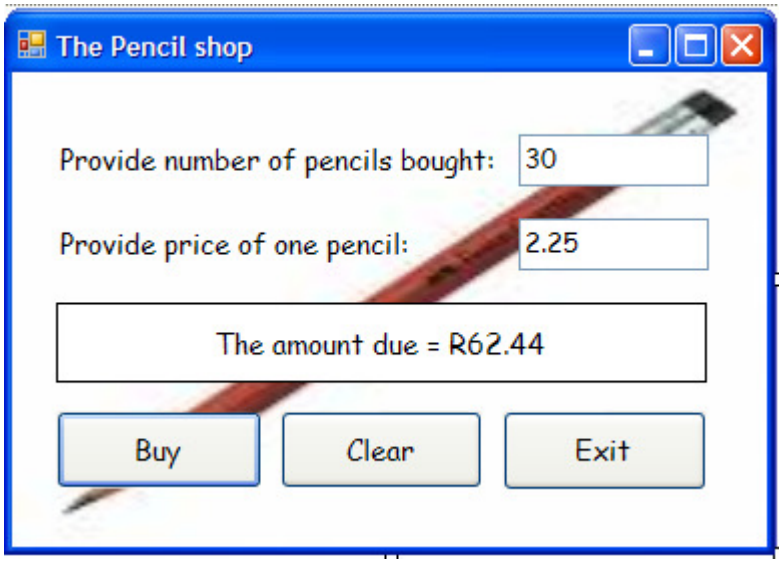


Figure 6-4

6.9 THE IF-THEN-ELSE-STATEMENT

In the previous paragraphs we have tested a specific condition and if the condition proves to be TRUE, we processed certain statements. It is however possible that we need to process other statements when the condition is not TRUE but FALSE.

All the employees of a company receive an increase of 8%, but the employees in department A (that performed the best) receive an increase of 10%. This can be accomplished by means of an if-statement that contains an else-clause as indicated in the following example.

```
if department = "A" then
        increase = salary * 0.08     ~  executed if the condition is true
else
        increase = salary * 0.1      ~  executed if the condition is false
endif
salary = salary + increase
```

6.10 EXAMPLES OF IF-THEN-ELSE-STATEMENTS

In this section we will discuss the planning of various problems that may be solved by using simple if-statements.

6.10.1 EXAMPLE 1

Problem

Angelina and Freedom took part in the final of a competition. A rule of the competition is that judges may not assign equal points to competitors at this stage. Enter the number of points that each competitor obtained. Compare the points and determine who the winner is. Display the name and the number of points of the winner on the screen.

	Description	Type	Name of variable
Input:	Angelina's points	integer	pointA
	Freedom's points	integer	pointF
Output:	Winner's points	integer	pointW
	Winner's name	string	winner

I	P	O
pointA pointF	Prompt for points Enter points Determine winner Display winner and points	winner pointW

Algorithm:

DetermineWinner

```
~       Determine the winner of the competition
        ~       Enter input data
        display "Provide point for Angelina:"          ~ display on new line
        enter pointA
        display "Provide point for Freedom:"           ~ display on new line
        enter pointF

        ~       find the winner
        if pointA > pointF then
                pointW = pointA
                winner = "Angelina"        ~ more than one statement in body of if
        else
                pointW = pointF
                winner = "Freedom"
        endif

        ~       display result
        display "The winner is ", winner               ~ display on new line
        display " and the winning points:  ", pointW
end
```

Test the program - Angelina obtained 175 points and Freedom obtained 179 points

Output:

```
Provide points for Angelina: 175
Provide points for Freedom: 179
The winner is Freedom and the winning points:  179
```

6.10.2 EXAMPLE 2

Problem

Dennis loves reading books and you have been asked to calculate how many days he will need to read a given book by writing an algorithm to solve the problem. Enter the number of pages in the book and the number of pages that he is able to read per day. Display how many days he needs to read the book. You are also asked to determine if it is possible for him to read 5 books with the same number of pages during his 14 days of holidays. Display this answer.

	Description	Type	Name of variable
Input:	Pages per book	integer	bookPages
	Pages per day	integer	dayPages
Output:	Days per book	real	bookDays
	Result of if	string	message

I	P	O
bookPages dayPages	Prompt for input fields Enter input fields Calculate bookDays Determine message Display output fields	bookDays message

Algorithm:

ReadingSpeed

```
        ~       Enter input data
        display "Provide number of pages per book: "        ~ display on new line
        enter bookPages
        display "Provide pages Dennis can read per day: "   ~ display on new line
        enter dayPages
```

```
        ~       calculate days per book and days to read 5 books
        bookDays = bookPages / dayPages
        days = bookDays * 5         ~ an intermediate variable is used (not I or O)
        ~       test if 5 books can be read during holidays and display output
        display "Dennis takes ", bookDays, "  days per book of ", bookPages, " pages"
        if days <=  14 then
                message = "Dennis can read 5 of these books during the holidays"
        else
                message = "It is not possible for Dennis to read 5 of these books
                          during the holidays"
        endif
        display message      ~ display on new line
end
```

Test the program:

Book has 120 pages and 333 pages
Dennis can read 50 pages per day

Output:

```
Provide number of pages per book: 120
Provide pages read per day: 50
Dennis takes 2.4 days per book of 120 pages
Dennis can read 5 of these books during the holidays

Provide number of pages per book: 333
Provide pages read per day: 50
Dennis takes 6.7 days per book of 333 pages
It is not possible for Dennis to read 5 of these books during the holidays
```



Exercises

1. In each of the following cases, write an algorithm to solve the problems:

1.1 The Great College offers 2 different courses in Programming that are offered as self study courses while the lecturer is assisting. The student can decide beforehand how many weeks he/she wants to study. The code of the first course is an A, while the code of the second course has a code of B. You may assume that the user will only enter an A or a B. The price for course code A is R234 per week, while the course for the other course is

R287.50 per week. The user has to enter the course code and the number of weeks of study. Calculate and display the course code and total price of the course on the screen.

2. Write only the if-statement for the following. You may choose your own names for the variables where the names are not specified:

2.1 The weights of 2 dogs, Fluffy and Terry, have been entered. Display the name of the dog that weighs more than the other dog. You may assume that the weights will not be the same

2.2 An employee's salary *empSalary* has been entered. Determine and display a message to indicate if this person needs to pay income tax. Only people with an income of more than R3000 per month, has to pay tax.

2.3 Tom decided that he must have one shirt for every pair of pants. Determine if this is true and display an appropriate message.

2.4 Josie wants to go to the movies, but is only allowed to go if she has done her homework (use a Boolean variable) and has R10 to pay for the ticket. Display a message to indicate if she can go or not.

2.5 Test the mark of a student. If the mark is more than 47, increase the mark by 2.5%, otherwise decrease the mark by 5.7%. Display the new mark.

2.6 Values for the variables A and B are given. Both contain real numeric values. If the value of A is greater than 17 and less than 47, swap the values of A and B, but if the value of A is either 5 or 87, increase the value of A by 20 and decrease the value of B by 5. Hint: Make use of a temporary variable if it is necessary to swap the values.

6.11 THE IF-THEN-ELSE-STATEMENT IN VB.NET

In VB.NET an if .. then .. else statement can be coded as follows:

```
If condition Then
     Statement(s)
Else
     Statement(s)
End If
```

Eg.

```
If intNumber > 20 Then
     decDiscount = 0.075D
Else
     decDiscount = 0.05D
End If
```

As mentioned before, once the condition is entered and the enter key pressed, the compiler will insert the **Then** and the **End If** automatically. However, the user must add the **Else**. The compiler will automatically indent the code properly when the body is entered.

6.12 A SOLUTION IN VB.NET WITH AN IF-THEN-ELSE STATEMENT

For this example, the problem planning in **example 6.10.2** will be used.

6.12.1 USER INTERFACE WITH NAMED CONTROLS

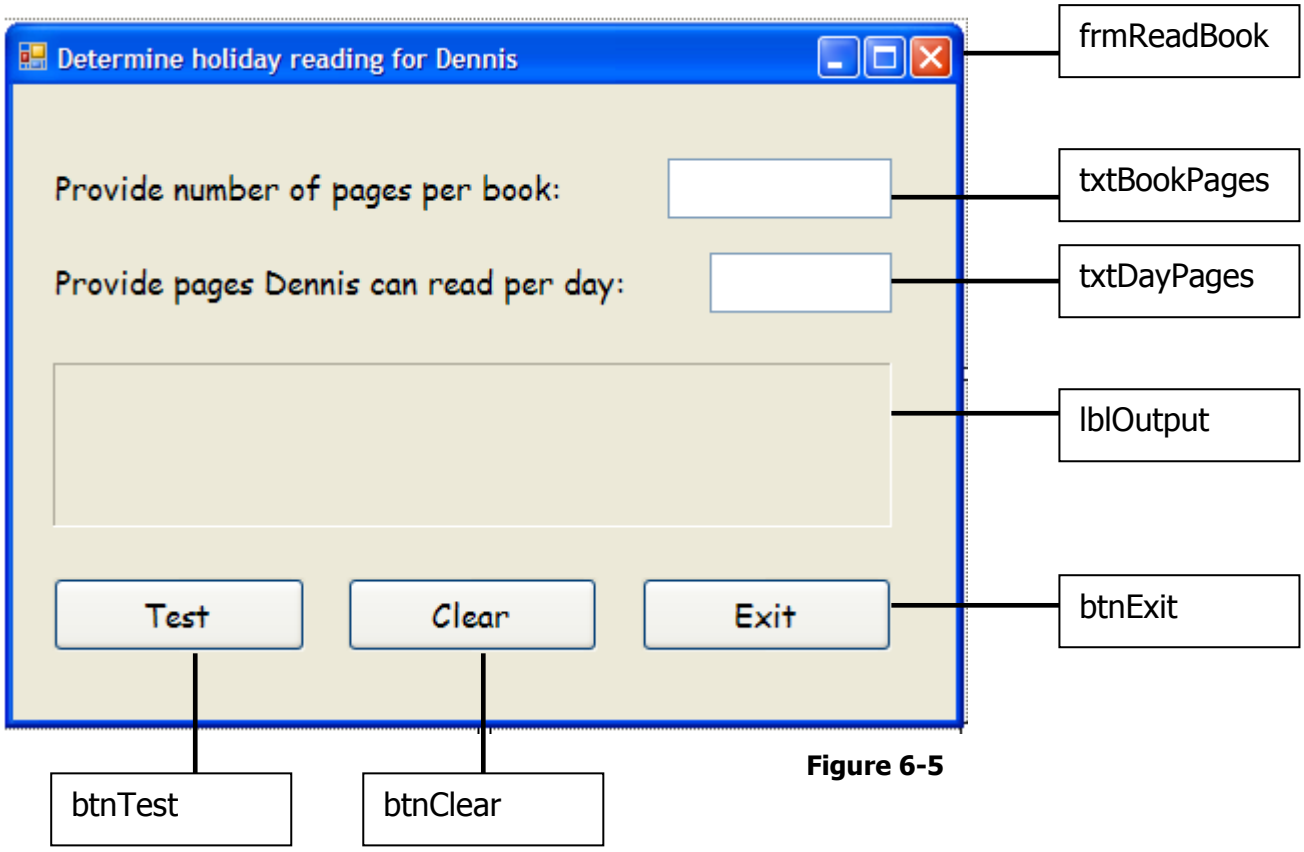


Figure 6-5

6.12.2 CODE IN THE CODE EDITOR

Figure 6-6 indicates the code in the code editor for the Test-button:

```
Private Sub btnTest_Click(. . .) Handles btnTest.Click
    Dim intBookPages, intDayPages As Integer
    Dim decBookDays As Decimal
    Dim decDays As Decimal
    Dim strMessage As String
    intBookPages = Convert.ToInt32(txtBookPages.Text)
    intDayPages = Convert.ToInt32(txtDayPages.Text)
    decBookDays = Convert.ToDecimal(intBookPages / intDayPages)
    decDays = decBookDays * 5D
    lblOutput.Text = "Dennis takes " & decBookDays.ToString("N1") & _
                     " days per book of " & intBookPages & " pages"
    If decDays <= 14 Then
        strMessage = "Dennis can read 5 of these books during the holidays"
    Else
        strMessage = _
        "It is not possible for Dennis to read 5 of these books during the holidays"
    End If
    lblOutput.Text = lblOutput.Text & ControlChars.NewLine & strMessage
End Sub
```


Figure 6-6

6.12.3 OUTPUT

Figure 6-7 indicates the output if the book contains 120 pages and Dennis can read 50 pages per day whereas **Figure 6-8** indicates the output if the book contains 333 pages and Dennis can read 50 pages per day.

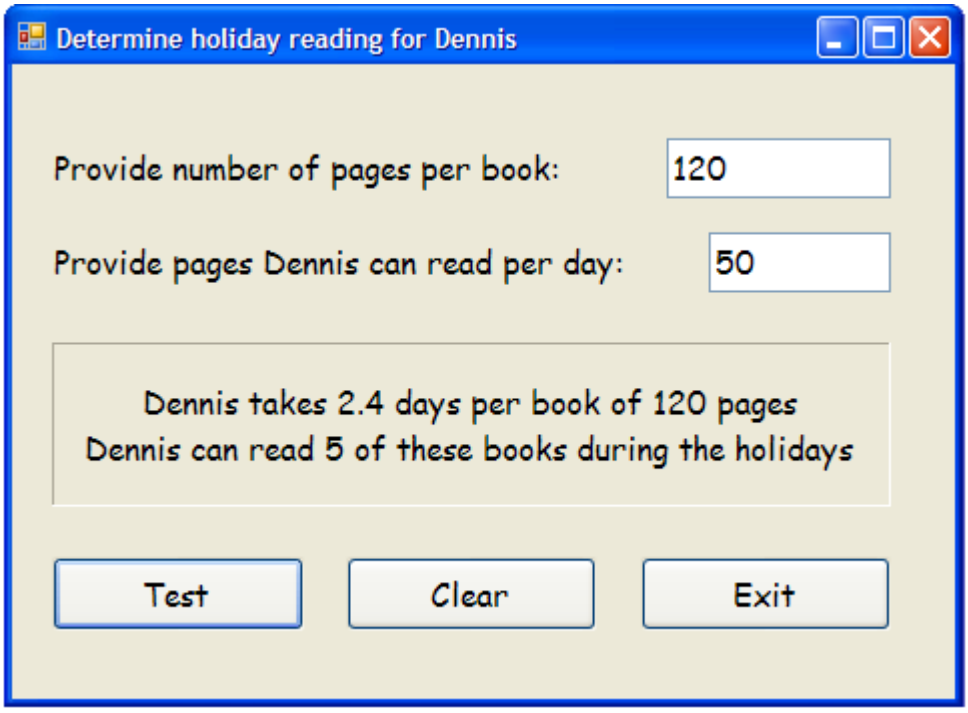


Figure 6-7

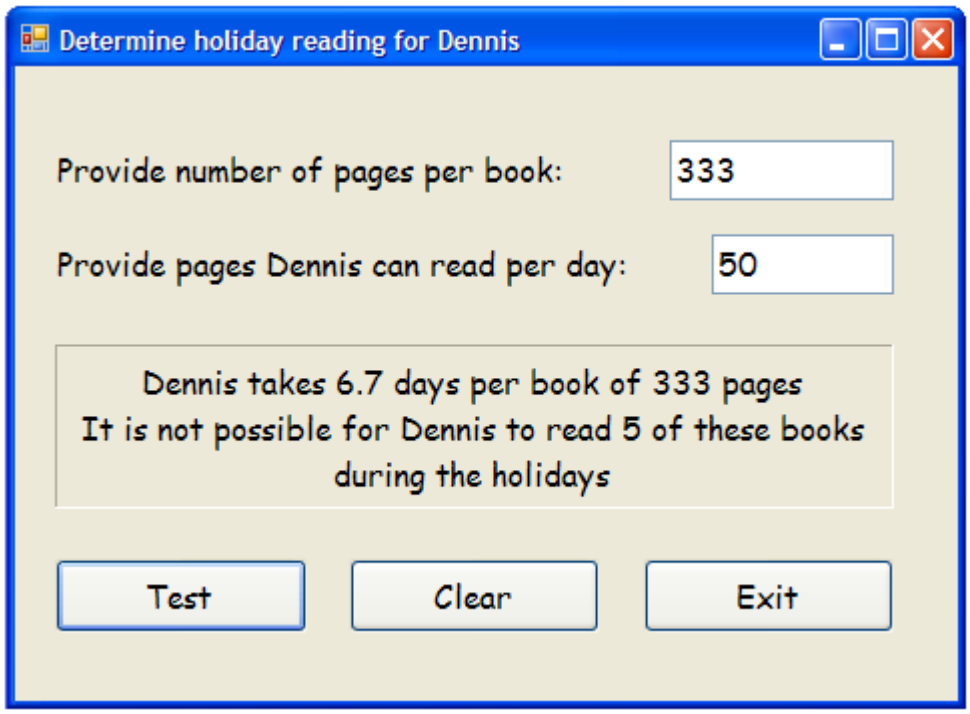


Figure 6-8

6.13 COMPOUND IF-STATEMENTS

The programmer often finds the situation where more than one condition must be tested before the result is true. In order to solve this problem, a compound if-statement that is based on logical operators, is used.

Example: If a person is 18 or more years of age and has a driver's license, this person may drive a car.

```
if age >= 18  AND licensed then          ~  licenced is a boolean variable
        display "This person may drive a car"
else
        display "This person may not drive a car"
endif
```

The result for this compound test is only true when **both** the conditions are true. If only one or none of the conditions is true, the result is false.

Example: If an employee is a grade C employee or has worked for the company for more than 5 years, a special bonus of R1000.00 is awarded to this employee.

```
if grade = "C"  OR  years > 5 then
        bonus = 1000
else
        bonus = 0
endif
```

For the result of this compound test to be true **at least one** of the conditions must be true. If none is true, the result is false.

In the case where the if-statement contains more than 2 conditions, it is advisable to use parenthesis to indicate which tests must be done before the other, since the tests in the parenthesis are done first.

```
if gender = "M"  OR  (age > 20  AND  member) then
        ..statement(s) …
endif
```

or depending on the question / conditions

```
if (gender = "M"  OR  age > 20)  AND  member then
        ..statement(s) …
endif
```

If parenthesis are not used, all the AND tests are evaluated from left to right and only then the OR tests will be evaluated from left to right.

The word NOT may also be used, but must be handled extremely careful, for the beginner-programmer the opposite result might be achieved. NOT has a higher precedence than AND and OR which means it will be evaluated first. You can also refer back to 6.3 page 107 where these logical operators have been summarized in a truth table.

6.14 EXAMPLE OF COMPOUND IF-STATEMENTS

This section will indicate the planning of a problem with a compound if-statement and the VB.NET solution will also be provided.

6.14.1 EXAMPLE

Problem

Everybody in the town Little River will go to the circus during the weekend. Write an algorithm to determine the entrance fee to the circus as persons younger than 12 years or 60 years of age or more, receive a 25% discount.
The user must enter the normal entrance fee and their age in years only. Display the actual entrance fee to be paid on the screen.

	Description	Type	Name of variable
Input:	entrance fee	real	fee
	person's age	integer	age
Output:	actual fee	real	actFee

I	P	O
fee age	Prompt to read fee and age Enter input fields Determine actFee Display actFee	actFee

Algorithm:

DetermineEntranceFee

```
~ Enter input data
display "Enter the normal fee: "        ~ display on new line
enter fee
display "Enter the age: "               ~ display on new line
enter age
```

```
        ~       test  age
        if age < 12  OR  age > 59
                actFee = fee  -  fee * 0.25              ~ subtract discount
        else
                actFee = fee
        endif

        ~       display actual fee on a new line
        display "This person must pay an entrance fee of R", actFee
end
```

Test the program twice:

Fee of R50 and an age of 9 years
Fee of R80.50 and an age of 32 years

Output:

```
Enter the normal fee: 50
Enter the age: 9
This person must pay an entrance fee of R37.50

Enter the normal fee: 80.50
Enter the age: 32
This person must pay an entrance fee of R80.50
```

6.14.2 SOLUTION IN VB.NET FOR THIS PROBLEM

6.14.2.1 USER INTERFACE WITH NAMED CONTROLS

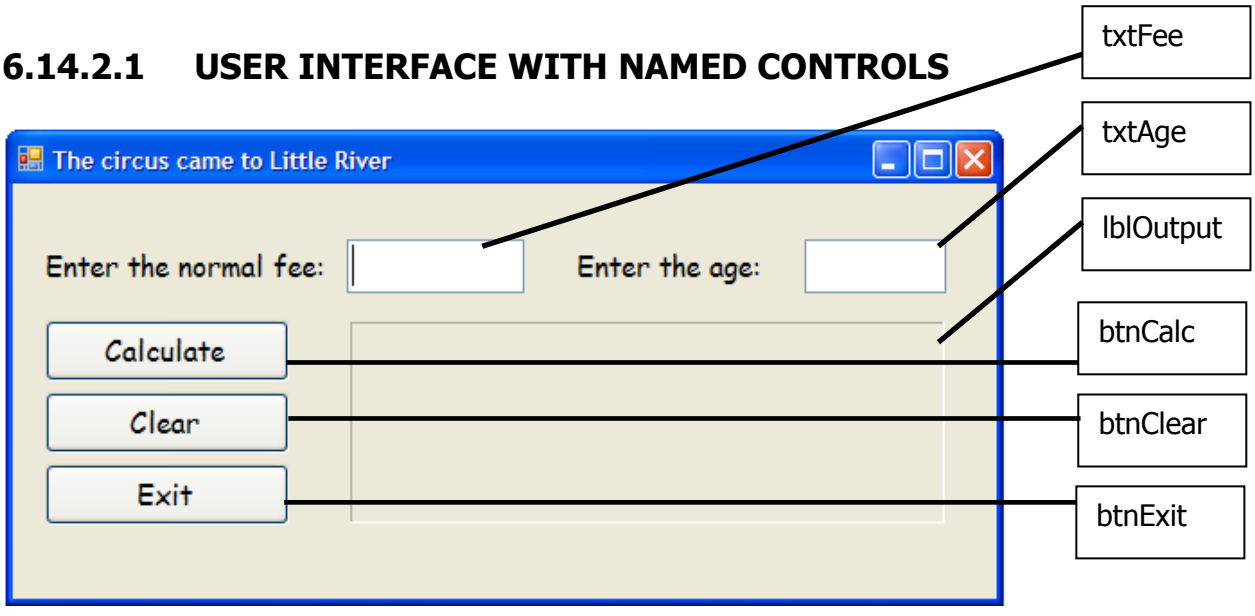


Figure 6-9

6.14.2.2 CODE IN THE CODE EDITOR

```
Option Explicit On
Option Strict On

Public Class frmCircus

Private Sub btnCalculate_Click(. . .) Handles btnCalculate.Click
   Dim decFee, decActFee As Decimal
   Dim intAge As Integer
   decFee = Convert.ToDecimal(txtFee.Text)
   intAge = Convert.ToInt32(txtAge.Text)
   If intAge < 12 Or intAge > 59 Then
      decActFee = decFee - decFee * 0.25D
   Else
      decActFee = decFee
   End If
   lblOutput.Text = "This person must pay an entrance fee of R" _
                    & decActFee.ToString("N2")
End Sub

Private Sub btnClear_Click(. . .) Handles btnClear.Click
   txtFee.Text = ""
   txtAge.Text = ""
   lblOutput.Text = ""
   txtFee.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
   Close()
End Sub

End Class
```

6.14.2.3 EXAMPLES OF OUTPUT

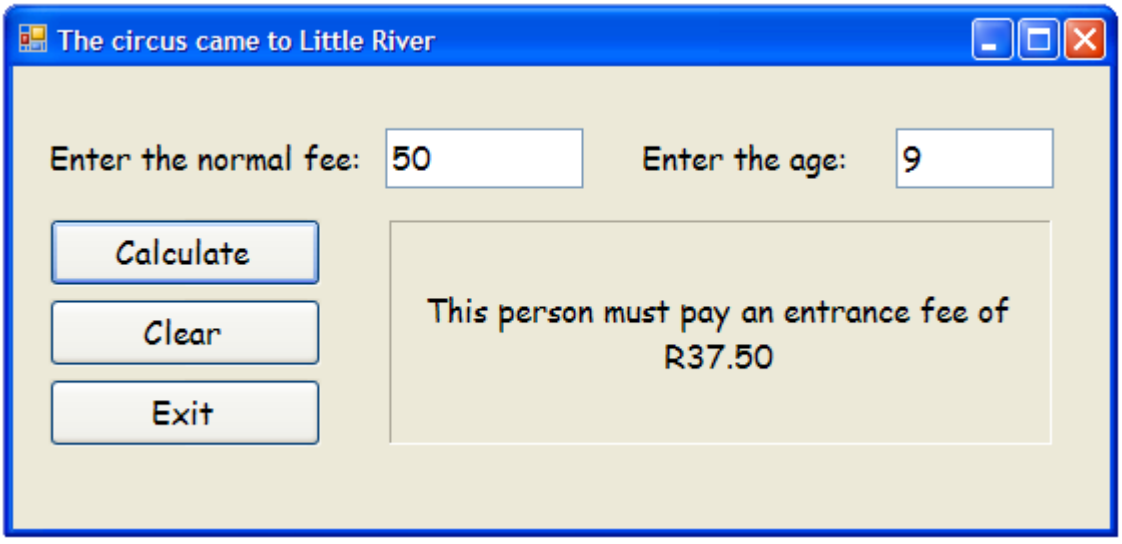


Figure 6-10

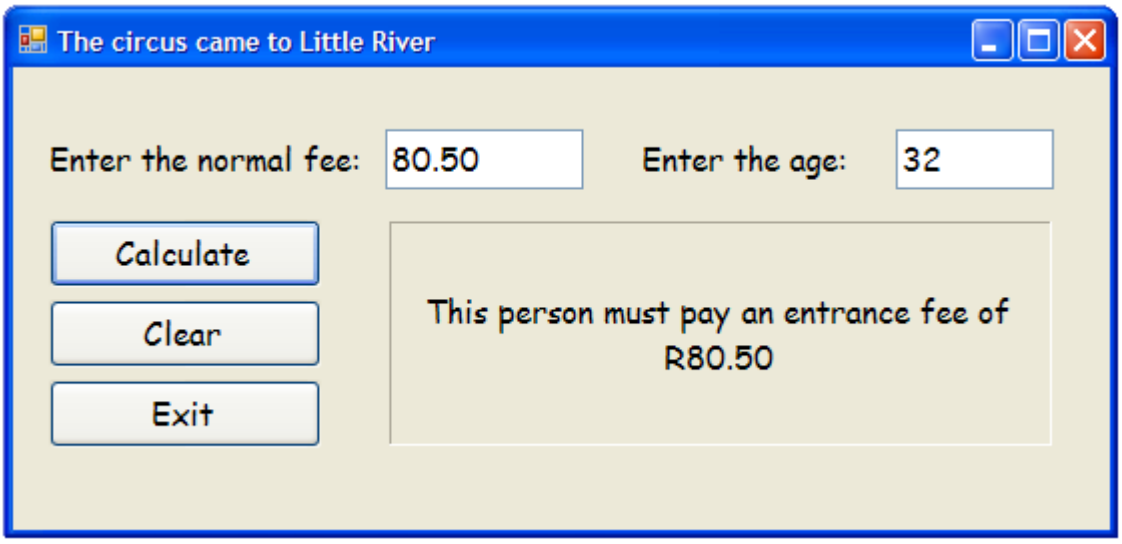


Figure 6-11

6.15 THE LOGICAL OPERATORS AndAlso & OrElse

Apart from the NOT, AND and OR operators, Visual Basic.NET also provides for the AndAlso and OrElse operators. Both these operators provide for short-circuit evaluation.

We know that for an AND to evaluate to TRUE, both conditions must be TRUE. Therefore, in an AndAlso, if the first condition evaluates to FALSE, the second condition will not even be evaluated because the answer will already be FALSE.

We also know that an OR operator will evaluate to TRUE if at least one of the conditions evaluates to TRUE. Therefore, in an OrElse, if the first condition evaluates to TRUE, the second condition will not even be evaluated because the answer will already be TRUE.

The AndAlso will therefore produce exactly the same result as an AND, and the OrElse will produce exactly the same result as an Or. However, both these operations will be more efficient.

The AndAlso and OrElse operators can be summarized in the following truth table.

Condition1	**Condition2**	Condition1 **AndAlso** Condition2
True	True	True
True	False	False
False	Not Evaluated	False
		Condition1 **OrElse** Condition2
True	Not Evaluated	True
False	True	True
False	False	False

6.16 EXAMPLE WITH BOOLEAN VARIABLES

In the next example the use of a boolean variable will be illustrated.

Problem:

A student wants to work out at the Keep Fit Gym. The daily entrance fee is R8 for a member and R12 for a non-member. The user has to enter the student name and an indication whether the student is a member or not (use a Boolean value). Display the student name and the entrance fee of the student.

	Description	Type	Name of variable
Input:	Membership status	boolean	member
	Student name	string	stName
Output:	Student name	string	stName
	Outcome of if	string	message

I	P	O
stName member	Prompt to read input fields Enter input fields Prepare message Display output fields	stName message

Algorithm:

```
DetermineEntranceFee

        ~       Enter input data
        display "Enter student name:"                       ~ display on new line
        enter stName
        display "Indicate if the student is a member or not:"   ~ display on new line
        enter member

        ~       test membership
        if member then                          ~ Equivalent to if member = TRUE
                message = " you must pay R8 entrance fee"
        else
                message = " you must pay R12 entrance fee"
        endif

        ~       display results
        display stName, message                             ~ display on new line
end
```

Test the program by setting the member to TRUE after the name Sally Jones has been entered:

Output:

```
Enter student name: Sally Jones
Indicate if the student is a member or not: TRUE
Sally Jones you must pay R8 entrance fee
```

Test the program again by setting the member to FALSE after the name Billy Ceronio has been entered:

Output:

```
Enter student name: Billy Ceronio
Indicate if the student is a member or not: FALSE
Billy Ceronio you must pay R12 entrance fee
```

6.17 CHECK BOXES

6.17.1 PURPOSE OF A CHECK BOX

A check box is a control that can be added to the user interface in order for a user to indicate whether a certain condition is true or not (e.g. if the student is a member of the club or not). If the user selects the check box, he or she will indicate that the condition is true, otherwise it is false.

There may be more than one check box grouped together and a user may select more than one of these check boxes. For instance, when the cost of a parcel must be calculated, it could be necessary to enter the weight of the parcel in a text box. It could also be necessary that the user must also indicate whether the parcel must be insured by selecting a check box and whether the parcel has a high priority or not, by selecting a second check box.

It could be possible that a parcel must be insured and has a high priority in which case both check boxes will have to be selected.

6.17.2 PROPERTIES COMMONLY USED FOR CHECK BOXES

Every check box must have a meaningful **name** to indicate the purpose of the check box and, according to the Hungarian notation it must start with the prefix chk.

When a check box is selected, the **checked property** will be set to true, and if a check box is inactive, the checked property will be false. Therefore, in the clear-button the checked property of all check boxes must be set to false.

The **text property** of the check box must specify the text that will appear inside the check box and the **font** of the check box may also be specified.

6.17.3 GUI DESIGN GUIDELINES FOR CHECK BOXES

Keep the following in mind when placing check boxes onto a form:

- Use a check box when you want the user to select one or more choices from a group of options
- A check box can also be used on its own if you want the user to indicate whether a certain condition is true or false e.g. indicate whether you are a member of the club – yes or no.
- The label in the text property of the check box should be entered in sentence capitalization.

6.18 A SOLUTION IN VB.NET WITH A CHECK BOX TO TEST A CONDITION

For this example, the problem planning in example 6-16 will be used. However, the problem is modified in the following way:

The daily entrance fee at the Keep Fit Gym is R8 for a member and R12 for a non-member. Students get 10% discount. The user has to enter his/her name and an indication whether he or she is a member or not (use a Boolean value) and whether he/she is a student or not (another Boolean value).

Display the name of the person entering as well as the entrance fee to be paid.

The program will not only display the entrance fee in a message, but will calculate the entrance fee and discount in variables. The answer will then be concatenated into the message that must be displayed on the label.

6.18.1 USER INTERFACE AND NAMED CONTROLS

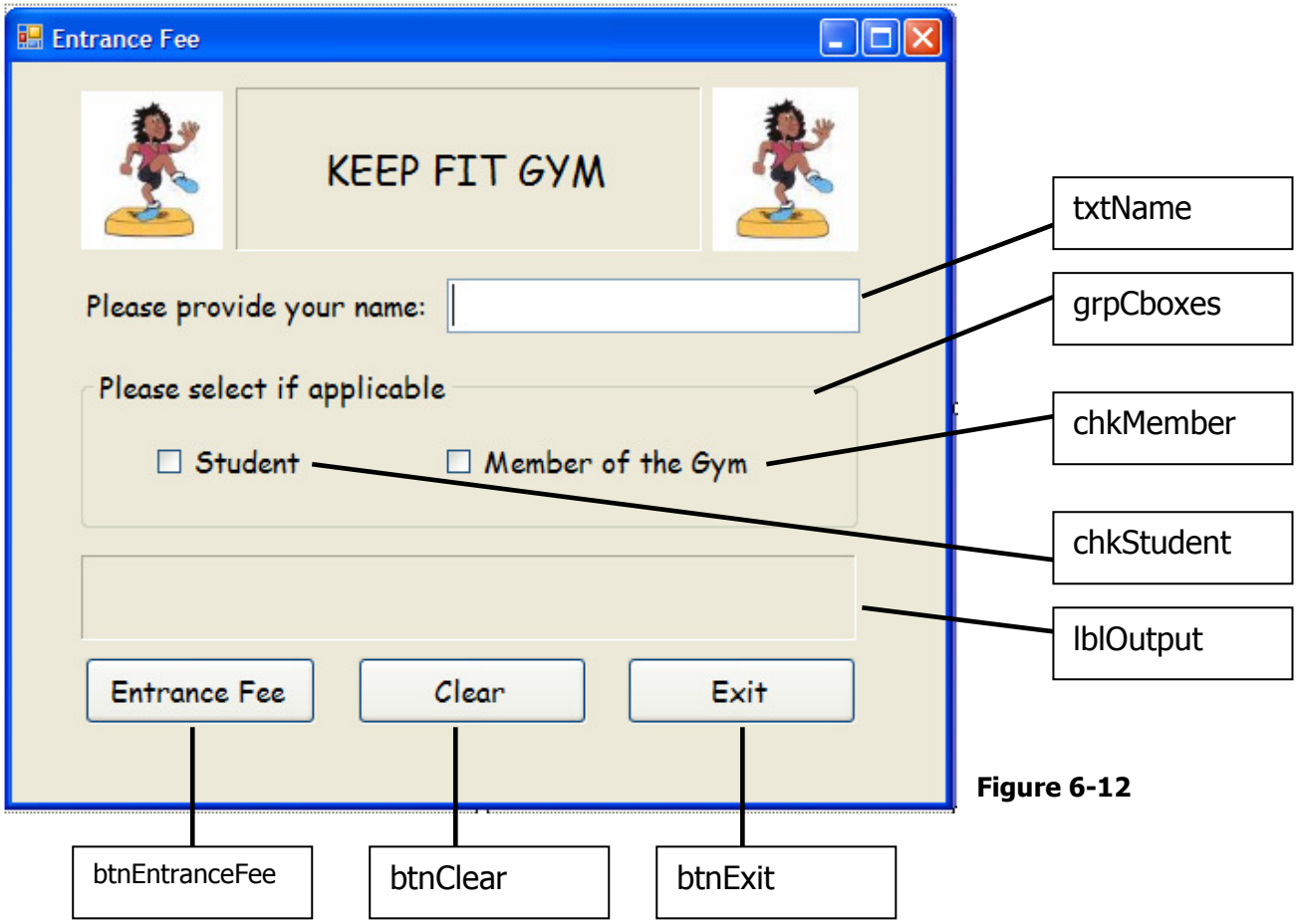


Figure 6-12

6.18.2 CODE IN THE CODE EDITOR

```
Option Explicit On
Option Strict On

Public Class frmGym
Private Sub btnEntranceFee_Click(. . .) Handles btnEntranceFee.Click
   Dim strName, strMessage As String
   Dim decEntranceFee, decDiscount As Decimal
   strName = txtName.Text
   If chkMember.Checked Then
      decEntranceFee = 8D
   Else
      decEntranceFee = 12D
   End If
   If chkStudent.Checked Then
      decDiscount = decEntranceFee * 0.1D
   Else
      decDiscount = 0
   End If
   decEntranceFee = decEntranceFee - decDiscount
   strMessage = " you must pay R" & decEntranceFee.ToString("N2") _
                & " entrance fee"
   lblOutput.Text = strName & strMessage
End Sub
```

Test if member check box has been selected in order to determine entrance fee.

Test if student check box has been selected in order to calculate discount.

```
Private Sub btnClear_Click(. . .) Handles btnClear.Click
   txtName.Text = ""
   chkStudent.Checked = False
   chkMember.Checked = False
   lblOutput.Text = ""
   txtName.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
   Close()
End Sub
End Class
```

A check box can be cleared by setting the checked property to False

6.18.3 EXAMPLES OF OUTPUT

As indicated in **Figures 6-13 to 6-15**, it is possible to select 0, 1 or both check boxes which will each result in a different entrance fee to be paid:

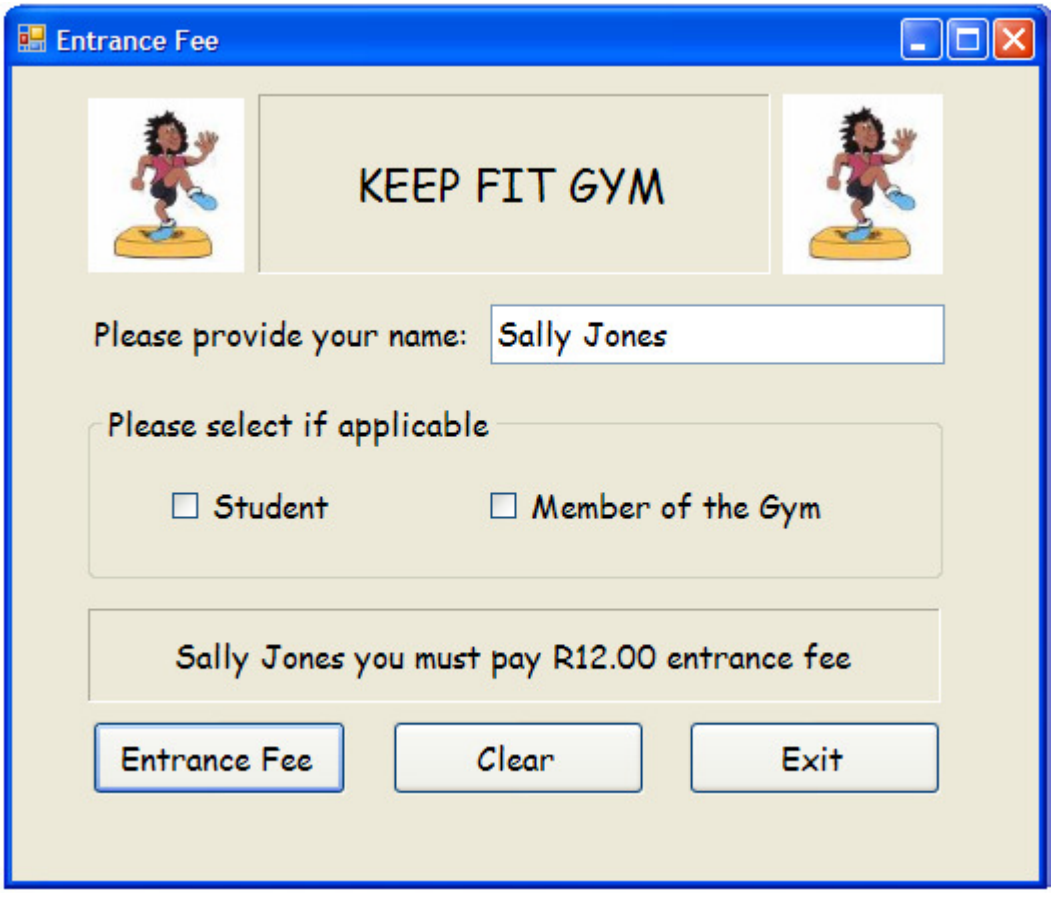


Figure 6-13

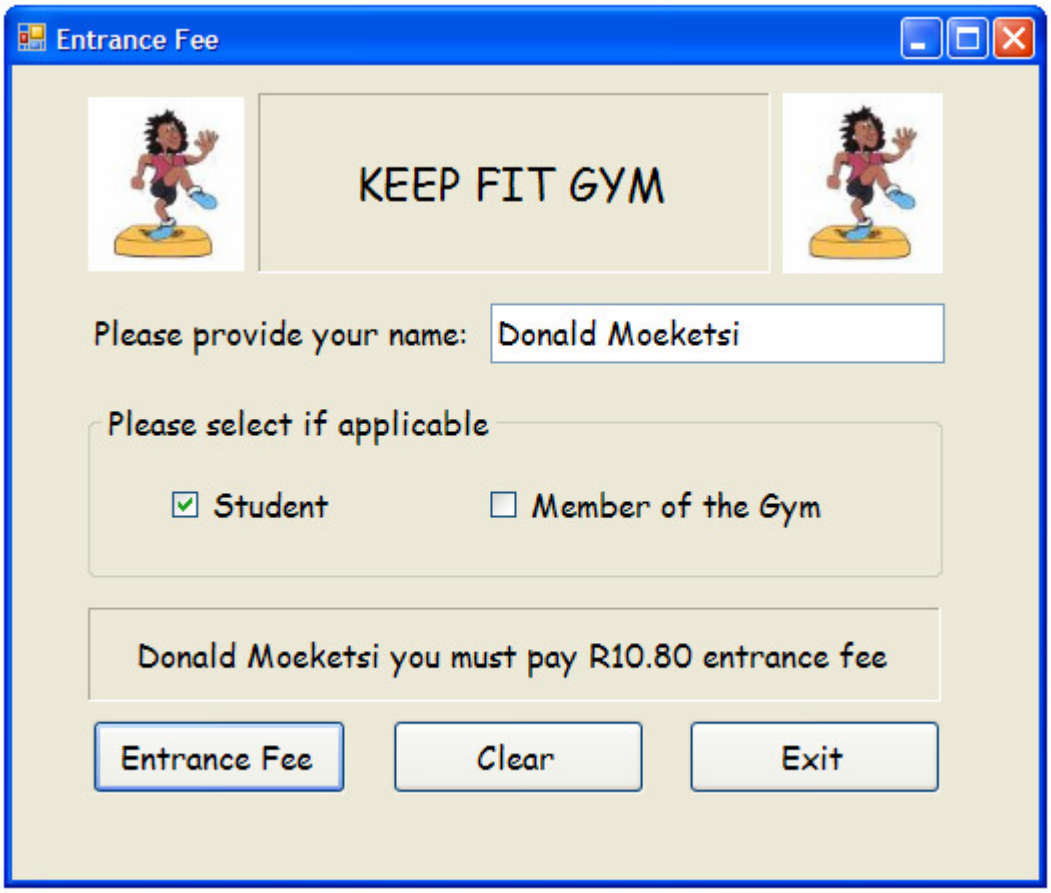


Figure 6-14

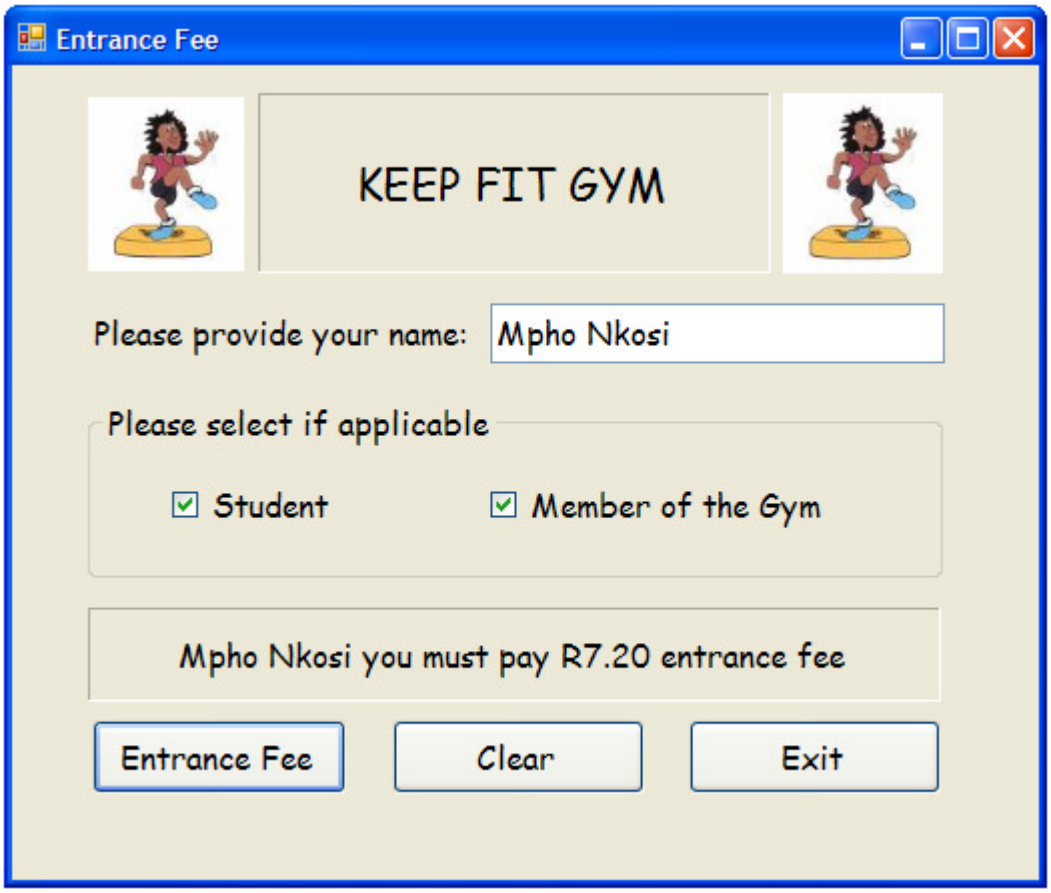


Figure 6-15

6.19 DATA VALIDATION

It is a good programming technique to code defensively and to try and detect and identify all errors that a user could possibly make. Sometimes users are inexperienced and many computer errors that occur are actually a result of incorrect user entries. If a program can detect an error before executing any processing, the information will be accurate and reliable.

There are a number of data validation tests that a programmer can use to code defensively:

6.19.1 NUMERIC TEST

Obviously, a numeric field must contain a numeric value before it can be used for any processing. The program will not be able to do calculations if the field does not contain a numeric value.

Example of numeric test in an algorithm:

```
display "Please provide the hours worked and the tariff per hour"
enter hours
enter tariff
if hours is numeric and tariff is numeric
        wage = tariff * hours
else
        display "Invalid input values"
endif
```

6.19.2 TEST FOR CORRECT VALUES IN INPUT FIELDS

Assume that the distance that contestants must run in a certain race depends on their gender. Female contestants run a distance of 75 km and male contestants run 100 km. The gender must be entered. It is good practice not to assume that the user entered an 'F' if the gender is not an 'M'. The user could have entered an incorrect code and must get the opportunity to rectify the mistake until the correct gender is entered.

Example of poor code that could lead to possible errors in the output:

```
if gender = "M"
        distance = 100
else
        distance = 75
endif
```

Improved code:

```
if gender = "M"
        distance = 100
else
        if gender = "F"
                distance = 75
        else
                distance = 0
                display "You have entered an invalid gender"
        endif
endif
```

NOTE: This previous if-statement is an example of a nested if-statement that will be discussed in chapter 7.

6.19.3 RANGE TEST

Sometimes the user must enter a value that falls in a certain range. Suppose a user must enter 1 to add a new record to a file, 2 to modify an existing record, 3 to delete a record and 4 to end the process. Before the actual update process is executed, the following test can occur to determine whether a valid choice has been entered or not.

Example:

```
if choice < 1 or choice > 4
        display "You have entered an invalid choice"
else
        ~ do actual updating
endif
```

6.19.4 OTHER TESTS

It is also a good practice to do **range tests** to check whether a value is within normal bounds. For instance, a company might have a policy not to pay a monthly salary of more than R100 000.00. A **consistency test** can ensure that fields match. For example, it can check whether an employee's salary agrees with his or her job description. **Completeness tests** are necessary to verify that all input fields contain a value. This is especially important before calculations can be done or before a new record is added to a file.

Although this course will not deal with dates, file processing or arrays it is worth mentioning, that when working with **dates**, ensure that date fields contain valid values. For instance, the range of the month must be from 1 to 12, the day must be in the range 1 – 28, 29, 30 or 31 depending on the number of days in the specific month. The necessary test for leap years must also be done.

With **file processing** it could be necessary to check that records being read from the file are in the correct sequence, and to test records for their existence. E.g. before a record is added to a file, a test must be done to ensure that it does not already exist. Similar, before a record can be updated, ensure that it exists and when dealing with **arrays**, an index test must be done to ensure that the value of the index is in the definition range of the array.

6.20 HOW DATA VALIDATION TECHNIQUES ARE APPLIED IN VB.NET

By now we know that once an input value has been entered via a text box it must be converted to the same data type as that of the variable into which it will be placed.

Examples:

```
decAmount = Convert.ToDecimal(txtAmount.Text)
intNoStudents = Convert.ToInt32(txtNum.Text)
chaDept = Convert.ToChar(txtDept.Text)
strName = Convert.ToString(txtName.Text)
```

The last example can also be rewritten as strName = txtName.Text, because any text property is of the format String.

If we think in terms of data validation, it would make sense to ensure that the value entered, is in the same format as the variable into which you want to place it. A variable of the data type integer cannot contain a real value nor can it contain a non-numeric value and if such an incorrect value has been entered an appropriate error message must be displayed.

The error message can be displayed on a label or in a message box which is a more acceptable way to do it in any windows based application.

If no data validation is performed the program will end in an abrupt way when the invalid data is entered and the program tries to convert it to an incompatible data type.

Let's consider an example to enter the number of students in a class. The number of students must be an integer numeric value. Let's keep the example simple and the only thing that the program must do is to display the number of students entered on a label.

If no data validation is done, the program will display the following VB.NET error when a wrong value is entered (refer to **Figure 6-16**)

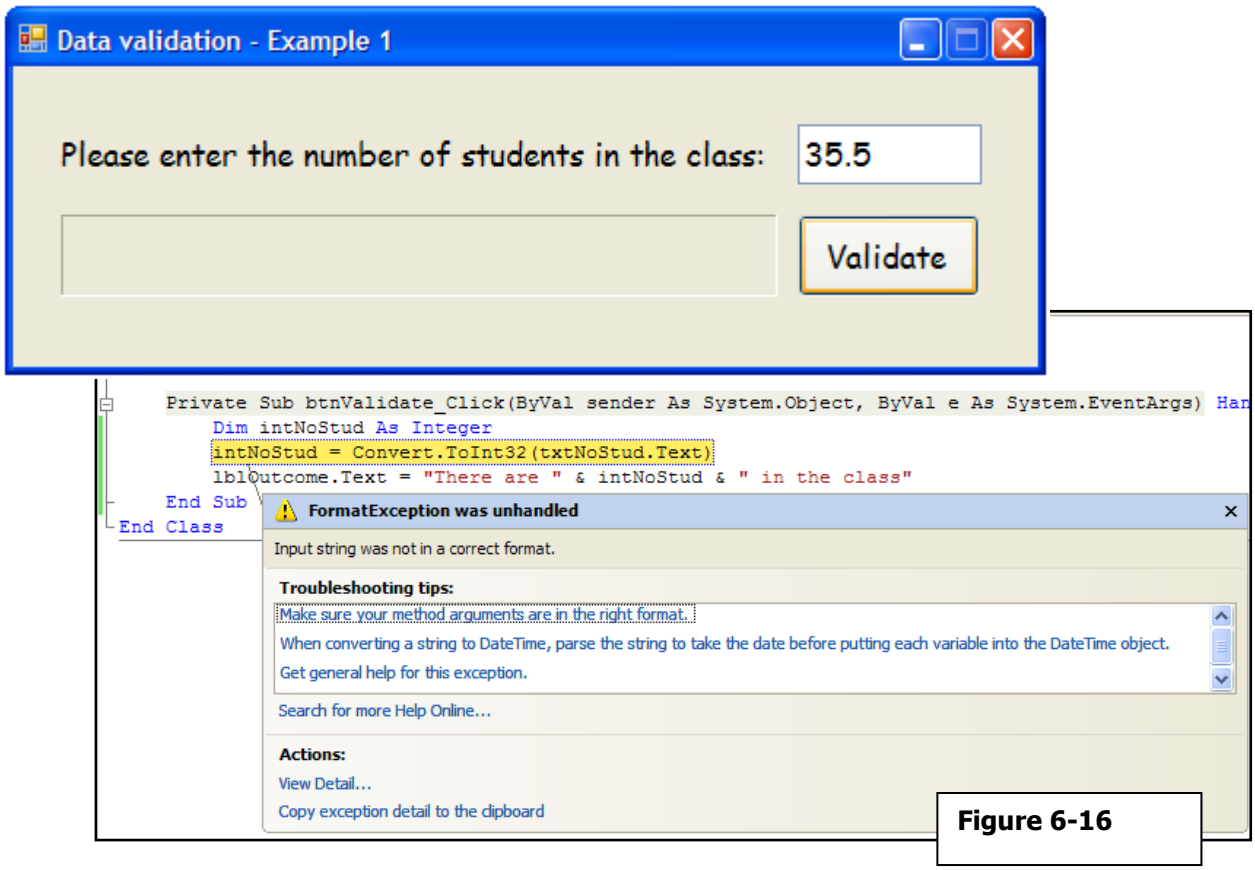


Figure 6-16

6.20.1 TryParse METHOD

The TryParse method is used to convert a string to a specific numeric data type (e.g. integer, decimal etc.) and to test whether the conversion was successful or not.

The word Parse means to analyze something by separating it into individual elements. The TryParse method analyzes a string of input data by separating it into individual characters, and determining whether each character can be converted to the specified data type.

In essence the TryParse method does exactly the same as the Convert method, yet it also tests whether the result is successful and returns a Boolean value (True or False) depending on the outcome of the conversion.

Format of the TryParse method:

*Datatype.***TryParse***(string, [numberStyles, IFormatProvider,] variable)*

The TryParse method has 4 arguments, but 2 of these arguments (numberStyles and IFormatProvide) are optional and will not be covered in this course.

Compulsory arguments:

String: This is a string value that must be converted to a new data type. This is typically the Text property of a control where input data has been entered.

Variable: This is the name of the variable where the TryParse method must store the result of the conversion. It must be of the same type as the *datatype* portion of the syntax.

E.g.

Decimal.TryParse(txtSales.Text, decSales)

The string that has been entered in the text property of the text box txtSales, must be converted to a numeric decimal value and placed in the variable decSales.

The optional arguments will enable the user to enter certain formatting characters such as decimal points and currency characters and it will still be able to successfully convert it to specified numeric data types.

More Examples:

decAmount = Convert.ToDecimal(txtAmount.Text) is equivalent to
Decimal.TryParse(txtAmount.Text, decAmount)

intNoStudents = Convert.ToInt32(txtNum.Text) is equivalent to
Integer.TryParse(txtNum.Text, intNoStudents)

chaDept = Convert.ToChar(txtDept.Text) is equivalent to
Char.TryParse(txtDept.Text, charDept)

In our example in **Figure 6-16**, to enter the number of students in a class, we can replace the Convert method by the TryParse method and make use of an if-statement to display the number of students in the class on a label only if a valid value was entered. If not, a suitable error message can be displayed instead.

New code in the code editor for the click event of the btnValidate button:

```
Dim intNoStud As Integer
If Integer.TryParse(txtNoStud.Text, intNoStud) Then
   lblOutcome.Text = "There are " & intNoStud & _
                     " students in the class"
Else
   lblOutcome.Text = "The value entered may only be an integer"
End If
```

If an invalid value such as 35.5 or xx is entered, the error message will be displayed as indicated in **Figure 6-17**. For a valid number such as 77, the message indicated in **Figure 6-18** will be displayed on the label.

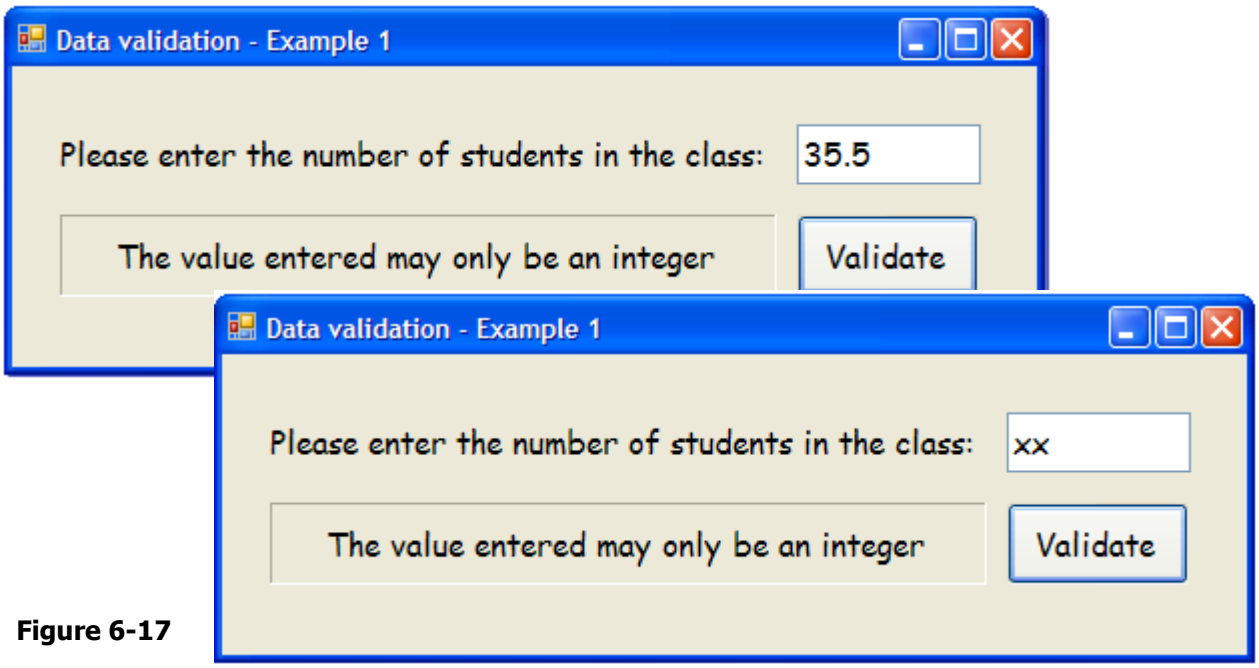


Figure 6-17

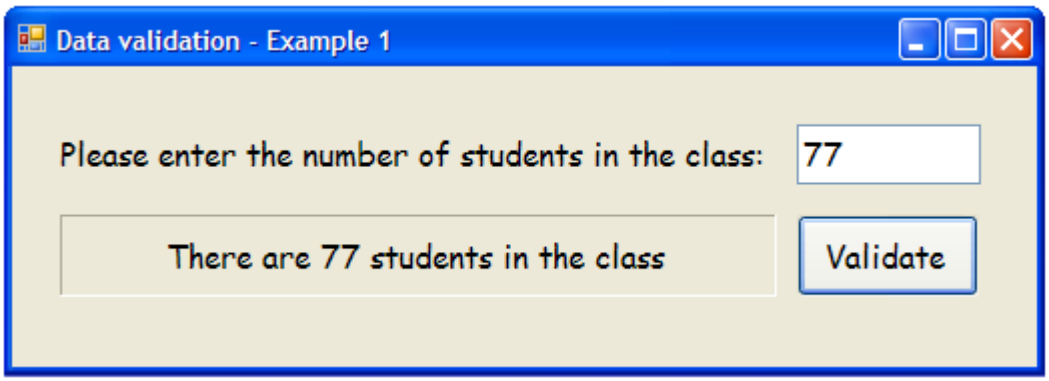


Figure 6-18

The TryParse method only tests if the value entered is in the correct format and other input errors may still occur unless we specifically test for it.

We know that the number of students in a class must be a **positive** integer and a value of -7 should also result in an error. However, with our previous code for the click event of the btnValidate button, the following output will be displayed (as indicated in **Figure 6-19**).

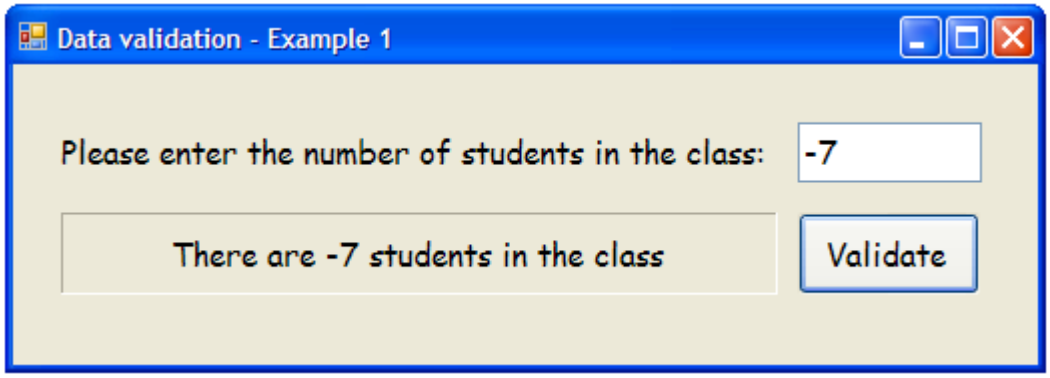


Figure 6-19

After we are satisfied that the value entered is an integer, we will have to include a test to ensure that a valid integer is entered. We can either do it in an additional if-statement (A nested if-statement which will be discussed in the next chapter) or we can modify the code as follows:

```
Dim intNoStud As Integer
If Integer.TryParse(txtNoStud.Text, intNoStud) > 0 Then
   lblOutcome.Text = "There are " & intNoStud & _
                     " students in the class"
Else
   lblOutcome.Text = "The value entered must be a positive integer"
End If
```

Our if-statement now tests whether the value entered in the text box can successfully be converted to an integer and if it is greater than 0. This code will produce the output as indicated in **Figure 6-20**.

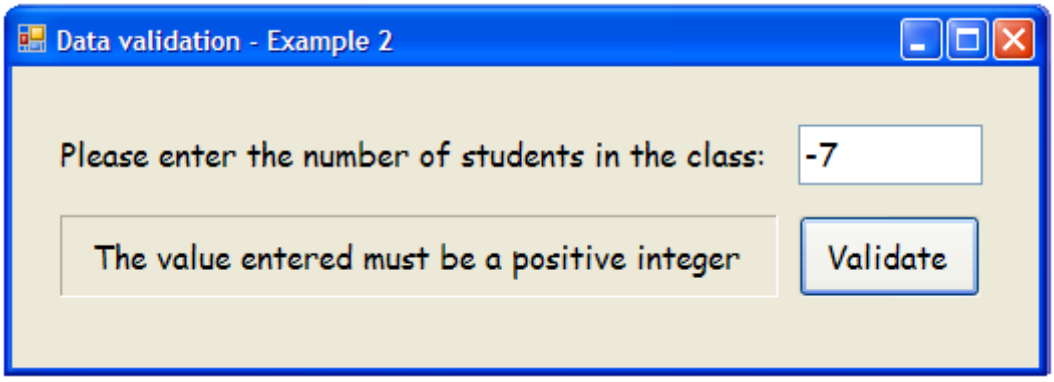


Figure 6-20

The outcome of the TryParse method can also be assigned to a boolean variable and this variable can then be used in the if-statement to test whether the conversion was valid.

Example:

```
Dim blnValid As Boolean
Dim intNoStud As Integer
blnValid = Integer.TryParse(txtNoStud.Text, intNoStud)
If blnValid = True then
   lblOutcome.Text = "There are " & intNoStud & _
                     " students in the class"
Else
   lblOutcome.Text = "The value must be a positive integer"
End If
```

6.20.2 MESSAGE BOX.SHOW METHOD

As mentioned earlier, the more appropriate way of displaying error messages is by means of a message box as that is peculiar to the windows environment.

A message box contains text, one or more buttons and/or an icon. The programmer can design the message box in such a way to support the conveying message. For example, if a message must be displayed to inform the user about a certain process, the icon could indicate that it is information and there only need to be an OK button for the user to click after he or she has read the information.

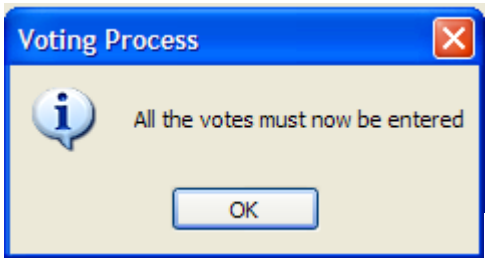


Figure 6-21

On the other hand, if a certain input error has been detected, the icon could be an exclamation mark to indicate the seriousness of the message and the message box could contain a retry button and a cancel button to allow the user to choose whether he or she would like to rectify the mistake or to cancel the process. If the user must rectify the mistake, an ok button could also be used.

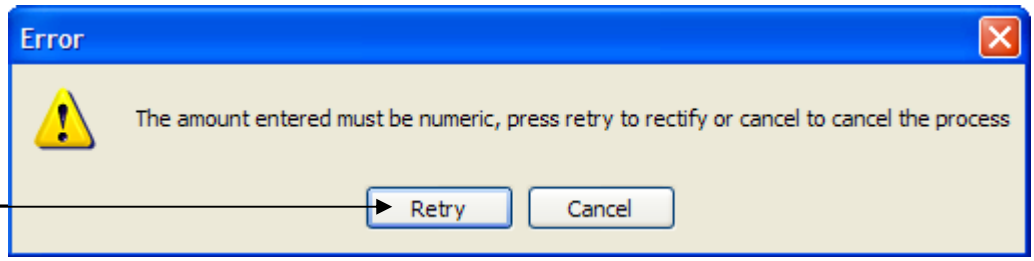


Figure 6-22

Syntax of a message box:
MessageBox.Show(text, caption, buttons, icon, [default button])

caption: Text displayed in title bar of message box
text: The message displayed in the message box
icon: The icon to be displayed in the message box
buttons: One or more buttons to be displayed
default button: The button that will automatically be selected if the user presses enter

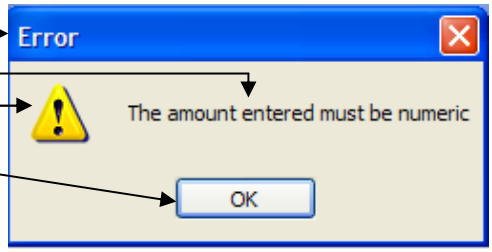


Figure 6-23

The following icons are available:

- MessageBoxIcon.Exclamation
- MessageBoxIcon.Information
- MessageBoxIcon.Stop
- MessageBoxIcon.Question
- MessageBoxIcon.None



The following buttons are available:

- MessageboxButtons.AbortRetryIgnore
- MessageboxButtons.OK
- MessageboxButtons.OKCancel
- MessageboxButtons.RetryCancel
- MessageboxButtons.YesNo
- MessageboxButtons.YesNoCancel

There may be a maximum of three buttons on a message box and any of these three may be set as the default button. This entry is optional and if it is omitted, the first button will be treated as the default.

- MessageBoxDefaultbutton.Button1 (default setting)
- MessageBoxDefaultButton.Button2
- MessageBoxDefaultButton.Button3

It is once again not necessary to enter the complete method. As soon as you type the "." after MessageBox, the word Show will appear and you can only press the tab key to select it. After the bracket, text and caption have been provided, a list of all possible Icons will appear. Use the up and down arrow and press the tab key to select the desired option. The same applies for the Buttons and Default button **(See Figure 6-24)**.

E.g.

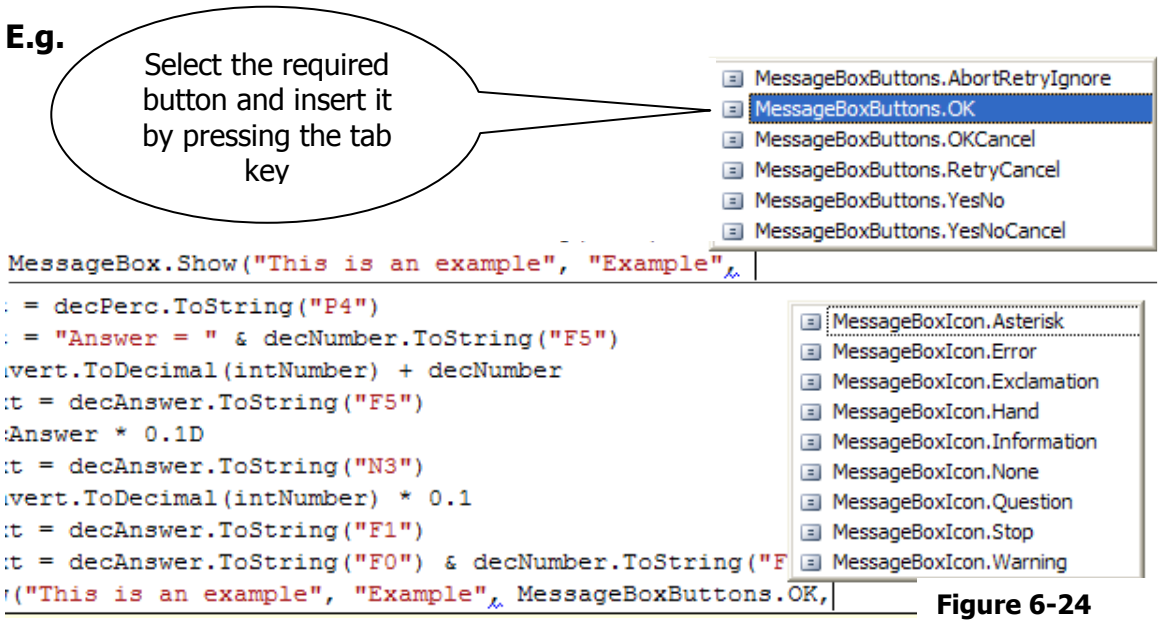


Figure 6-24

Our example can be modified as follows in order to display the error message not on a label but in a message box.

```
Dim intNoStud As Integer
If Integer.TryParse(txtNoStud.Text, intNoStud) > 0 Then
   lblOutcome.Text = "There are " & intNoStud & _
                     "students in the class"
Else
   MessageBox.Show("The value entered must be a positive integer", _
          "Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
End If
```

It will result in the output indicated in **Figure 6-25** if incorrect input data is supplied.

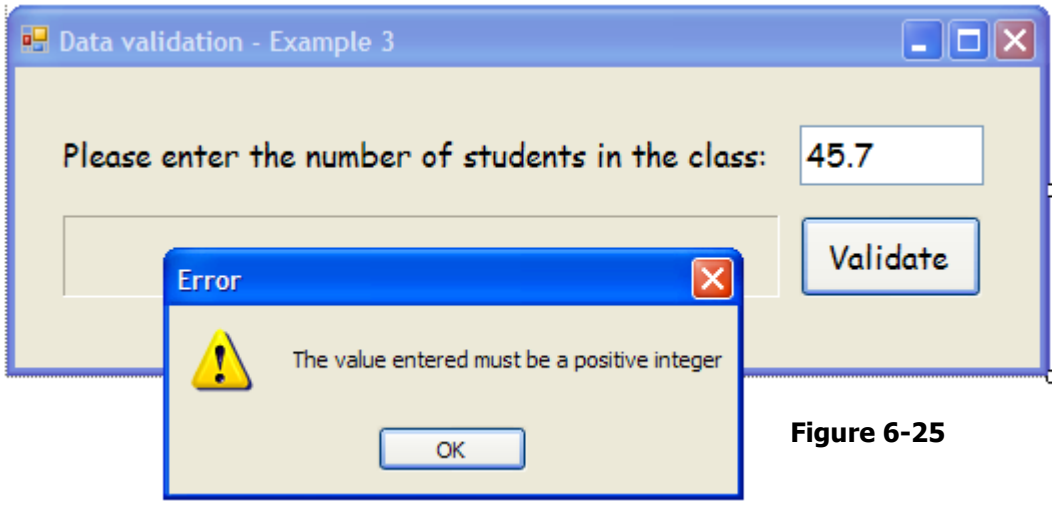


Figure 6-25

6.20.3 String.IsNullOrEmpty METHOD

Sometimes is it necessary to test whether the user has entered an input value before processing can be done. The String.IsNullOrEmpty method will test whether the text property of a control contains a value or not. It can also be used to test whether a variable of data type string, contains a value.

The result of this method is a boolean value – either true or false, depending on whether the string contains a value.

Example 1:

```
If String.IsNullOrEmpty(txtNoStud.Text) Then
   MessageBox.Show("You must enter a value first", _
          "Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
End If
```

Example 2:

In chapter 2 we created an application to ask the first name and surname of a user and then display a welcome message to that specific person. The code did not contain any validation and if the user clicked on the welcome-button before entering a name and surname, the output would have been as indicated in **Figure 6-26**.

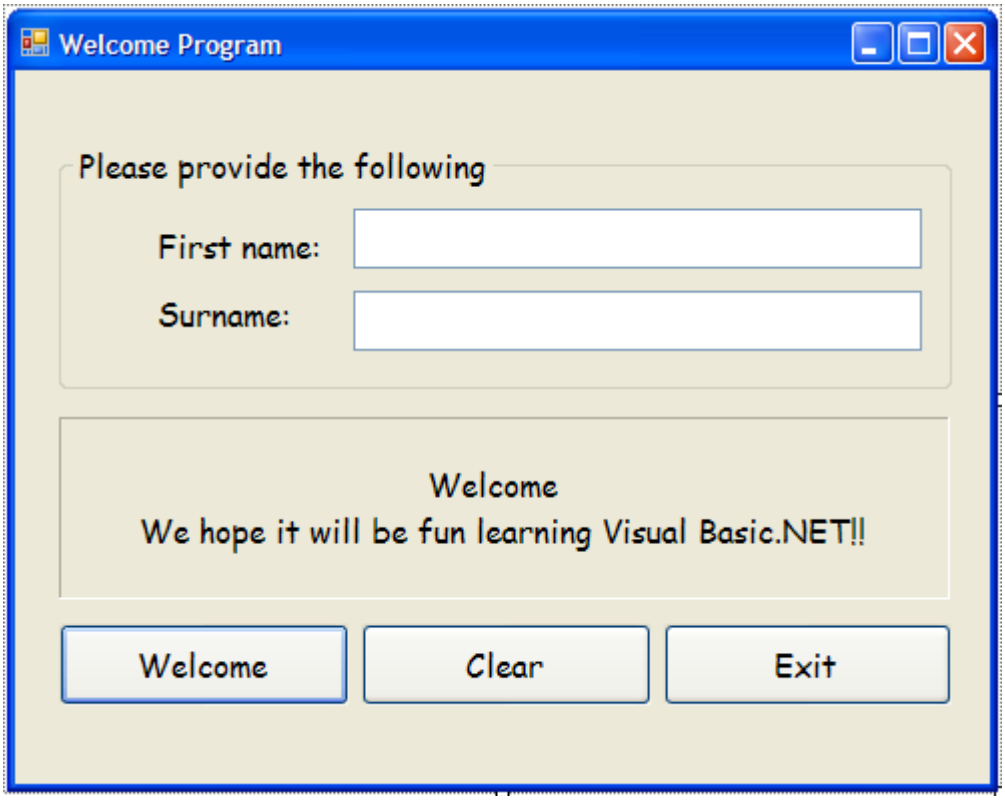


Figure 6-26

Although the program produced output, it is a logic error, because it is not the required output. We can modify the code for the click event of the welcome-button as follows in order to code defensivelty.

```
If String.IsNullOrEmpty(txtFirstName.Text) AndAlso _
   String.IsNullOrEmpty(txtSurname.Text) Then
   MessageBox.Show("You must enter your name first", "Error", _
              MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
Else
   lblWelcomeMessage.Text = "Welcome " & txtFirstName.Text & " " & _
                txtSurname.Text & ControlChars.NewLine & _
                "We hope it will be fun learning Visual Basic.NET!!"
End If
```

This will result in the output as indicated in **Figure 6-27** when the user clicks on the welcome button before a name has been provided.

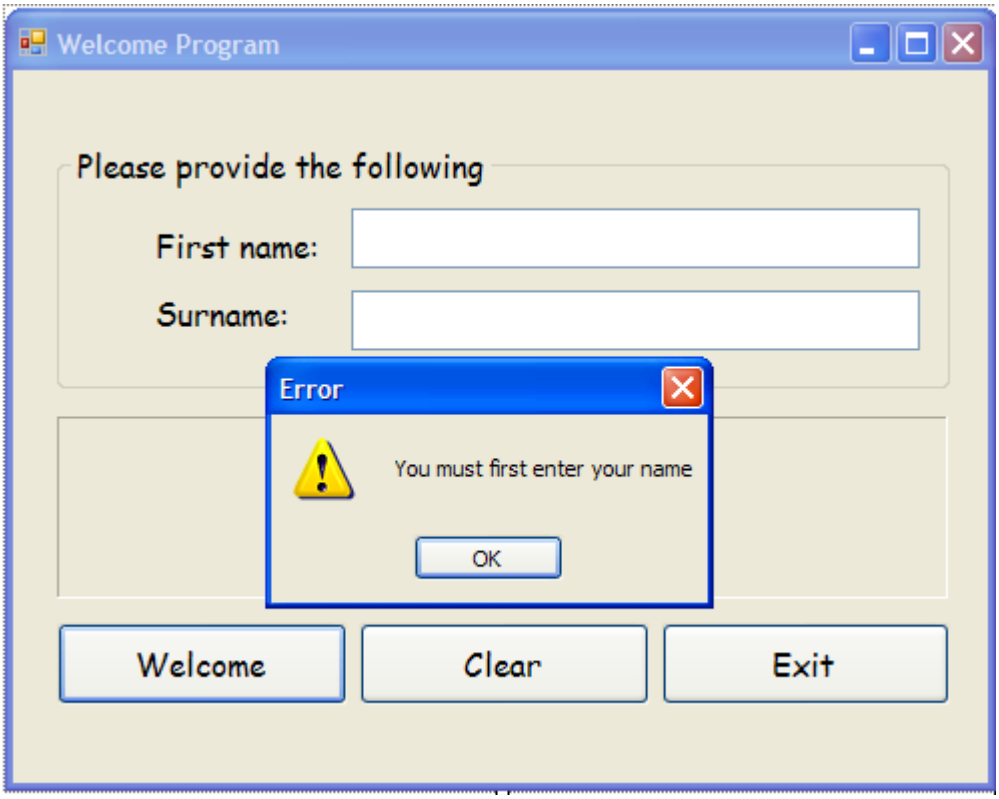


Figure 6-27

6.20.4 ACTIONS TO BE TAKEN AFTER A MESSAGE BOX HAS BEEN DISPLAYED

We will normally just use a message box to display an error message to the user. The user will have to acknowledge this message by clicking on the OK button and the application will simply continue with normal processing. It is good programming to restore the focus to the relevant input control after an error message has been displayed. In this way the user can then rectify his or her mistake by entering a correct value.

A message box need not only be used for error messages. It can also be used to display any other type of message or even an answer of a calculation.

A message box can also contain more than one button (e.g. a YES and a NO button). In such a case it is possible to take certain actions based on the user answer. The constant DialogResult followed by a period and the specific button that the user clicked, enables the programmer to instruct the computer which statements to execute based on the action of the user.

In order to be able to take a certain action, the message box can either be displayed in an if-statement or it can be assigned to a variable that has been declared of the data type integer. This variable or the complete message box can then be tested against the specific dialog result.

The following user choices are possible:

Constant	Meaning
DialogResult.OK	User clicked on the OK button
DialogResult.Cancel	User clicked on the Cancel button
DialogResult.Abort	User clicked on the Abort button
DialogResult.Retry	User clicked on the Retry button
DialogResult.Ignore	User clicked on the Ignore button
DialogResult.Yes	User clicked on the Yes button
DialogResult.No	User clicked on the No button

Example:

Up to now, the code for the Exit-button has only contained the statement Close(). However, the user could have clicked on this button by mistake and it would be a good idea to display a message first and ask the user to confirm (Yes or No) whether that is what he or she intended to do. If the user then clicks on Yes, the application can be terminated, otherwise, nothing must be done and the user will still be active in the application.

The Exit button will now not only contain the function Close(), but would have to change as follows to accommodate a message box.

```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Hand
    Dim intAnswer As Integer
    intAnswer = MessageBox.Show("Are you sure you want to terminate the application?", _
                        "Exit-Button", MessageBoxButtons.YesNo, MessageBoxIcon.Question)
    If intAnswer = DialogResult.Yes Then
        Close()
    End If
End Sub
```

Simple If-statement to test the action that the user has taken

Figure 6-28

The following message box will be displayed after the user clicked on Exit:

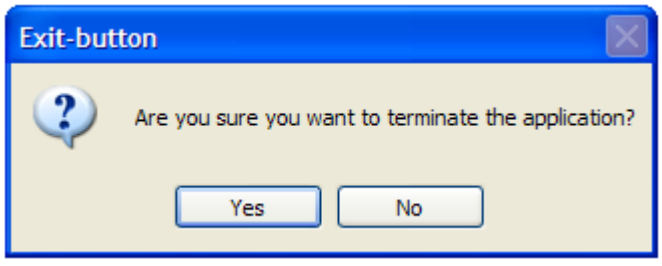


Figure 6-29

If the user now clicks on Yes, the application will be terminated, otherwise the user interface will remain unchanged and it may be cleared in order to accept new input and continue with processing for this new data.



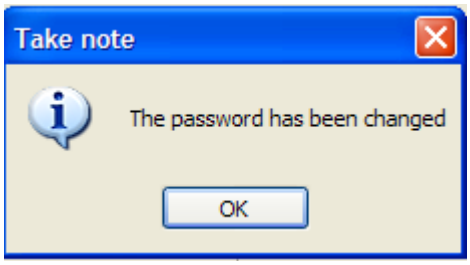
Exercises

Write the necessary statements to display the following message boxes.

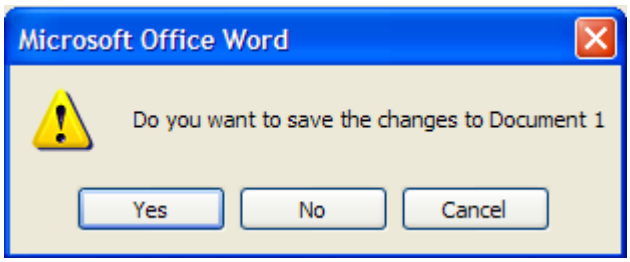
1.



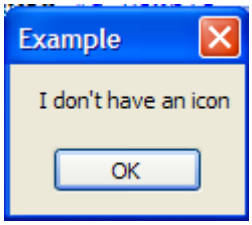
2.



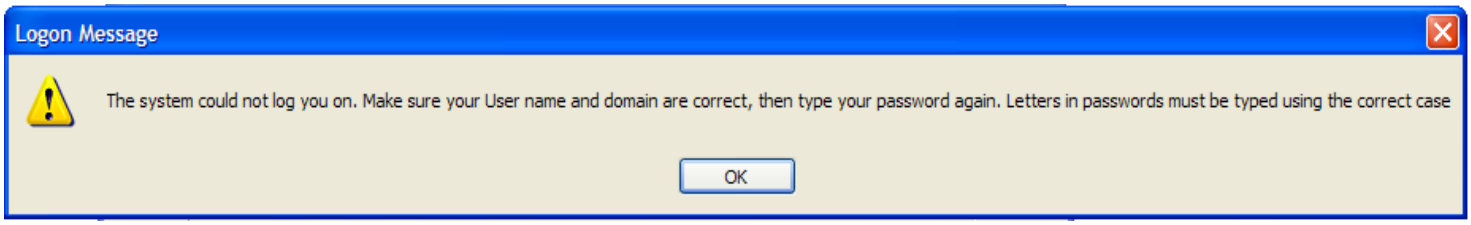
3.



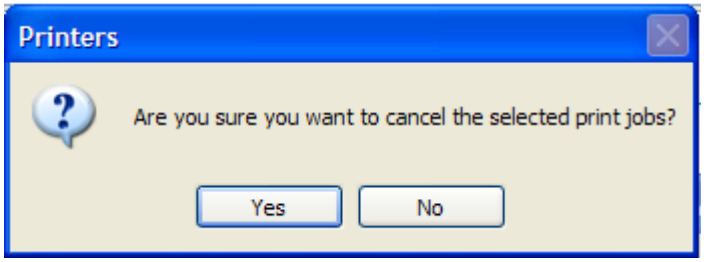
4.



5.



6.





Practical

Accept an employee's annual salary and then calculate and display his/her net monthly salary. Full time employees pay 29.5% income tax. All other employees pay 25% income tax. Display the gross monthly salary, percentage tax, monthly tax payable and net monthly salary as output.

- Identify all input and output variables and draw an IPO-chart for this problem
- Write the algorithm to solve this problem.
- Design two sets of test data and test if your logic works correctly.
- Create the user interface as indicated in Figure 6-30.
- Write the solution in VB.NET and provide for the following:
  Data validation. Use the TryParse Method and a Boolean variable to test if the annual salary can be converted to a decimal value. If not, display the message box indicated in Figure 6-31. Format all output fields in a suitable way.

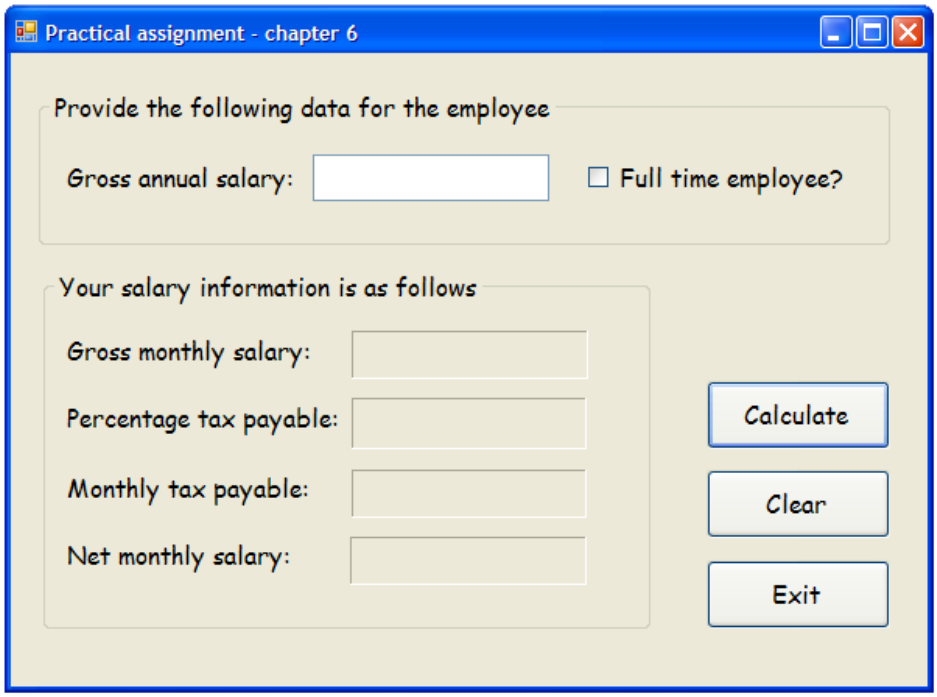


Figure 6-30

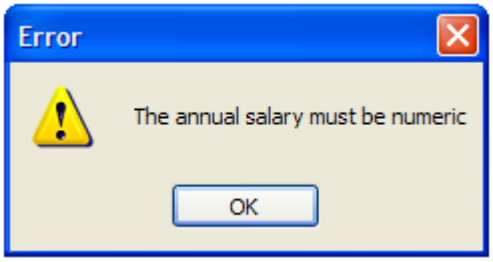


Figure 6-31

CHAPTER 7
THE SELECTION CONTROL STRUCTURE – PART 2

7.1 Introduction

Up to this point we tested only one condition in an if-statement or we have tested more than one condition in a compound if statement, but it is often necessary to test more conditions. This chapter will deal with more complicated decision structures.

OUTCOMES

When you have studied this chapter you should be able to

- write a nested if-statement in pseudocode and in VB.NET,
- write an If-ElseIf-statement in VB.NET,
- use the ToUPPER or ToLOWER methods to change the case of a string,
- know how to add a radio button control to a form and how to use it in decision statements in the code editor,
- write algorithms and corresponding VB.NET solutions containing
  - nested if statements,
  - select case-statements.



Questions

Before we proceed to more complex if-statements, let's revise by answering the following questions about previous chapters. In each case indicate whether the statement is True or False and provide a reason for your answer.

- A constant may not be used in an arithmetic calculation.
- All the steps in any algorithm must always be processed.
- The following expressions are equivalent: A <= B and A > B.
- The following expressions are equivalent: X <= 20 and X NOT > 21
- There may be any number of statements between the if clause and the else clause of an if-statement.

- The statement following the else clause is processed when the condition in the if-statement is true
- In an if-statement without parentheses (brackets), all AND conditions will be evaluated before all OR conditions.
- The following if-statement will be TRUE is x is a multiple of 7
  if x / (x \ 7) = 0
- The value of the following condition is TRUE if a = 5 and b = 17
  if a > b – 13 AND b <= a ^ 2
- Evaluating an algorithm for accuracy can also be called "desk checking"
- IF 7 < age < 12 is a valid test in pseudocode
- NOT (NOT 5 = 6 OR 6 = 5) = (5 = 5)

7.2 NESTED IF-STATEMENTS

Study the following if-then-else-statement:

```
if condition1 then
        statement1          ~ process when condition is true
else
        statement2          ~ process when condition is false
endif
```

Statement1 and / or statement2 may also be if-then-else-statements depending on the problem statement that will change the general format as follows to a nested if-statement:

```
if condition1 then
        if condition2 then
                statement1
        else                        }   ~  if condition1 is true
                statement2
        endif
else
        if condition3 then
                statement3
        else                        }   ~  if condition1 is false
                statement4
        endif
endif
```

Note the indentation in columns in the nested if-statement.

A nested selection structure will be used if more than one decision must be made before the appropriate action can be taken. In the above example, condition1 is the outer selection structure and condition2 and condition3 are the inner selection structures. The inner selection structures may also be nested.

7.3 EXAMPLES OF NESTED IF-STATEMENTS

Write nested if-statements without AND and/or OR for each of the following:

Example 1:

The gender of a person is indicated by a M (male) or F (female). In a competition every male older than 15 years receives 17 points, while a male less 16 receives 20 points. A female less than 18 years receives 18 points, while a female older than 17 years receives 21 points.

```
if  gender = "M"  then
        if age > 15  then
                points = 17                 ~ male older than 15
        else
                points = 20                 ~ male younger than 16
        endif
else
        if gender = "F"
                if  age < 18  then
                        points = 18         ~ female younger than 18
                else
                        points = 21         ~ female older than 17
                endif
        else
                display "Invalid gender"
        endif
endif
```

Following the first else, the points = 20 because in the case where the second if-statement tests false, the age of this male is less or equal to 15. There is no need to test again for this condition. The same rule (logic) applies to the second part of the if statement where a female is tested.

Example 2

The following table contains the values of the real variables A and B and the actions to take when different conditions apply.

Value of A	Value of B	Steps(s) to take
5.2	Greater than 20	Increase the value of A by 5
5.2	Less or equal to 20	Decrease the value of B by 3.5
6	Any value	Initialize the values of A and B

Note: To initialize a value means that zero must be assigned to a numeric value and spaces to a non-numeric value.

```
if  A = 5.2  then
        if B > 20  then
                A = A + 5
        else
                B = B – 3.5          ~ the value of A is still 5.2
        endif
else
        if  A = 6  then              ~ the value of A is not 5.2, but 6
                A = 0
                B = 0
        endif
endif
```

7.4 NESTED IF-STATEMENTS IN VB.NET

The format of a nested if-statement in VB.NET is very similar to that of a nested if-statement in an algorithm.

```
If condition1 Then
   If condition2 Then
      Statement(s)
   Else
      Statement(s)
   End If
Else
   If condition3 Then
      Statement(s)
   Else
      Statement(s)
   End If
End If
```

e.g.

```
If chaGender = "M" Then
   If intAge > 15 Then
      intPoints = 17
   Else
      intPoints = 20
   End If
Else
   If chaGender = "F"
      if intAge < 18 Then
         intPoints = 18
      Else
         intPoints = 21
      End If
End If
```

Visual Basic also provides an If-ElseIf-Else-structure which is sometimes referred to as an extended selection structure.

With the normal If-Else nested if-statement, there must be an End If for every if-statement, regardless whether it contains an else-clause. Sometimes there are a lot of alternatives and the structure may become complex.

With the If-ElseIf-Else structure, only one End If is needed at the end of the complete statement.

Compare the following two examples in VB.NET

If-Else nested structure

```
If chaDept = “A” Then
   lblDept.Text = “Department A”
Else
   If chaDept = “B” Then
      lblDept.Text = “Department B”
   Else
      If chaDept = “C” Then
         lblDept.Text = “Department C”
      Else
         If chaDept = “D” Then
            lblDept.Text = “Department D”
         Else
            lblDept.Text = “Wrong department code”
         End If
      End If
   End If
End If
```

Equivalent If-ElseIf-Else structure

```
If chaDept = “A” Then
   lblDept.Text = “Department A”
ElseIf chaDept = “B” Then
   lblDept.Text = “Department B”
ElseIf chaDept = “C” Then
   lblDept.Text = “Department C”
ElseIf chaDept = “D” Then
   lblDept.Text = “Department D”
Else
   lblDept.Text = “Wrong department code”
End If
```

7.5 ToUpper AND ToLower METHODS

In the previous example it is possible that the user can enter the code of the department in upper or lower case depending if the caps lock key on the keyboard is on or off. Although both cases will be correct, the above code will result in “Wrong department code” because to the computer “a” is not equal to “A” and “a” will be treated as incorrect input.

In order to rectify this, the code must be modified as follows which can cause quite a lot of coding:

```
Dim chaDept As Character
If char.TryParse(txtDept.Text, chaDept) Then
   If chaDept = “A” Or chaDept = “a” Then
      lblDept.Text = “Department A”
   Else
      If chaDept = “B” Or chaDept = “b” Then
         lblDept.Text = “Department B”
      Else
         If chaDept = “C” Or chaDept = “c” Then
            lblDept.Text = “Department C”
         Else
            If chaDept = “D” Or chaDept = “d” Then
               lblDept.Text = “Department D”
            Else
               lblDept.Text = “Wrong department code”
            End If
         End If
      End If
   End If
Else
   lblDept.Text = “you may only enter one character”
End If
```

The ToUpper method will convert the input in a certain variable to uppercase before the test is done and the code could be modified as follows:

```
Dim chaDept As Character
If char.TryParse(txtDept.Text, chaDept) Then
   If chaDept.ToUpper() = “A” Then
      lblDept.Text = “Department A”
   Else
      If chaDept.ToUpper() = “B” Then
         lblDept.Text = “Department B”
      Else
         If chaDept.ToUpper() = “C” Then
            lblDept.Text = “Department C”
         Else
            If chaDept.ToUpper() = “D” Then
               lblDept.Text = “Department D”
            Else
               lblDept.Text = “Wrong department code”
            End If
         End If
      End If
   End If
End If
```

In a similar way, the ToLower method will convert the value in a variable to lowercase.

The ToUpper or ToLower methods can also be implemented when the value in the text box is converted to the correct data type, e.g.

```
chaDept = Convert.ToChar(txtDept.Text.ToUpper())
```

7.6 RADIO BUTTONS

7.6.1 PURPOSE OF RADIO BUTTONS

A radio button is a control that can be placed on a form in order for the user to select an option. It can be used as an input control when the user has to enter a specific value, e.g. A – D to indicate a department. As seen in the previous examples, if this value is entered in a text box, this value will have to be converted to the correct format and it will have to be validated. This is not necessary with a radio button.

A radio button is similar to a check box in the sense that the user will select it and that the checked property will be true if the radio button has been selected. However, when radio buttons are grouped together, the user may select **only one** of the radio buttons by clicking on it. As soon as the user selects another radio button, the first one that has been selected will become inactive.

7.6.2 PROPERTIES COMMONLY USED FOR RADIO BUTTONS

Similar to check boxes, the properties most commonly set for radio buttons are the following:

Name:	Give the radio button a meaningful name starting with rad
Text:	Specify the text that appears inside the radio button
Font:	Specify the font to use for text
Checked:	Indicate whether the radio button has been selected or not

7.6.3 GUI DESIGN GUIDELINES FOR RADIO BUTTONS

Keep the following in mind when placing grouped radio buttons on a form:

- Use radio buttons when you want to limit the user to one of two or more choices when only one of those choices may be selected.
- The minimum number of radio buttons in a group is two and the recommended maximum is seven.
- It is good programming practice to designate a default radio button is each group of radio buttons.
- Radio buttons can be grouped together by means of a group box.
- Only one radio button in each group can be selected at a given time.
- Every radio button must have a text entry to indicate the purpose of the radio button.
- Use sentence capitalization for the optional identifying label in a group box control

7.6.4. EXAMPLE OF A RADIO BUTTON

If we modify the examples on pages 155 and 156 to enter the department by means of a radio button and we name the radio buttons radDeptA, radDeptB, radDeptC and radDeptD, the code can be implemented as follows:

```
If radDeptA.Checked Then
   lblDept.Text = "Department A"
ElseIf radDeptB.Checked Then
   lblDept.Text = "Department B"
ElseIf radDeptC.Checked Then
   lblDept.Text = "Department C"
ElseIf radDeptD.Checked Then
   lblDept.Text = "Department D"
Else
   lblDept.Text = "Please select the appropriate department"
End If
```

As mentioned earlier, it is good programming standard to assign a default radio button to a group. In such a case it will not be necessary to test whether a radio button has been selected or not.

7.7 PROGRAM EXAMPLES WITH NESTED IF-STATEMENTS

7.7.1 EXAMPLE 1

Problem statement:

The Bright Light Company is increasing the salaries of their employees according to their department as can be seen in the table below.

Department code	Percentage increase
A	7.2
B	6.8
other	6.3

7.7.1.1 PROGRAM PLANNING

Do the planning and write an algorithm to solve this problem. The user has to enter the department code and the current **annual** salary of the employee. Calculate and display the increased **monthly** salary. Use a nested if statement without AND and/or OR. Ensure that all data entered are correct.

	Description	Type	Name of variable
Input:	department code	character	deptCode
	annual salary	real	anSalary
Output:	new monthly salary	real	monSalary

I	P	O
deptCode anSalary	Prompt to read input fields Enter input fields Calculate monSalary Display monSalary	monSalary

Algorithm:

```
CalcNewMonSalary
~       Calculate the increased monthly salary
        ~ Enter input
        display "Provide the department code:"          ~ display on new line
        enter deptCode
        display "Provide annual salary:"                ~ display on new line
        enter anSalary
        if anSalary is numeric
                ~       Convert to monthly salary
                monSalary = anSalary / 12
                ~       Calculate increase according to department code
                if deptCode = "A" then
                        monSalary = monSalary + monSalary * 7.2 / 100
                else
                        if deptCode = "B" then
                                monSalary = monSalary + monSalary * 6.8 / 100
                        else
                                monSalary = monSalary + monSalary * 6.3 / 100
                        endif
                endif
                ~       display result
                display "The increased monthly salary is R" , monSalary
                                                        ~ display on new line
        else
                display "The annual salary must be numeric"
        endif
end
```

Test the program four times, to cover every outcome of the if-statement:

Department code = A Annual salary R24000.00
Department code = B Annual salary R38500.00
Department code = G Annual salary R30000.00
Department code = C Annual salary = R5OOOO.OO

When testing the program we will have to include incorrect data as well. In the last example, O's in stead of zeros will cause an error.

Output:

```
Provide the department code:  A
Provide annual salary:  24000.00
The increased monthly salary is R2144.00

Provide the department code: B
Provide annual salary: 38500.00
The increased monthly salary is R3426.50

Provide the department code: G
Provide annual salary: 30000.00
The increased monthly salary is R2657.50

Provide the department code: C
Provide annual salary: 5OOOO.OO
The annual salary must be numeric
```

7.7.1.2 IMPLEMENTATION IN VB.NET

The user interface in **Figure 7-1** has been developed for this example. It contains **radio buttons** and the code contains a nested if-statement. The default radio button is set to "other"-department and if a user does not select A or B it will automatically default to the last else of the nested if-statement. The code for the btnCalc button click event will only execute if a valid annual salary has been entered.

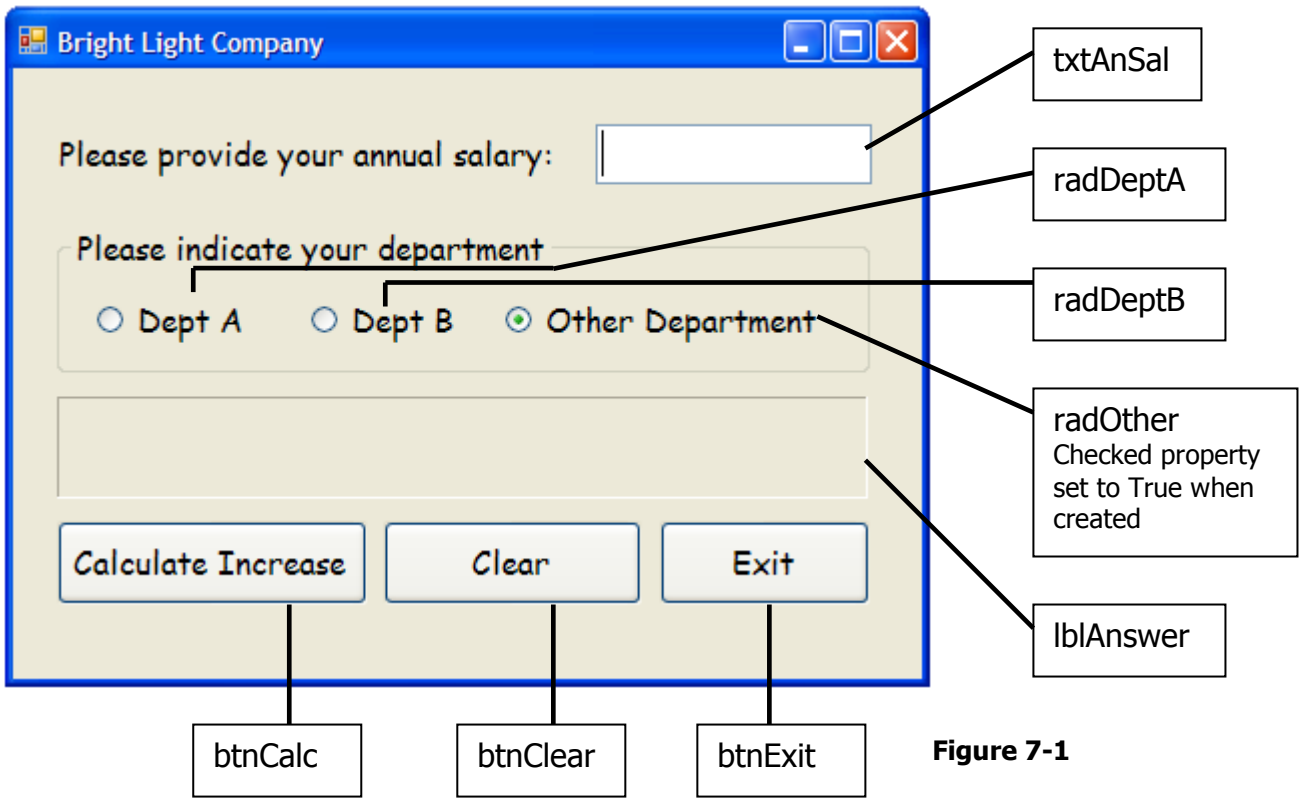


Figure 7-1

CODE IN THE CODE EDITOR

```
Public Class frmIncrease

Private Sub btnCalc_Click(. . .) Handles btnCalc.Click
   Dim decAnSal, decMonSal As Decimal
   If Decimal.TryParse(txtAnSal.Text, decAnSal) Then
      decMonSal = decAnSal / 12
      If radDeptA.Checked Then
         decMonSal = decMonSal + decMonSal * 7.2 / 100
      Else
         If radDeptB.Checked Then
            decMonSal = decMonSal + decMonSal * 6.8 / 100
         Else
            decMonSal = decMonSal + decMonSal * 6.3 / 100
         End If
      End If
      lblAnswer.Text = "The increased monthly salary = R" & _
                       decMonSal.ToString("N2")
    Else
      MessageBox.Show("The annual salary must be numeric", "Error", _
                      MessageBoxButtons.OK, MessageBoxIcon.Information)
    End If
End Sub

Private Sub btnClear_Click(. . .) Handles btnClear.Click
    txtAnSal.Text = ""
    radDeptA.Checked = False
    radDeptB.Checked = False
    radOther.Checked = True
    lblAnswer.Text = ""
    txtAnSal.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
    Close()
End Sub

End Class
```

No need to test if radOther radio button has been selected, because it will be the default button.

Set as the default button when controls are cleared

EXAMPLE OF OUTPUT

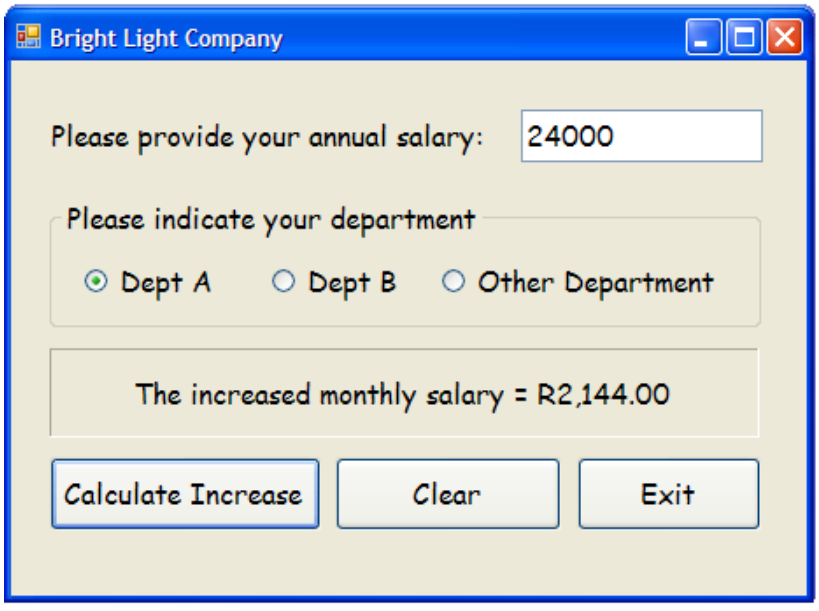


Figure 7-2

EXAMPLE OF INCORRECT INPUT:

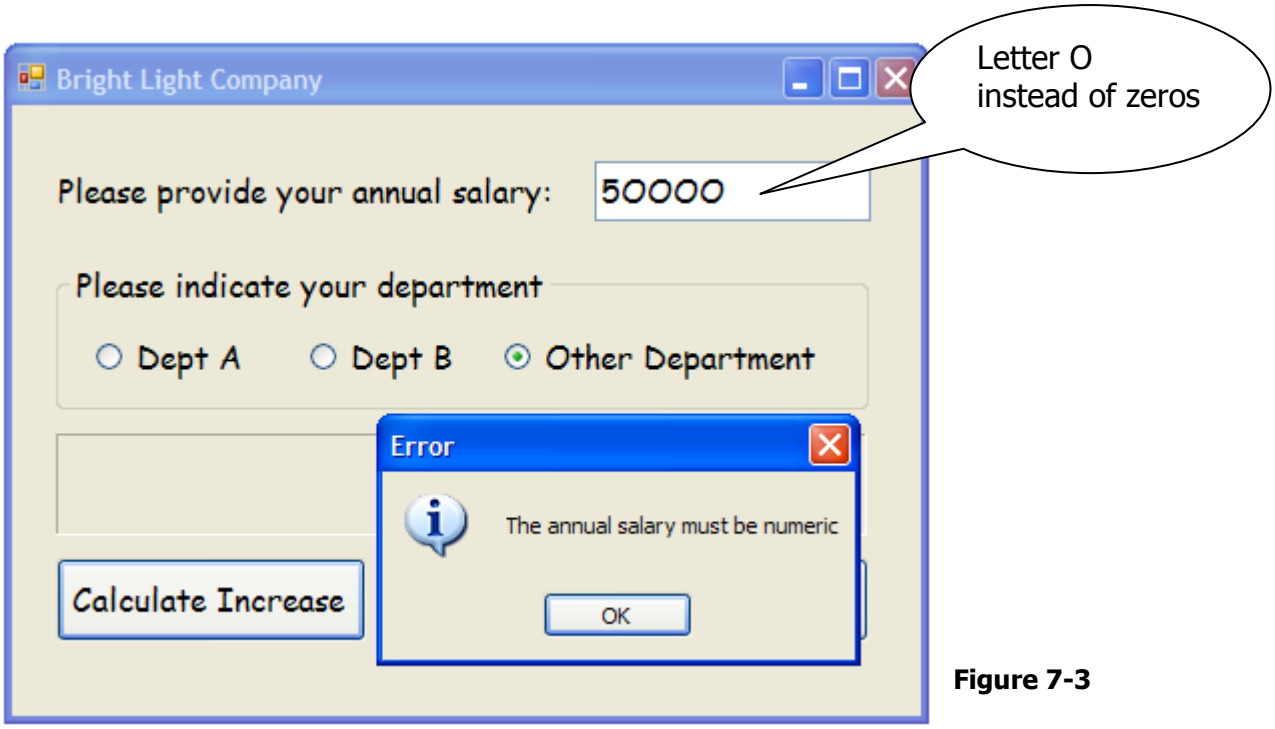


Figure 7-3

Take note: Although only 2 examples of output screens are provided, you must test your program with all input possibilities, as indicated in our planning, to ensure that the program works in the correct way.

7.7.2 EXAMPLE 2

A vet prescribes medicine for dogs to keep them healthy and strong. The daily dosage depends on the weight of the dog and is calculated according to the following table:

Weight of dog in kg	Milliliters medicine per 500gm of weight
< 5	1.0
5 - 8	0.9
>8 – 12	0.75
>12	0.6

The user must enter the name and the weight of the dog in kilogram (to the nearest 500 gram) and then the program must calculate and display the daily dosage that must be administered to the dog as well as the dog's name.

7.7.2.1 PROGRAM PLANNING

	Description	Type	Name of variable
Input:	name of dog	string	name
	weight of dog	real	dogWeight
Output:	name of dog	string	name
	daily medicine	real	medicine

I	P	O
name dogWeight	Prompt for input data Enter input data Calculate medicine Display name, medicine	name medicine

Algorithm:

```
CalcMedicine
~       The algorithm will test whether the name for the dog has been entered and whether the
~       weight for the dog is numeric and is in kg, to the nearest 500g (e.g. 3.5, 5, 6.5). A weight
~       of 7.8 will be invalid. The dosage will only be calculated and displayed if the input is valid
        ~       Enter name of dog
        display "Provide the name of the dog:"
        enter name
        ~       Test if name has been entered
        If      name = " "      ~ Outer if-statement
                Display "The name of the dog must be entered"
        else
                ~ Enter weight only if name has been entered
                display "Provide the weight of dog in kg (to nearest 500 gm):"
                enter dogWeight
                ~  Test if weight is numeric, in which case it must be converted to the equivalent
                ~  number in 500g
                If      dogWeight is numeric            ~  Inner if-statement 1
                        number500 = dogWeight * 2
                        ~       Test if weight entered was to the nearest 500g. If the weight in kg
                        ~       is 3.5, the weight in 500 gm will be 7, If the weight in kg is 3, the
                        ~       weight in 500 gm will be 6. Therefore the weight in 500g will
                        ~       never contain a remainder.
                        If      number500 MOD 1 = 0             ~ Inner if-statement 2
                                If      dogWeight < 5  then
                                        medicine = number500 * 1
                                else
                                        if      dogWeight <= 8 then    ~ See comment 1
                                                medicine = number500 * 0.9
                                        else
                                                if      dogWeight <= 12 then  ~ See
comment2
                                                        medicine = number500 * 0.75
                                                else
                                                        medicine = number500 * 0.6
                                                endif
                                        endif
                                endif
                                display  name, " must receive a daily dose of  ", medicine, "ml"
```

```
                    else            ~ else-clause for  inner if-statement 2
                            display "The weight must be in kg and to the nearest 500g,
                                    e.g. 3, 3.5 or 4"
                    endif
            else                    ~ else-clause for inner if-statement 1
                    display "The weight entered for ", name, " must be numeric"
            endif
    endif
end
```

Comment 1:

In every one of the if-statements in the program above, it only tested one condition of weight e.g. the 2nd if statement:

```
if weight <= 8
```

It did not test if weight >= 5 AND weight <= 8, because the condition where weight is compared to 5 was already tested in the first if statement. When control is passed to the 2nd if statement (the else part), the weight is definitely greater or equal to 5!

Comment 2:

In a similar way it is not necessary to test if weight > 8 AND weight <= 12. In the last inner if-statement it is only necessary to test if weight <= 12 as it will already be greater than 8 when it reaches this part of the if-statement.

When testing this program it is important to use test data that will test all the borderline cases to ensure that this program yields the correct output in all cases. It is also necessary to test the program with incorrect input data to see if applicable error messages will be displayed.

Test the program as follows:

Name of dog: Any name
Test weights: 3.5, 5.0, 8.0, 12.0 and 25.5 kg
Then test the program with weights 1O (O in stead of 0) and 10.8
Lastly test the program where the name for the dog is omitted.

Output:

```
Provide the name of the dog: Fluffie
Provide the weight of dog in kg (to nearest 500 gm): 3.5
Fluffie must receive a daily dose of 7.0 ml

Provide the name of the dog: Daisy
Provide the weight of dog in kg (to nearest 500 gm): 5.0
Daisy must receive a daily dose of 9.0 ml
```

Output (Continued):

```
Provide the name for the dog: Georgy Girl
Provide the weight of dog in kg (to nearest 500 gm): 8.0
Georgy Girl must receive a daily dose of 14.4 ml

Provide the name for the dog: Danny Boy
Provide the weight of dog in kg (to nearest 500 gm): 12.0
Danny Boy must receive a daily dose of 18.0 ml

Provide the name for the dog: Victor
Provide the weight of dog in kg (to nearest 500 gm): 25.5
Victor must receive a daily dose of 30.6 ml

Provide the name for the dog: Lady
Provide the weight of dog in kg (to nearest 500 gm): 1O
The weight entered for Lady must be numeric

Provide the name for the dog: Tinky
Provide the weight of dog in kg (to nearest 500 gm): 3.8
The weight must be in kg and to the nearest 500g, e.g. 3, 3.5 or 4.

Provide the name for the dog:
The name of the dog must be entered
```

7.7.2.2 IMPLEMENTATION IN VB.NET

The user interface has been designed as indicated in **Figure 7-4**.

The names for the controls are as follows:

- txtName
- txtDogWeight
- lblAnswer
- btnDosage
- btnClear

In the corresponding code for the btndosage click event, the String.IsNullOrEmpty method and TryParse method have been used to test whether the input data has been entered and whether it is in the correct format. The statement to test if the weight is in kg and to the nearest 500g is similar to that in the algorithm. The program uses the If-ElseIf structure.



Figure 7-4

CODE IN THE CODE EDITOR

```
Option Explicit On
Option Strict On

Public Class frmHealthyDog

Private Sub btnCalc_Click(. . .) Handles btnCalc.Click
   Dim strName As String
   Dim sngDogWeight As Single
   Dim sngNumber500 As Single
   Dim sngMedicine As Single
   If String.IsNullOrEmpty(txtName.Text) Then
      MessageBox.Show("The name of the dog must be entered", _
                      "Error, please correct", MessageBoxButtons.OK, _
                      MessageBoxIcon.Information)
   Else
      strName = txtName.Text
      If Single.TryParse(txtDogWeight.Text, sngDogWeight) Then
         sngNumber500 = sngDogWeight * 2
         If sngNumber500 Mod 1 = 0 Then
            If sngDogWeight < 5 Then
               sngMedicine = sngNumber500
            ElseIf sngDogWeight <= 8 Then
               sngMedicine = sngNumber500 * 0.9D
            ElseIf sngDogWeight <= 12 Then
               sngMedicine = sngNumber500 * 0.75D
            Else
               sngMedicine = sngNumber500 * 0.6D
            End If
```

Data Validation to test whether a name for the dog has been entered

Data validation to test if weight is in correct format

Second test to see if weight is in kg to nearest 500g

Use of If-Elseif statement to simplify the nested structure. It is also necessary to use only one End If to end off the complete structure.

```
            lblAnswer.Text = strName & " must receive a daily dose of " _
                                     & sngMedicine.ToString("N1") & " ml"
         Else
            MessageBox.Show_
            "Weight must be in kg, rounded to the nearest 500g, e.g. 3, 3.5 or 4 kg",_
                        "Error, please correct", MessageBoxButtons.OK, _
                        MessageBoxIcon.Information)
         End If
      Else
         MessageBox.Show("The weight entered for " & strName & _
                         " must be numeric", "Error, please correct", _
                         MessageBoxButtons.OK,
MessageBoxIcon.Information)
      End If
   End If
End Sub

Private Sub btnClear_Click(. . .) Handles btnClear.Click
   txtName.Text = ""
   txtDogWeight.Text = ""
   lblAnswer.Text = ""
   txtName.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
    Close()
End Sub

End Class
```

EXAMPLES OF OUTPUT

Examples of valid output are provided in **Figures 7-5** and **Figure 7-6**.



Figure 7-5



Figure 7-6

Examples of incorrect data are provided in **Figure 7-7** to **Figure 7-9**:



Figure 7-7



Figure 7-8



Figure 7-9

7.7.3 EXAMPLE 3

The price for hiring a car per day from the Reliable Car Hire Company depends on the type of car hired by a customer. The customer may choose between small (code = S), medium (M) and large (L) cars as can be seen in the following table:

Type of Car	Daily Price in Rand
Small	200
Medium	260
Large	400

The user is asked to enter the type of car he needs as well as the number of days hired. Calculate and display the total amount he or she has to pay using a nested if statement. There is however a special offer at this moment on the hiring of small cars and a discount of 12.5% is given. Provide for correct input data.

7.7.3.1 PROGRAM PLANNING

	Description	Type	Name of variable
Input:	type of car	character	code
	number of days	integer	noDays
Output:	amount due	real	amtDue

I	P	O
code noDays	Prompt for input fields Enter input fields Validate data Calculate amtDue Display amtDue	amtDue

Algorithm:

CalcAmtDue

```
~       Calculate the amount to hire a car
        ~ Input
        display "Enter the type of car you need  S, M or L:"    ~ display on new line
        enter code
        display "Enter number of days for car hire:"            ~ display on new line
        enter noDays
        ~       Validate and calculate amount
        if noDays not numeric then          ~ see example in VB.NET
                display "Number of days entered not numeric"    ~ display on new line
                display "You entered  ", noDays                 ~ display on new line
```

```
        else
                if code = "S"' or code = "s" then
                        amtDue = noDays * 200
                        discount = amtDue *  0.125
                        amtDue = amtDue – discount
                        display "The amount due is R", amtDue  ~ display on new line
                else
                        if code = "M" or code = "m" then
                                amtDue = noDays * 260
                                display "The amount due is R", amtDue
                                                        ~ display on new line
                        else
                                if code = "L" or code = "l" then
                                        amtDue = noDays * 400
                                        display "The amount due is R", amtDue
                                                        ~ display on new line
                                else
                                        display "Wrong code entered, only S, M or L"
                                                        ~ display on new line
                                        display "You entered the code  ", code
                                                        ~ display on new line
                                endif
                        endif
                endif
        endif
end
```

When testing this program we have to include incorrect data. We will include a type G car in our test data as well as a non-numeric value for number of days. We will also include a small car because of the special offer, as well as a normal transaction.

Test the program as follows:

Type of car:	Small	Number of days:	4
	Large		3
	Grand		9
	Medium		K

Output:

```
Enter the type of car you need S, M or L: S
Enter number of days for car hire: 4
The amount due is R700.00
```

Output (continued):

```
Enter the type of car you need  S, M or L:  L
Enter number of days for car hire:  3
The amount due is R1200.00

Enter the type of car you need  S, M or L:  G
Enter number of days for car hire:  9
Wrong code entered, only S, M or L
You entered the code G

Enter the type of car you need  S, M or L:  M
Enter number of days for car hire:  K
Number of days entered not numeric
You entered  K
```

7.7.3.2 IMPLEMENTATION IN VB.NET

The user interface has been designed as indicated in **Figure 7-10**.

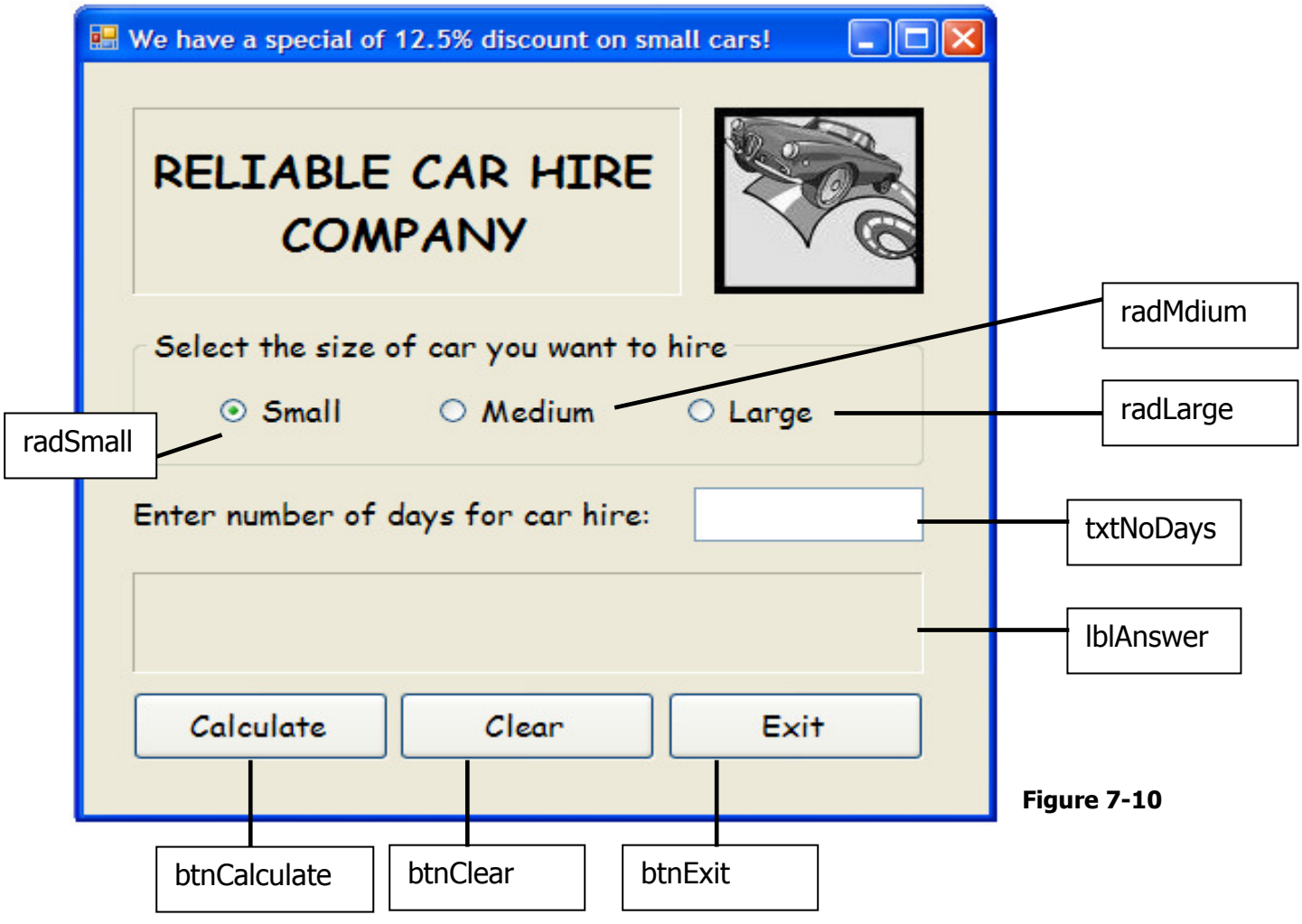


Figure 7-10

The fact that the type of car to be hired is indicated by means of a radio button will simplify the code because now it will not be necessary to test if the user entered an incorrect code for the type of car. We can also specify that the small car must be the default button, because the company has a special on small cars, and therefore we can assume that most people would prefer to hire small cars.

CODE IN THE CODE EDITOR

```
Option Explicit On
Option Strict On

Public Class frmCarHire

Private Sub btnCalculate_Click(. . .) Handles btnCalculate.Click
   Dim intNoDays As Integer
   Dim decAmtDue As Decimal
   Dim decDiscount As Decimal
   If Integer.TryParse(txtNoDays.Text, intNoDays) Then
      If radSmall.Checked Then
         decAmtDue = intNoDays * 200D
         decDiscount = decAmtDue * 0.125D
         decAmtDue = decAmtDue - decDiscount
      Else
         If radMedium.Checked Then
            decAmtDue = intNoDays * 260D
         Else
            decAmtDue = intNoDays * 400D
         End If
      End If
      lblAnswer.Text = "The amount due = R" & _
                       decAmtDue.ToString("N2")
   Else
      MessageBox.Show("Number of days not an integer, you
entered"_
                & txtNoDays.Text, "Error", MessageBoxButtons.OK, _
                 MessageBoxIcon.Information)
   End If
End Sub
```

The code for the click event of the button btnCalculate is much shorter than in the algorithm because radio buttons are used. It is therefore not necessary to test for an invalid car size code. The answer has also been displayed only once at the end of the nested if-statement which is more effective.

```
Private Sub btnClear_Click(. . .) Handles btnClear.Click
   txtNoDays.Text = ""
   radSmall.Checked = True
   radMedium.Checked = False
   radLarge.Checked = False
   lblAnswer.Text = ""
   txtNoDays.Focus()
End Sub
```

Set default button

EXAMPLES OF OUTPUT

Examples of output are provided in **Figure 7-11** to **Figure 7-14.**



Figure 7-11



Figure 7-12

Because the TryParse statement not only tested that the number of days must be numeric, but that it must also be converted to a variable of data type integer, both the following cases will result in an error.



Figure 7-13



Figure 7-14



Exercises

Write nested if statements without AND and/or OR for each of the following:

1. Calculate the medical aid contribution of an employee. The amount is calculated according to the following table. You may assume that his salary is stored in the variable called *salary* and his number of dependents is stored in the variable *noDepend.*

Monthly Salary	Dependents	Percentage of salary
0 – 4999.99	any number	2%
5000 and more	0	4%
	1 and more	3.5%

2. If the field ABC has a negative value, calculate the value of XYZ by multiplying the value of ABC by 2.5, but if the value of ABC is positive, store 35% of ABC in XYZ. If the value of ABC is zero, decrease the current value of XYZ by 8.3%.

3. The variable *person* contains a code of A for adult or a C for child while the variable *member* contains a TRUE or FALSE to indicate if a person is a member of a club or not. The club is having a fun day and the entrance fee is determined as follows: An adult member pays R25 and a child member pays R8. A non-member pays 50% more than a member. Store the entrance fee in the variable called *fee.*



Practical

1. Poppy who always counts her cents to buy the most for her money, needs to buy washing powder. She went to the shop and wrote down the prices of 500 gm, 750 gm and 1 kg washing powder. She wants to write a program to determine which one of the packets is the best buy. Do the calculations and display which packet she has to buy. The user is asked to enter the prices of the different packets.

You may test your program with the following test data:

500 gm.	R14.85
750 gm.	R21.95
1 kg.	R29.83

You may not use these values as fixed values in your program - only as test values.

2. Johnny has saved some money in the bank on which he earns an annual interest of 9.5%. The user must enter the amount he has saved and then calculate the amount of interest he receives monthly. For the sake of the calculation you may assume that an equal amount of interest is paid per month (disregard the different number of days per month). Calculate and display the income tax he has to pay on the interest earned according to the following table:

Amount of interest	**Percentage tax payable**
Less than R1000	0
R1000.01 – R2000	7.5% on the amount > R1000
>= R2000.01	9.5% on the amount > R1100

3. The Anything Company has representatives that sell products to the public. These representatives (code = REP) are organized in different areas. Every area has an area manager (code = AM). A region contains 5 areas and has a regional manager (code = RM).
The user enters the name, code and amount of sales of the representative, area manager or regional manager (only one of the three). A representative receives a commission of 20%, another 5% goes to the area manager and another 2% goes to the regional manager. Calculate and display how much every person will receive for a transaction. Provide for incorrect input values.

Note:

- If the representative sold the products, he, the area manager and the regional manager will receive commission.
- If the area manager sold the products, only he and the regional manager will receive commission.
- If the regional manager sold the products, he will be the only one to receive the total amount of commission.

For each of these practical assignments, design a suitable, user friendly interface and code as effective as possible.

7.8 THE SELECT CASE STRUCTURE

The select case structure is another form of selection to do different processing steps depending on a condition.

Study the following if-statement:

```
if  number = 1 then
        display "group 1"
else
        if number = 3 then
                display "group 3"
        else
                display "group unknown"
        endif
endif
```

The same condition can be tested as follows in a select case statement, in a much easier readable way:

```
select case number                  ~ indicate the variable to be tested
        case 1
                display "group 1"
        case 3
                display "group 3"
        case else
                display "unknown"   ~  if no match is found
endselect
```

There may also be more than one statement for an outcome of a specific condition.

In the following example different ranges of values are tested to calculate a discounted amount and display the answer.

```
select case noItems
        case  1 to 10
                percentage = 5
                display "You receive 5% discount"
        case 11 to 20
                percentage = 6.4
                display "You receive 6.4% discount"
        case > 20
                percentage = 7
                display "You receive 7% discount"
endselect
amount = (noItems * price) – (noItems * price * percentage / 100)
display "The amount is R", amount
```



Questions

True or False?

- If none of the matches in the case statement is true, it is not possible to do any processing regarding the value that is entered.
- The case structure can test a maximum of 5 different options.
- The different cases in the case statement must be in ascending or descending order.
- The last line in a case structure always contains the word endselect.
- It is possible to use an if-statement or a case-statement to test any condition.

7.9 THE SELECT CASE STRUCTURE IN VB.NET

The code for the select case structure in VB.NET is exactly the same as indicated in the pseudocode.

The End Select will automatically be inserted, similar to the End If for an if-statement. The code will also automatically be indented. Minor typing errors will also automatically be corrected by the compiler.

E.g. Case > 10 will automatically be changed to Case Is > 10.

When you use the **To** keyword, the value preceding the **To** must always be less than the value following the **To**.
An entry such as Case 20 to 5 will not give an error message, but the results will be incorrect. Another reason why it is important to test your code thoroughly for all input possibilities.

Example:

```
Select Case intNumber
     Case 1
          lblMessage.Text = “Group 1”
     Case 3
          lblMessage.Text = “Group 3”
     Case 5
          lblMessage.Text = “Group 5”
     Case Else
          lblMessage.Text = “Unknown”
End Select
```

7.10 EXAMPLES CONTAINING THE SELECT CASE STRUCTURE

Example 1

Write a select case statement to calculate the new value of the variable *price* depending on the *grade* according to the table below.

grade	Processing of price
A	Double the price
B	Add sales tax of 14%
C	Subtract a discount of 8.75%
other	Add R5 if the price is more than R50 Subtract R4 if the price is less or equal to R50

```
select case grade
        case "A"      price = price * 2
        case "B"      price = price * 1.14
        case "C"      price = price * 0.9125      ~ (1 - .0875 = 0.9125)
        case else     if price > 50 then
                              price = price + 5
                      else
                              price = price – 4
                      endif
endselect
```

Example 2

Write a select case statement for the following: if the name is Sally or Tania, display the message "she is a girl"; but if the name is Jim, John or Russell, display the message "he is a boy".

```
select case name
        case "Sally", "Tania"              display "she is a girl"
        case "Jim", "John", "Russell"      display "he is a boy"
endselect
```

7.11 PROGRAMMING EXAMPLE CONTAINING THE SELECT CASE STRUCTURE

Problem:

Write an algorithm and VB.NET application to solve the following problem:

Summer has to read a number of prescribed books and she has decided to reward herself by watching TV for an allocated number of minutes according to the number of pages she has read. Enter the number of pages

read, determine and display the minutes using a select case statement. If more than 400 pages are entered, an error message must be displayed, because none of the prescribed books have that many pages!

Pages read	Minutes
0 - 20	0
21 - 50	10
51 - 100	40
More than 100	75

7.11.1. PROGRAM PLANNING

	Description	Type	Name of variable
Input:	number of pages	integer	pages
Output:	number of minutes	integer	minutes
	error message	string	message

I	P	O
pages	prompt for pages enter pages determine minutes display minutes / message	minutes message

Algorithm:

```
RewardingMinutes
    ~ Initialise a boolean variable to assume that input entered is valid
    validPages = True
    ~ Input
    display "Enter the number of pages read: "  ~ display on new line
    enter pages
    ~ Validate input
    If pages is numeric
        select case pages
            case 0 to 20
                minutes = 0
            case 21 to 50
                minutes = 10
            case 51 to 100
                minutes = 40
            case 101 to 400
                minutes = 75
            case else
                display "The pages must be > 0 and not more than 400"
                validPages = False
        endselect
    else
        display "The pages entered must be a valid integer"
    endif
```

```
        ~ Display output only if input was valid. Display a different message if Summer
        ~ is not allowed to watch TV
        if      minutes = 0
                display "You must read more than 20 pages to watch TV,
                        Better luck next time"
        else
                display "Good girl!"                                    ~ on a new line
                display "You may watch TV for ", minutes, " minutes"    ~ on a new line
        endif
end
```

Test data:

Pages = 45, 432, -6, 255, x

Output:

```
Enter the number of pages read:  45
Good girl!
You may watch TV for 10 minutes

Enter the number of pages read:  432
The pages must be > 0 and not more than 400

Enter the number of pages read:  -6
The pages must be > 0 and not more than 400

Enter the number of pages read:  255
Good girl!
You may watch TV for 75 minutes

Enter the number of pages read: x
The pages entered must be a valid integer
```

7.11.2 IMPLEMENTATION IN VB.NET

The user interface has been designed as indicated in **Figure 7-15**.

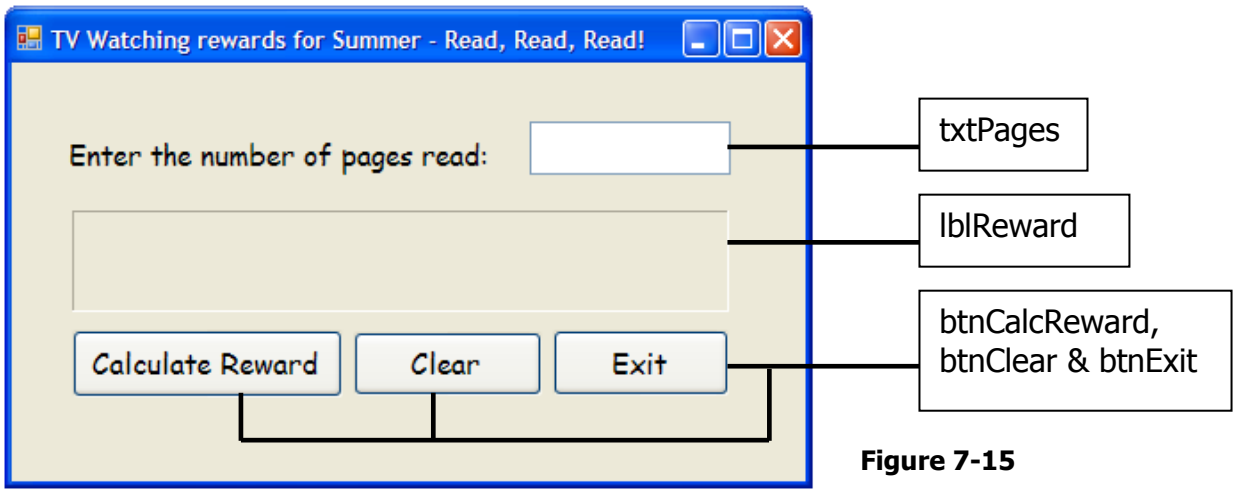


Figure 7-15

CODE IN THE CODE EDITOR

```
Public Class frmSummerReading

Private Sub btnCalcReward_Click(. . .) Handles btnCalcReward.Click
   Dim intPages, intMinutes As Integer
   Dim blnValid As Boolean = True
   If Integer.TryParse(txtPages.Text, intPages) Then
      Select Case intPages
         Case 0 To 20
              intMinutes = 0
         Case 21 To 50
              intMinutes = 10
         Case 51 To 100
              intMinutes = 40
         Case 101 To 400
              intMinutes = 75
         Case Else
              MessageBox.Show_
                "The pages must be > 0 and not greater than 400",_
                "Error", MessageBoxButtons.OK, _
                MessageBoxIcon.Exclamation)
                blnValid = False
      End Select
      If blnValid Then
         If intMinutes = 0 Then
            lblReward.Text = _
              "You must read more than 20 pages to watch TV" & _
              ControlChars.NewLine & "Better luck next time!"
         Else
            lblReward.Text = "Good girl!" & ControlChars.NewLine _
              & "You may watch TV for " & intMinutes & " minutes"
         End If
      Else
         MessageBox.Show_
           ("The pages entered must be a valid integer", "Error",_
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
      End If
   End if
End Sub

Private Sub btnClear_Click(. . .) Handles btnClear.Click
   txtPages.Text = ""
   lblReward.Text = ""
   txtPages.Focus()
End Sub

Private Sub btnExit_Click(. . .) Handles btnExit.Click
   Close()
End Sub

End Class
```

EXAMPLES OF OUTPUT

Examples of output due to invalid input are provided in **Figure 7-16** to **Figure 7-18.**

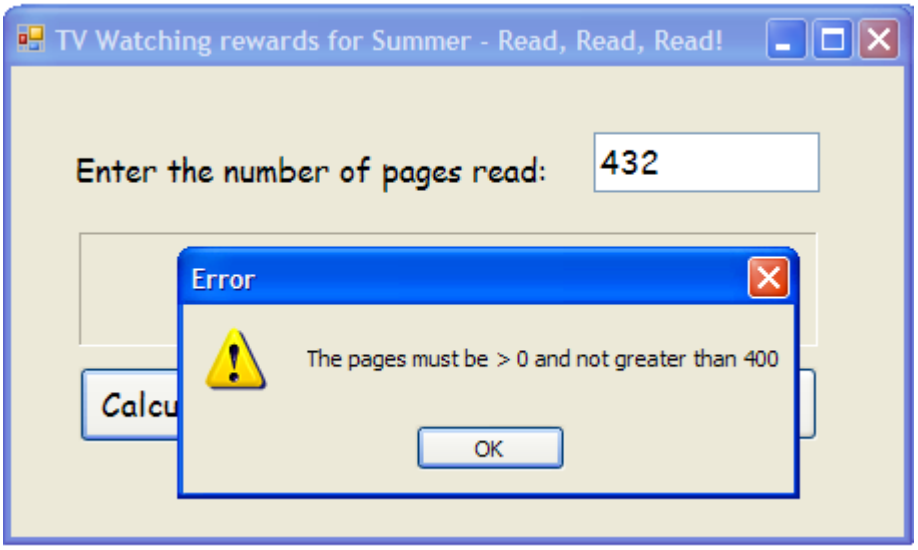


Figure 7-16

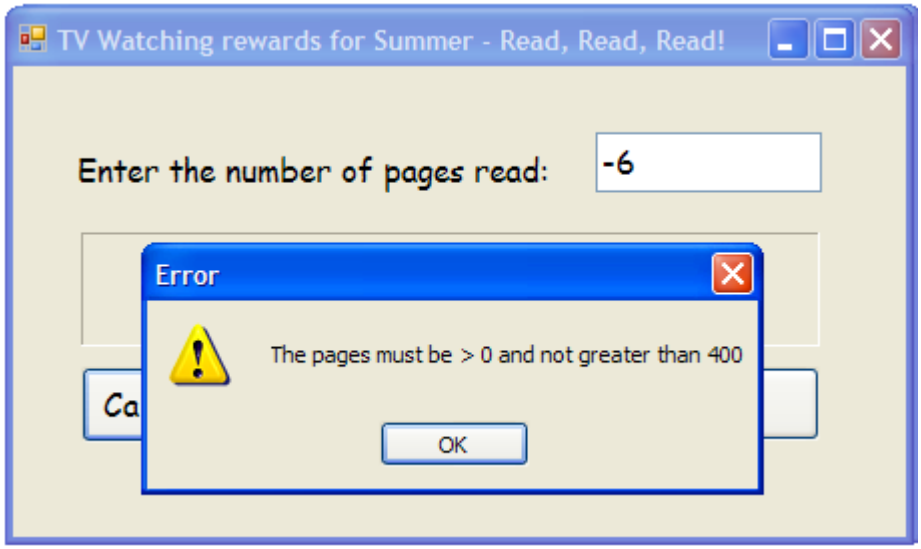


Figure 7-17

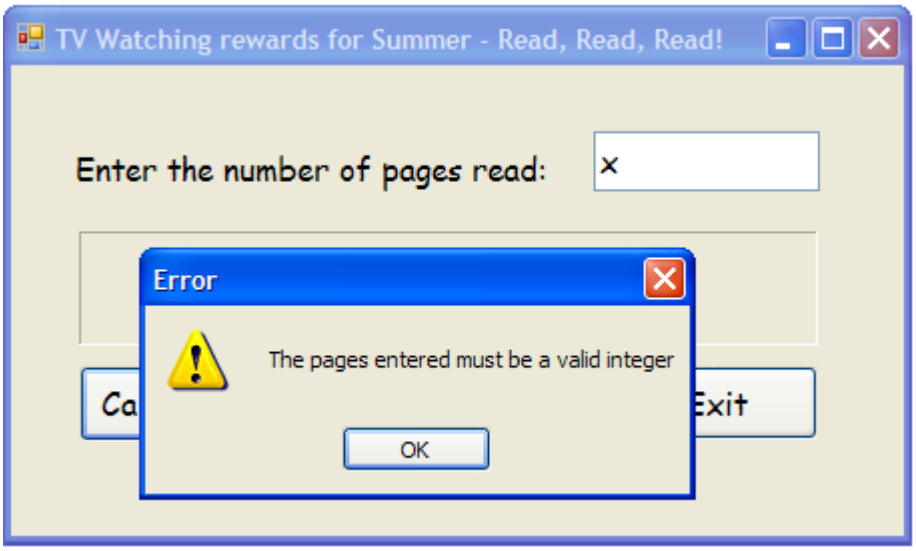


Figure 7-18

Examples of output produced due to valid input are given in **Figure 7-19** to **Figure 7-21**.



Figure 7-19



Figure 7-20

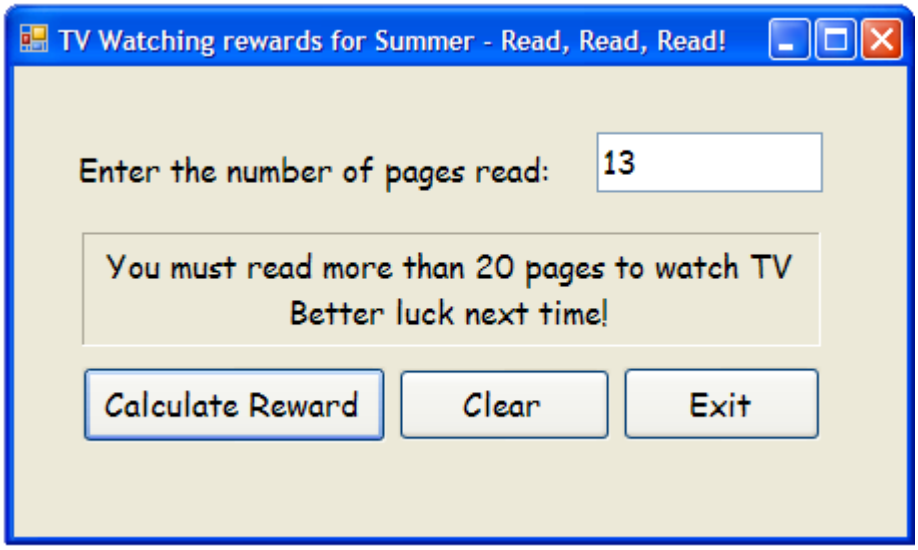


Figure 7-21



Exercises

1. Write only the select case statement for the following:

   If the integer variable *xyz* contains a 7, 8 or 15, display 3 asterisks, but if *xyz* has a negative value, 5 asterisks must be displayed. However, if the value is from 10 to 20, then 7 asterisks must be displayed. If *xyz* contains any other value, display 4 equal signs.

2. Rewrite the following IF-statement as a CASE-statement.

```
If      code = 'S'
        sport = "Soccer"
Else
        If      code = 'R'
                sport = "Rugby"
        Else
                If      code = 'T'
                        sport = "Tennis"
                Else
                        If      code = 'H'
                                sport = "Hockey"
                        Else
                                If      code = 'B'
                                        sport = "Basketball"
                                Else
                                        sport = "Unknown"
                                Endif
                        Endif
                Endif
        Endif
Endif
```



Practical

1. Rewrite this program that you have done earlier in this chapter, but use a select case statement instead of a nested if statement.
   Johnny has saved some money in the bank on which he earns a annual interest of 9.5%. The user must enter the amount he has saved and then calculate the amount of interest he receives monthly. For the sake of the calculation you may assume that an equal amount of interest is paid per month (disregard the different number of days per month). Calculate and display the income tax he has to pay on the interest earned according to the following table:

Amount of interest	Percentage tax payable
Less than R1000	0
R1000.01 – R2000	7.5% on the amount > R1000
R2000.01 – R3200	9.5% on the amount > R1100

2. Enock has a gardening service where he and his workers get paid per hour worked as can been seen in the table below. He also charges an additional R2.50 per hour for equipment used. Enter the hours worked as a real number and calculate and display the amount owed to him. Use a select case statement to calculate the amount.

Hours worked	Payment per hour
0 - 2	R20.00
>2 – 4.5	R19.20
4.6 – less than 6	R18.20
6 and more hours	R17.88

For each of these practical assignments, design a suitable, user friendly interface and code as effective as possible.

CHAPTER 8
ITERATION USING A FIXED COUNT LOOP

8.1 INTRODUCTION

Programming very often includes repeating a set of instructions a number of times. Sometimes we know exactly how many times we need to repeat the instructions and other times we do not know. This type of programming is called iteration or looping. In this chapter we will discuss the for-loop where we know exactly how many times to repeat a set of instructions.

OUTCOMES

When you have studied this chapter you should be able to

- write a loop in pseudocode and in VB.NET by means of the for-next-statement,
- know the purpose of an accumulator and implement it in a solution,
- know when to display output during every execution of a loop and when to display it at the end of the loop,
- know how to add a list box control to a form, how to set the properties of a list box and how to add items to it,
- use an input box function to enter data during every execution of a loop,
- plan and write VB.NET programs for problems that can be solved with an automatic counter loop, in other words where the number of executions of the loop is known.

8.2 THE FOR-NEXT-LOOP

The syntax of the for-loop is as follows:

```
for variable1 = begin-value to end-value [step variable2]
        statement-1
        statement-2          ~ body of the for-statement
        :
        Statement-n
next variable1
```

Variable1 indicates a variable that contains the initial value of begin-value. The statements in the body of the loop will then be executed after which variable1 will be incremented by variable2. This process will continue until the value of variable1 has moved past end-value for the first time. Variable1 is called the index of the for-loop.

In the case where [step variable2] is omitted a default value of +1 is used. There may be one or many statements in the body of the for-loop.
The step may be negative, in which case begin-value must be bigger than end-value.

It is not necessary that the variables must be integers. The step may be a fraction, in which case all variables may be of the data type decimal.

8.3 EXAMPLES

8.3.1 EXAMPLE 1

Use the for-statement to display the consecutive numbers from 1 to 10.

```
k = 1
for i = 1 to 10
        display k, " "   ~ on the same line
        k = k + 1
next i
```

The output will be
1 2 3 4 5 6 7 8 9 10

This for statement can be written more effectively and using less steps by using the index i as follows:

```
for i = 1 to 10
        display i, " "  ~ on the same line
next i
```

These statements will produce the same result as above.

8.3.2 EXAMPLE 2

Use the for-statement to display the consecutive numbers from 1 to 10 in descending sequence.

The step must now be negative and the above statements must change as follows:

```
for i = 10 to 1 step -1
        display i, " "   ~ on the same line
next i
```

The output will now be
10 9 8 7 6 5 4 3 2 1



Questions

What will the output be of the following for-loops?

1.
```
for x = 0.5 to 5.5 step 0.5
        display x, " "     ~ on the same line
next x
```

2.
```
for x = 1 to 15 step 2
        display x, " "     ~ on the same line
next x
```

3.
```
for x = 2 to 18 step 3
        display x          ~ on a new line
next x
```

4.
```
for y = 5 to 0 step -1
        display y          ~ on a new line
next y
```

8.3.3 EXAMPLE 3

Calculate and display the sum of the first 5 odd numbers:

```
sum = 0                 ~ initialize the value of sum in the beginning
odd = 1                 ~ first odd number
for k = 1 to 5
        sum = sum + odd
        odd = odd + 2       ~ move to the next odd number
next k                      ~ move to the next value of k
display "The sum of the first 5 odd numbers is ", sum
```

People often think that they only need the sum right at the end of the processing, so they are only going to calculate the sum just before it is needed. But it simply does not work that way!

Right at the end, only the last odd number is available. So, whenever the number (in this case the next odd number) is available, it must be added to the sum. Every number will only be available in the current execution of the loop.

Let us do a trace table to make it more understandable:

Instruction	sum	odd	k	Output
Assignment	0			
Assignment		1		
For			1	
Calculation	1			
Calculation		3		
For			2	
Calculation	4			
Calculation		5		
For			3	
Calculation	9			
Calculation		7		
For			4	
Calculation	16			
Calculation		9		
for			5	
calculation	25			
calculation		11		
for			**6**	
display				The sum of the first 5 odd numbers is 25

Note that in this example, the answer is only displayed once, after the loop has terminated.

> The technique used above to add or summate inside a loop, is sometimes referred to as **accumulation.** The variable, sum, is called an **accumulator**. Accumulation is often used in problem solving to calculate a total or an average. If you only add 1 to the accumulator during every execution of the loop, the accumulator is called a **counter**.

8.3.4 EXAMPLE 4

Display the first 6 multiples of 5 (starting at 5) as well as their sum.

Once again, we need the sum of all these numbers right at the end. We can only calculate the sum if we add every number (multiple of 5) to the sum when it is available during every execution of the loop.

The sum must be displayed only once, at the end of the loop. However, we need to display every new multiple of 5 before we add it to the sum. We will therefore also need a **display statement inside the loop**. We can display it just after we have added it to the sum, as long as we display it before the next multiple of 5 is calculated.

The loop will have to execute 6 times, because we need to calculate 6 multiples of 5.

Algorithm

```
MultiplesOf5
        sum = 0                          ~ initialize the accumulator
        multiple = 5                     ~ first multiple of 5
        display "The first 6 multiples of 5 and their sum:"
        for x = 1 to 6
                display multiple         ~ on a new line
                sum = sum + multiple     ~ accumulate
                multiple = multiple + 5  ~ calculate the next multiple of 5
        next x
        display "Sum = ", sum
end
```

Instruction	sum	multiple	x	Output
Assignment	0			
Assignment		5		
Display				The first 6 multiples of 5 and their sum:
For			1	
Display				5
Calculation	5			
Calculation		10		
For			2	
Display				10
Calculation	15			
Calculation		15		
For			3	
Display				15
Calculation	30			
Calculation		20		
For			4	
Display				20
Calculation	50			
Calculation		25		
For			5	
Display				25
calculation	75			
calculation		30		
for			6	
display				30
calculation	105			
calculation		35		
For			**7**	
Display				Sum = 105

Note that in this example, every multiple of 5 is displayed in the loop and their sum is displayed only once, after the loop has terminated, as can be seen in the output column.

8.3.5 EXAMPLE 5

Enter 8 integers between 5 and 48. Display the average of these numbers.

Before we start doing the algorithm, we first have to think about it, make some notes and plan how we are going to do the logic.

Once more, we only need the average right at the end, but we need a sum before the average can be calculated. Regarding the average, we need the value of the final sum to be divided by the number of values added to the sum. This can only be done right at the end.

The numbers that must be entered must be obtained within the for-loop and added to the sum because that must also be done 8 times.

Lastly, the average can only be displayed after it has been calculated in the final stage.

Algorithm:

```
AverageOf8Integers
        sum = 0                 ~ initialize the value of sum to add the integers
        for x = 1 to 8
                display "Enter any integer between 5 and 48:" ~ on new line
                enter number
                sum = sum + number
        next x
        average = sum / 8
        display "The average of the 8 numbers is ", average
end
```

8.4 LOOPS IN VB.NET

8.4.1 FORMAT OF THE FOR-STATEMENT

Syntax:

```
For counter [As Data type] = startvalue To endvalue [Step stepvalue]
        Statement[s]
Next counter
```

Example 3 in 8.3.3 can be written in Vb.NET as follows:

```
Dim intSum As Integer = 0
Dim intOdd As Integer = 1
Dim intK As Integer
For intK = 1 to 5
     intSum = intSum + intOdd
     intOdd = intOdd + 2
Next intK
lblAnswer.Text = "The sum of the first 5 odd numbers is " _
                 & intSum
```

The variable intK can also be declared in the For-loop in which case it will have block scope.

```
Dim intSum As Integer = 0
Dim intOdd As Integer = 1
For intK As Integer = 1 to 5
     intSum = intSum + intOdd
     intOdd = intOdd + 2
Next intK
lblAnswer.Text = "The sum of the first 5 odd numbers is " _
                 & intSum
```

8.4.2 DISPLAYING OUTPUT DURING EVERY EXECUTION OF THE LOOP ON A LABEL

In VB.NET, when a value is assigned to the text property of a label, that value will be displayed on the label.

Example:

lblAnswer.Text = "First answer"
lblAnswer.Text = "Second answer"

In this case, the value displayed on the label will be **Second answer**, because it replaced the previous value assigned to the text property of the label.

If output must be displayed on a label during every execution of the loop, it means that the new value must be displayed after the previous value. It must not replace the other values that have been placed onto the label during previous executions of the loop.

In such a case, the label must have an initial value (it could be an empty string) and during every execution of the loop, the new value must be concatenated to the value(s) already placed onto the label during previous executions of the loop.

Let's illustrate this, by using the following algorithm that has been discussed in Example 4 in 8.3.4.

```
MultiplesOf5
        sum = 0                          ~ initialize the accumulator
        multiple = 5                     ~ first multiple of 5
        display "The first 6 multiples of 5 and their sum:"
        for x = 1 to 6
                display multiple         ~ on a new line
                sum = sum + multiple     ~ accumulate
                multiple = multiple + 5  ~ calculate the next multiple of 5
        next x
        display "Sum = ", sum
end
```

User interface:

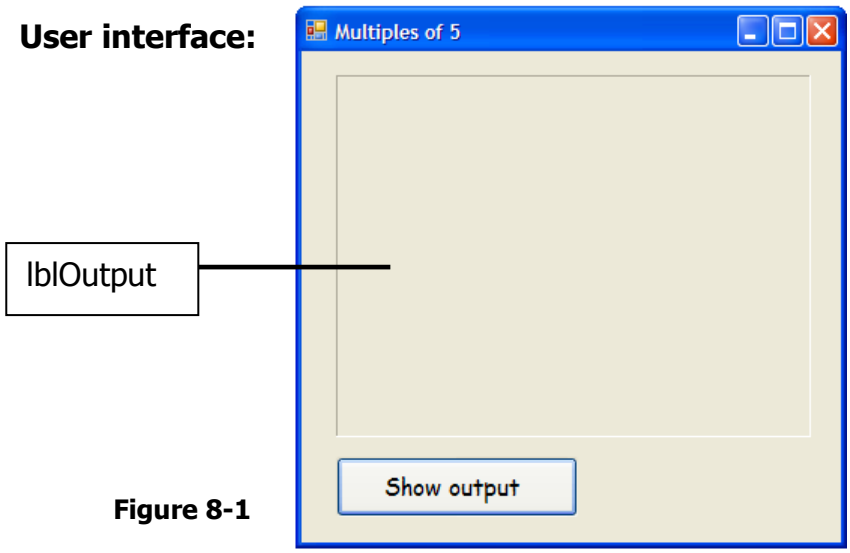


Figure 8-1

Code in code editor for the show output button click event:

```
Private Sub btnShow_Click(. . .) Handles btnShow.Click
    Dim intSum As Integer = 0
    Dim intMultiple As Integer = 5
    Dim intX As Integer

    lblOutput.Text = _
        "The first 6 multiples of 5 and their sum:" & _
        ControlChars.NewLine


    For intX = 1 To 6
        lblOutput.Text = lblOutput.Text & _
                         ControlChars.NewLine & _
                         intMultiple
        intSum = intSum + intMultiple
        intMultiple = intMultiple + 5
    Next
```

Initial value for text property of the label

Concatenate every multiple of 5 with the previous value on label

Go to a new line on the label before the multiple of 5 is displayed

```
    lblOutput.Text = lblOutput.Text & _
                     ControlChars.NewLine & _
                     ControlChars.NewLine & _
                     "Sum = " & intSum
End Sub
```

Display the sum of the multiples after the loop has been terminated. Concatenate it to the previous contents of the label after a blank line has been displayed

OUTPUT

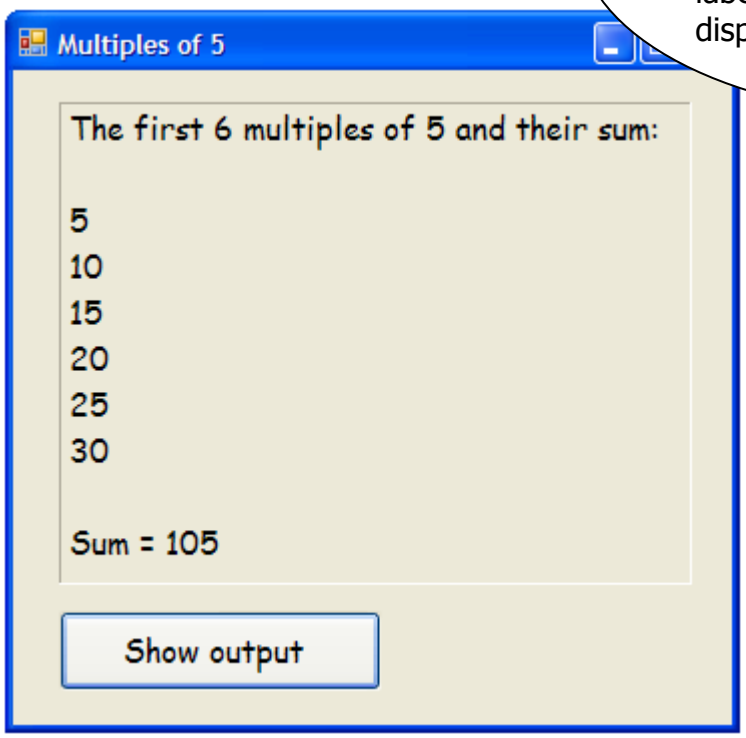


Figure 8-2

8.4.3 LIST BOXES

8.4.3.1 PURPOSE OF A LIST BOX CONTROL

In the previous example we have added every new output element to a label by concatenating it to the previous value on the label. The label had to be the right size in order to accommodate all the values that would be added to it, which is not always possible knowing in advance.

A list box is a control by which a number of output values may be displayed to the user. These values would typically be added during each execution of a loop. The advantage is that the size of the list box may be set according to the design of the user interface. If the number of elements to be displayed on the list box exceeds the size of the list box, a scroll bar will automatically be added to the list box in order to enable the user to scroll up and down to view all the items in the list box.

Another feature of a list box is that a user may select one or more of the entries in the list box. However this is normally done during array processing and for the purpose of this course, we will only use a list box to display values and therefore the SelectionMode property of our list boxes will be set to None.

A list box control can be instantiated (created) by choosing the ListBox in the toolbox window. The name of the list box must be meaningful and must start with lst according to the Hungarian naming convention.

Assume we want to design a user interface to display the first 200 even numbers in a list box. The form can be designed as indicated in **Figure 8-3**. Note that the name of the list box will appear in the design window. The size of the list box may vary according to the design and the scroll bar will only appear when items to be listed in the list box exceed the size of the list box.

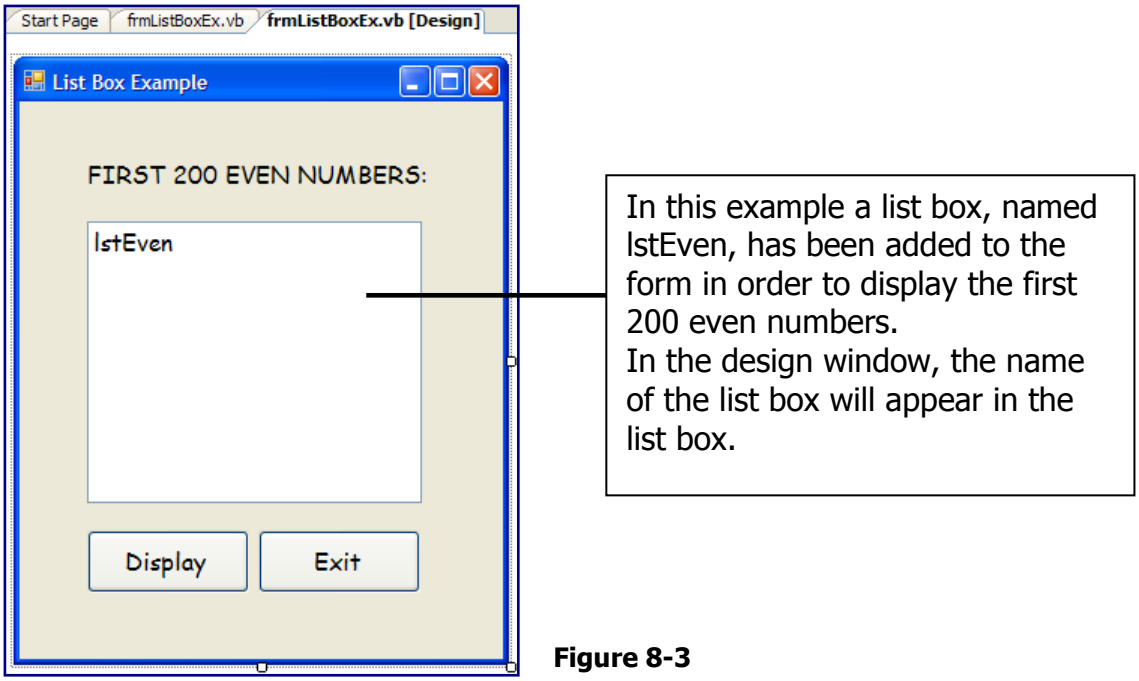


Figure 8-3

8.4.3.2 PROPERTIES COMMONLY USED FOR A LIST BOX CONTROL

The following properties of a list box can be set:

Name:	Give the list box a meaningful name starting with lst.
Font:	You may select a certain font and size for list box items.
SelectionMode:	Set to None. In such a case a list box will only be used to list items. No items may be selected.
Sorted:	Initially set to false, in which case items will be displayed in the order in which they have been added to the list box. If set to true, items will be displayed in alphabetical or numerical order.

MultiColumn	If false, every item will be displayed on a new line, otherwise items will be displayed in five columns across the list box.
Items.Count	Determine the number of items in a list box.
Items.Add	Adds a new item to the list box.

Students are advised to play around and explore with the different properties of a list box.

When SelectionMode is not none, but set to indicate that one or more items may be selected, you may also use the SelectedIndex property to get or set the index of the selected item and the SelectedItem property to get or set the value of the selected item. However, this will not be used that during this course.

8.4.3.3 GUI DESIGN GUIDELINES FOR A LIST BOX CONTROL

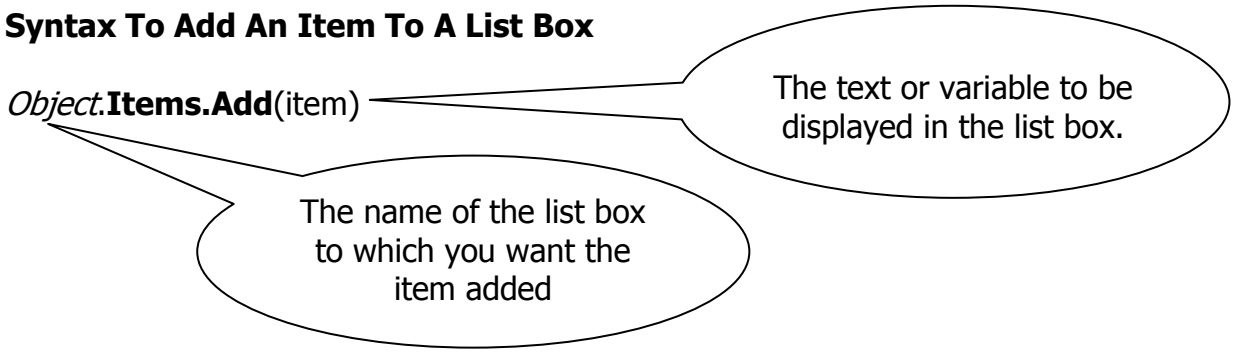
Keep the following in mind when placing a list box on the form:

- It is recommended that a list box should display a minimum of 3 and a maximum of 8 selections at a time.
- List box items are either arranged by use, where the entries most used will appear first in the list. Otherwise it is sorted in ascending or descending sequence depending on the application.

8.4.3.4 ADDING ITEMS TO A LIST BOX

The items in a list box belong to a collection called the items collection – a group of one or more individual objects treated as a unit. The first item in the item collection appears as the first item in the list box. Items can be added to the list box by using the Add method of the items collection.

Syntax To Add An Item To A List Box



Example:

lstNames.Items.Add("Sam")
The name Sam is added to the items in the list box called lstNames.

The code in the code editor for our example would be as follows if we want to display the first 200 even numbers in the list box when the Display-button is clicked.

```
Private Sub btnDisplay_Click(. . .) Handles btnDisplay.Click
   Dim intX As Integer
   Dim intEven As Integer = 2
   For intX = 1 To 200
       lstEven.Items.Add(intEven)
       intEven = intEven + 2
   Next
End Sub
```

It would result in the output indicated in **Figure 8-4** and **Figure 8-5**. In the second figure, the scroll bar has been moved to the end of the list box.

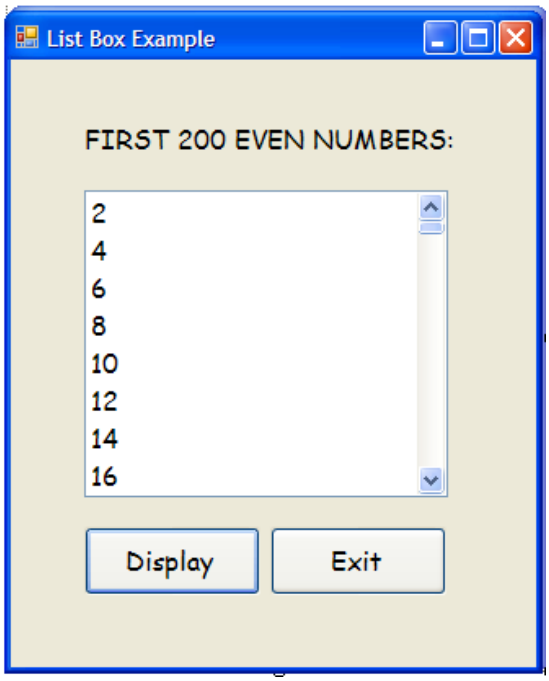


Figure 8-4

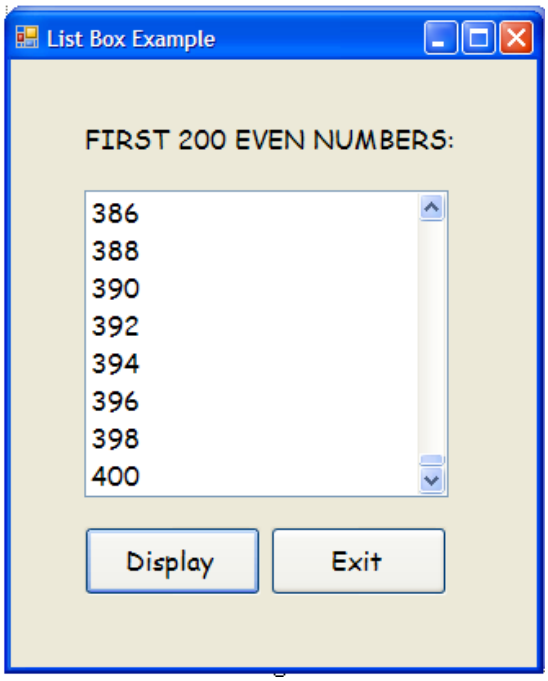


Figure 8-5

The following code would produce the same output:

```
Private Sub btnDisplay_Click(. . .) Handles btnDisplay.Click
   Dim intEven As Integer
   For intEven = 2 To 400 Step 2
       lstEven.Items.Add(intEven)
   Next
End Sub
```

If the MultiColumn property of the lstEven list box is set to true, the output will be displayed in columns as indicated in **Figure 8-6** and **Figure 8-7**. The scroll bar will then appear horizontally.

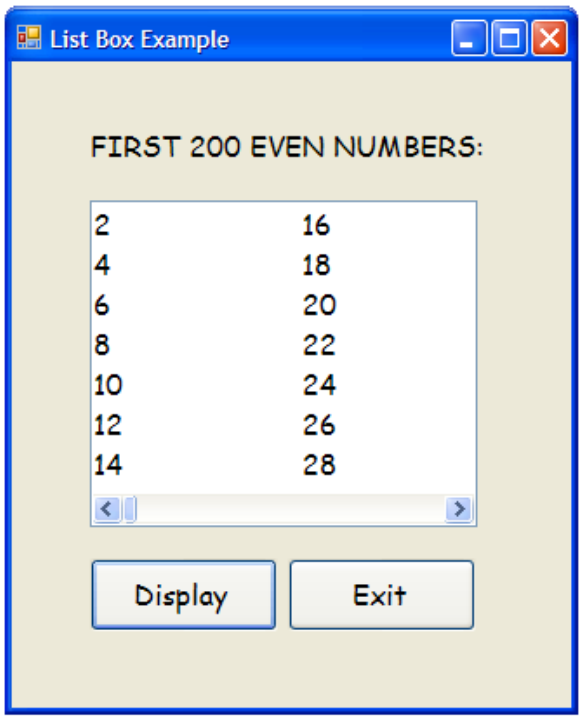


Figure 8-6

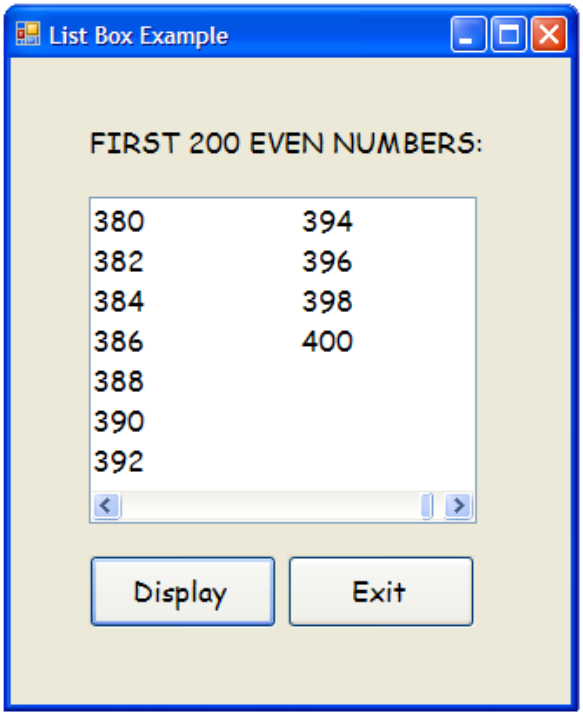


Figure 8-7

8.4.4 INPUT BOX FUNCTION

When data must be entered repetitively, it can not be entered into a text box but the input box function must be used instead to display a predefined dialog box.

The dialog box should contain a message to prompt the user to enter a value in the input area of the dialog box. The value entered must be stored in a variable that has been declared as data type **String**, regardless the data type of the value entered. If a numeric value must be entered, it must be stored in a variable declared as data type string and must then be converted to the correct data type before it can be used in calculations.

The **format** of the input box function is as follows and illustrated in **Figure 8-8**:

Variable = InputBox(prompt [, title] [, default value])

Prompt: The message to instruct the user to enter the correct data. The message must be clear and unambiguous and it must instruct the user clearly how to end the process. The message may also include a variable e.g. "Please enter amount " & intNum
The message Please enter amount 1 would be displayed if intNum contains the value 1. If intNum contains the value 2 during the next execution of the loop, the message would change to: Please enter amount 2 etc.

Title: The title that will appear as caption in the input box. It is optional.

Default value: A default value may be assigned to the value that will be entered in the input box. A default response for a numeric value could be 0.

E.g.

```
strAmount = InputBox("Please enter amount " & intNum & ",Cancel to stop", _
                     "Calculation of final Amt due", "0")
```

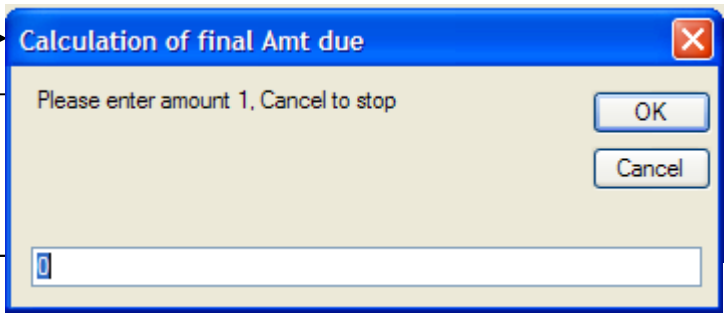


Figure 8-8

For this example, if the user enters an amount of 60.99, the value in strAmount would be "60.99". In order to use it in a calculation, it would have to be converted to decimal and to be placed in a variable that has been declared as decimal.

E.g.

```
decAmount = Convert.ToDecimal(strAmount)
```

Remember, all the validation techniques must still be applied to ensure that the value entered is valid.

E.g.

```
strAmount = InputBox("Please enter amount " & intNum _
  & ",Cancel to stop", "Calculation of final Amt due", "0")
If Decimal.TryParse(strAmount, decAmount) Then
     :
     :
```

The prompt and the title part of the input box can also be variables:

E.g.

```
Dim strPrompt, strTitle, strAmount As String
Dim intNum As Integer = 1

:
strPrompt = "Please enter amount " & intNum & _
            ",Cancel to stop"
strTitle = "Calculation of final Amt due"
:
strAmount = InputBox(strPrompt, strTitle, "0")
:
```

If the default value is omitted, no value will be indicated in the input area. However, if the title is omitted, the project name will be displayed as the title. In the following example only the prompt is provided and the name for the project is Project4 as indicated in the result in **Figure 8-9**:

```
strAmount = InputBox("Please enter amount " & intNum & _
                     ",Cancel to stop")
```

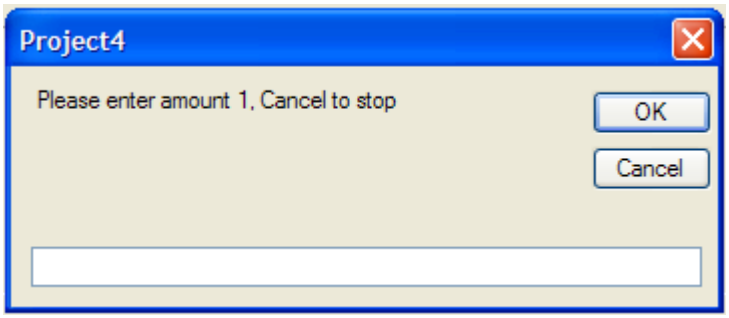


Figure 8-9

The user can enter the current input value by clicking on the OK-button or by pressing the enter-key because the OK-button is automatically the default button of an input box.

8.5 PRACTICAL EXAMPLES

8.5.1 PRACTICAL EXAMPLE 1

Do the planning and write a program to display a series of even numbers. The user is going to ask 2 questions:

- At what even number do you want to start?
- How many even numbers do you want to display?

8.5.1.1 PROGRAM PLANNING

	Description	Type	Name of variable
Input:	even number to start	integer	beginNo
	how many even numbers	integer	howMany
Output:	output even numbers	integer	evenNo

I	P	O
beginNo howMany	Prompt for input fields Enter input fields Calculate new even number Display evenNo	evenNo

Algorithm:

DisplayEvenNumbers

```
~       Calculate the desired even numbers

        display "Provide the beginning even number"          ~ display on new line
        enter beginNo
        display "How many even numbers must be displayed?"~ display on new line
        enter howMany
        if      beginNo is numeric
                if      howMany is numeric and howMany > 0
                        ~ test if beginNo is an even number
                        remainder = beginNo MOD 2
                        If      remainder = 0
                                ~       Use a loop to display the even numbers
                                even = beginNo
                                for     x = 1 to howMany
                                        display even, " "     ~ on one line
                                        even = even + 2
                                next x
                        else
                                display "The begin number must be an even number"
                        endif
                else
                        display "The number must be numeric and greater than 0"
                endif
        else
                display "The begin number must be numeric"
        endif
end
```

Test the program with the following 4 sets of test data:

beginNo = 8, howMany = 10	beginNo = 9, howMany = 15
beginNo = 16, howMany = -3	beginNo = x, howMany = 3

Output:

8 10 12 14 16 18 20 22 24 26

The begin number must be an even number

The number must be numeric and greater than 0

The begin number must be numeric



Questions

What will the output be if the user enters -16 for a begin value and 9 for howMany?

What will the output be if the user enters 0 for a begin value and 4 for howMany?



Exercises

The for-loop in the algorithm above can be changed as follows:

```
for x = beginNo to ((beginNo * 2) + howMany) step 2
        display x
next x
```

Do a trace table to prove that this last for loop will yield the correct result.

8.5.1.2 IMPLEMENTATION IN VB.NET

The user interface has been designed to display all the even numbers on a label. The two text boxes are named txtBeginNo and txtHowMany and the label to display the answer is lblDisplay. The button to activate the above event is btnDisplay.

CORRESPONDING CODE IN THE CODE EDITOR

```
Private Sub btnDisplay_Click(. . .) Handles btnDisplay.Click
   Dim intBeginNo, intHowMany, intRem As Integer
   Dim intEven, intX As Integer
   lblDisplay.Text = ""

   If Integer.TryParse(txtBeginNo.Text, intBeginNo) Then
      If Integer.TryParse(txtHowMany.Text, intHowMany) Then
         If intHowMany > 0 Then

            intRem = intBeginNo Mod 2
            If intRem = 0 Then
               intEven = intBeginNo

            For intX = 1 To intHowMany
               lblDisplay.Text = lblDisplay.Text & _
                                 intEven & " "
               intEven = intEven + 2
            Next
         Else
            MessageBox.Show(_
            "The begin number must be an even number",_
            "Error", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)
            txtBeginNo.Text = ""
            txtBeginNo.Focus()
         End If
      Else
         MessageBox.Show(_
         "The second number must be numeric", "Error",_
         MessageBoxButtons.OK, MessageBoxIcon.Information)
         txtHowMany.Text = ""
         txtHowMany.Focus()
      End If
   Else
      MessageBox.Show("The second number must be > 0",_
      "Error", MessageBoxButtons.OK, _
      MessageBoxIcon.Information)
      txtHowMany.Text = ""
      txtHowMany.Focus()
   End If
 End If
End Sub
```

Data Validation: Test if the two numbers entered are integers and if the second number is greater than zero

Test if the first number is an even number

Execute the loop only if all input values are valid. Display every new even value on the label by concatenating it to the previous even values already on the label

EXAMPLES OF OUTPUT:

Three examples of possible output are provided in **Figures 8-10 to 8-12**. **Figure 8-11** provides the answer for one of the previous questions on page 204. We are sure you answered it correctly. Yes, -16 is a valid even number as the program specification did not state that only positive even numbers must be displayed. 0 is also treated as an even number.

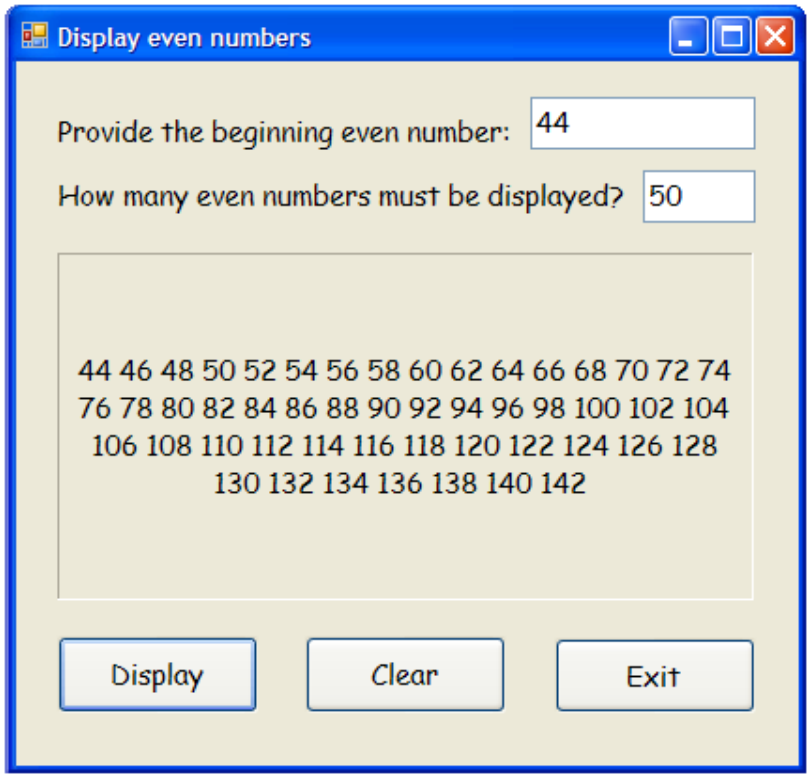


Figure 8-10

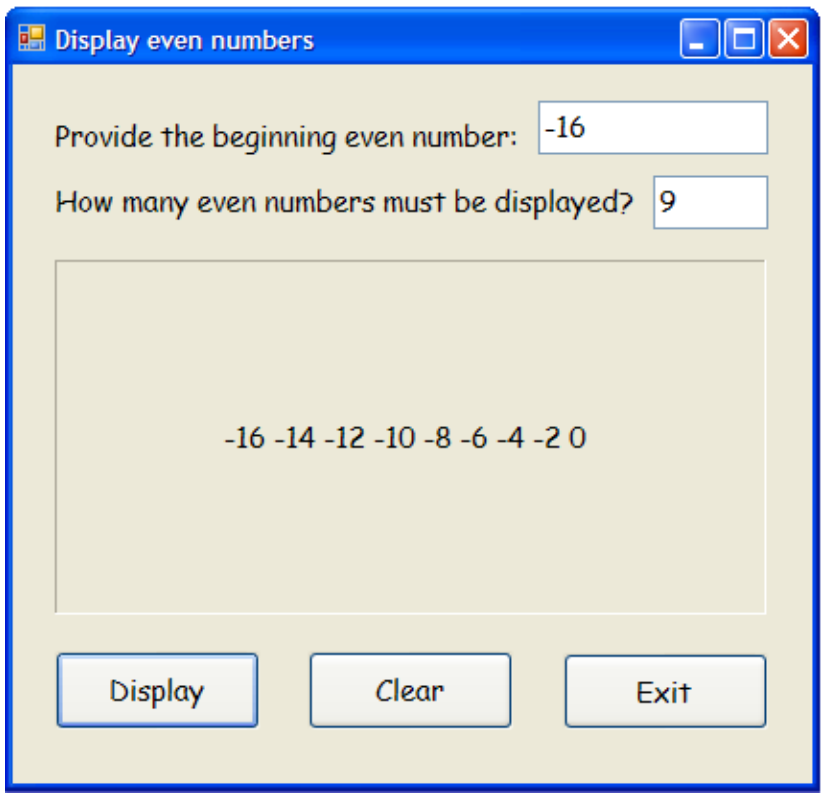


Figure 8-11

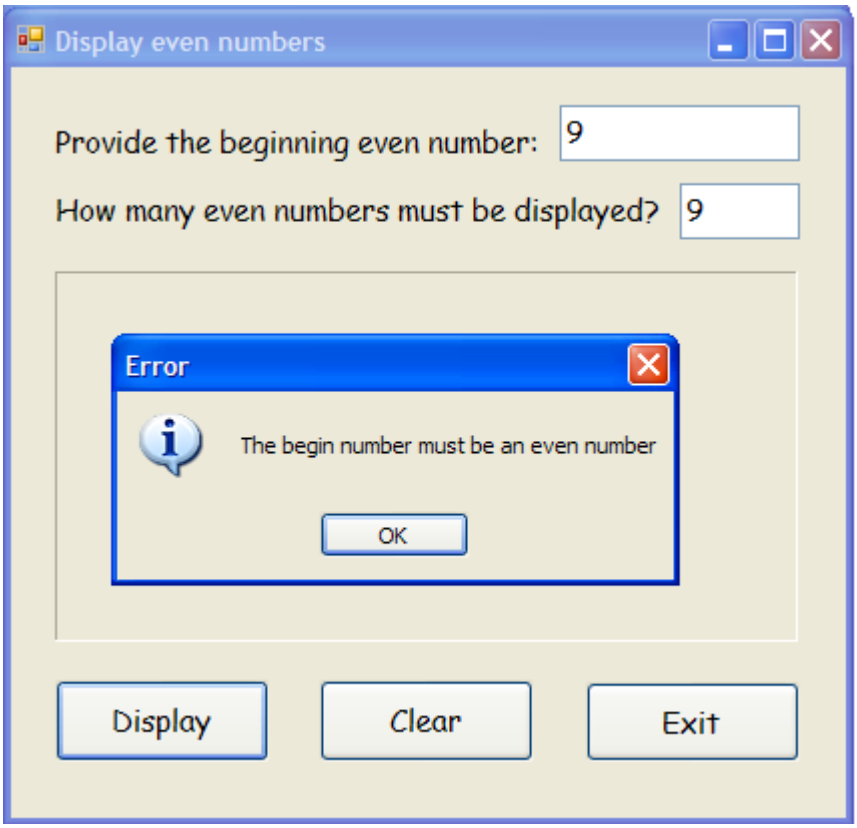


Figure 8-12

8.5.2 PRACTICAL EXAMPLE 2

The 10 students in the Information Systems class of Brilliant College wrote a test. The principle of the college wants to know what the highest mark is and who obtained it. He also requires the name of the student who obtained the lowest mark and the mark of this student. The user must enter the names and test marks of the students. Display the answers. The marks are percentages given as integers.

8.5.2.1 PROGRAM PLANNING

When planning this program, we need to clarify a few aspects. At first we need to have a value to compare the current mark to in order to determine which mark is lower or higher than the other one. So, we are going to declare a variable called *lowest* and another variable called *highest* that will contain the lowest and the highest test marks respectively.

There are 2 different ways of doing this:

a.

- Assign a very low value to highest i.e. –1 so that all the remaining values compared to this number will be higher.
- Assign a very high value to lowest i.e. 101 so that all the remaining values compare to this number will be less.
- Repeat the process 10 times.

or
b.

- Assign the mark of the first student to lowest as well as to highest in order to compare the remaining values to it.
- Repeat the process only 9 times, because the first value has already been dealt with.

We are going to use the second method in our program. *For this program demonstration we are going to assume that all input values will be valid.*

	Description	Type	Name of variable
Input:	name of student	string	stName
	test percentage of student	integer	testMark
Output:	name of best student	string	highestName
	highest percentage	integer	highest
	name of lowest student	string	lowestName
	lowest percentage	integer	lowest

I	P	O
stName testMark	Prompt for input fields Enter input fields Compare to find results Display output fields	highestName highest lowestName lowest

Algorithm:

TestResults
~ Find highest and lowest marks of test results

~ Enter the name and percentage for the first student

display "Enter the name of the first student" ~ display on new line
enter stName
display "Enter test percentage of the first student" ~ display on new line
enter testMark

~ assign values to highest and lowest
~ names belonging to the test marks must also be stored

highestName = stName
highest = testMark
lowestName = stName
lowest = testMark

~ Execute a loop. Repeat 9 times (from student 2 to student 10)
~ For each student compare his or her marks to the current highest
~ and current lowest mark.

```
    for st = 2 to 10
        ~       Enter every student name and mark.
        ~       Indicate number of student in message

        display "Enter the name of student no ", st      ~ display on new line
        enter stName
        display "Enter test percentage of student no ", st
                                                         ~ display on new line
        enter testMark

        ~       Compare to highest and lowest mark. If necessary, place new
        ~       name and mark in variables that keep track of highest and
        ~       lowest scores

        if      highest < testMark then
                highest = testMark          ~assign higher value to highest
                highestName = stName        ~ remember to store name as well
        else
                if lowest > testMark
                        lowest = testMark   ~assign lower value to lowest
                        lowestName = stName      ~ store name as well
                endif
        endif
    next st

    ~       The results can only be displayed after all the names and their test
    ~       marks have been entered and compared.  The final answers are
    ~       now ready to be displayed. Display on clear screen.

    display "The name of the student who obtained the highest mark is  ",
                    highestName                          ~ display on new line
    display "The highest mark obtained is ", highest     ~ display on new line
    display "The name of the student who obtained the lowest mark is  ",
                    lowestName                           ~ display on new line
    display "The lowest mark obtained is ", lowest       ~ display on new line
end
```

Test the program using the following test data:

Name:	Danny	Test percentage:	50
	Bill		67
	Don		92
	Dave		28
	Sonny		62
	Edith		54
	Bob		34
	Robbie		43
	Casandra		64
	Julie		78

Output:

```
The name of the student who obtained the highest mark is Don
The highest mark obtained is 92
The name of the student who obtained the lowest mark is Dave
The lowest mark obtained is 28
```

8.5.2.2 IMPLEMENTATION IN VB.NET

The following user interface has been designed for this problem. It contains only a label (lblAnswer), to display the output after the highest and lowest mark (and respective student names who obtained it), have been determined in the loop.

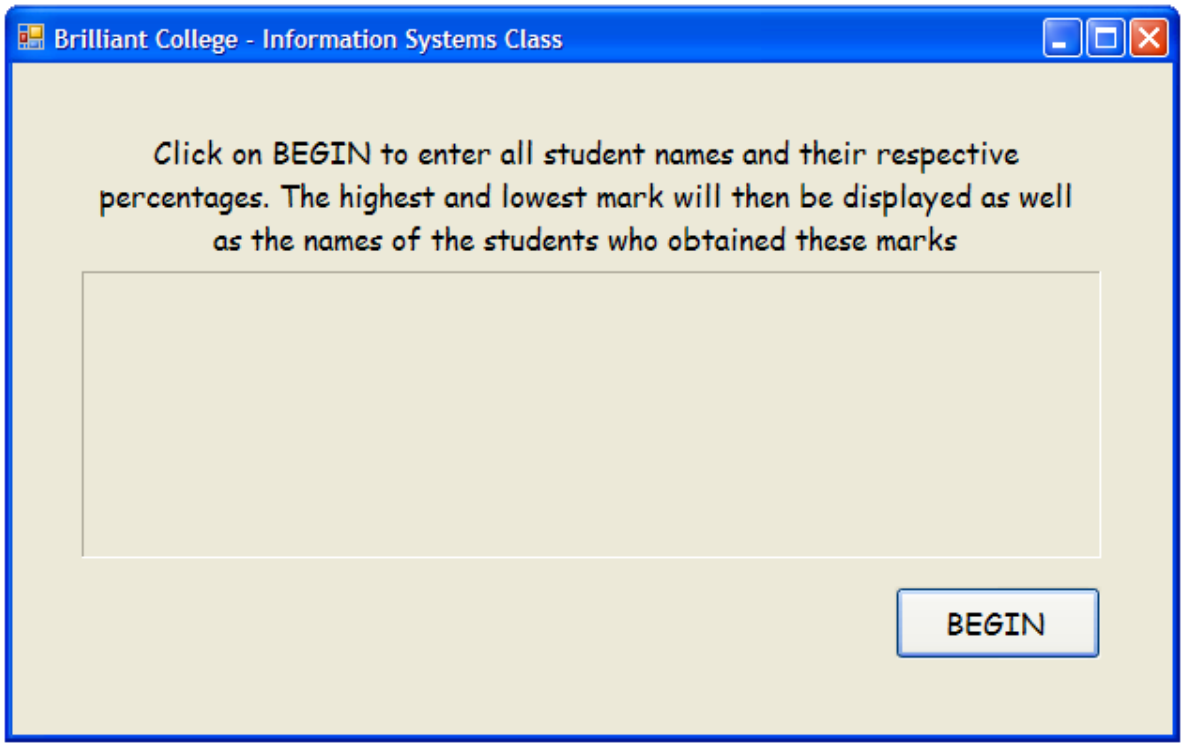


Figure 8-13

CODE IN THE CODE EDITOR

```
Private Sub btnBegin_Click(. . .) Handles btnBegin.Click
   Dim strStName, strHighestName, strLowestName, strTestMark As String
   Dim intTestMark, intHighest, intLowest As Integer
   strStName = InputBox("Enter the name for the first student", _
                        "Information Systems Test")
   strTestMark = InputBox("Enter the test mark for the first student",_
                          "Enter as an integer percentage")
   intTestMark = Convert.ToInt32(strTestMark)
   strHighestName = strStName
   intHighest = intTestMark
   strLowestName = strStName
   intLowest = intTestMark
```

```
    For intSt As Integer = 2 To 10
        strStName = InputBox("Enter the name for the student no " & _
                             intSt, "Information Systems Test")
        strTestMark = InputBox("Enter the test mark for student no " _
                               & intSt, "Enter as an integer percentage")
        intTestMark = Convert.ToInt32(strTestMark)
        If intTestMark > intHighest Then
           intHighest = intTestMark
           strHighestName = strStName
        Else
           If intTestMark < intLowest Then
              intLowest = intTestMark
              strLowestName = strStName
           End If
        End If
    Next
    lblAnswer.Text =
         "The name of the student who obtained the highest mark is " _
         & strHighestName & ControlChars.NewLine & _
         "The highest mark obtained is " & intHighest _
         & ControlChars.NewLine & _
         "The name of the student who obtained the lowest mark is " & _
         strLowestName & ControlChars.NewLine & _
         "The lowest mark obtained is " & intLowest
End Sub
```

OUTPUT

When the user clicks on BEGIN, the input boxes will be displayed and the user must enter the student name and test percentage for every student. As from student 2, the number of the student will appear as part of the message in the message box. The first 4 input boxes displayed are shown in **Figure 8-14**.

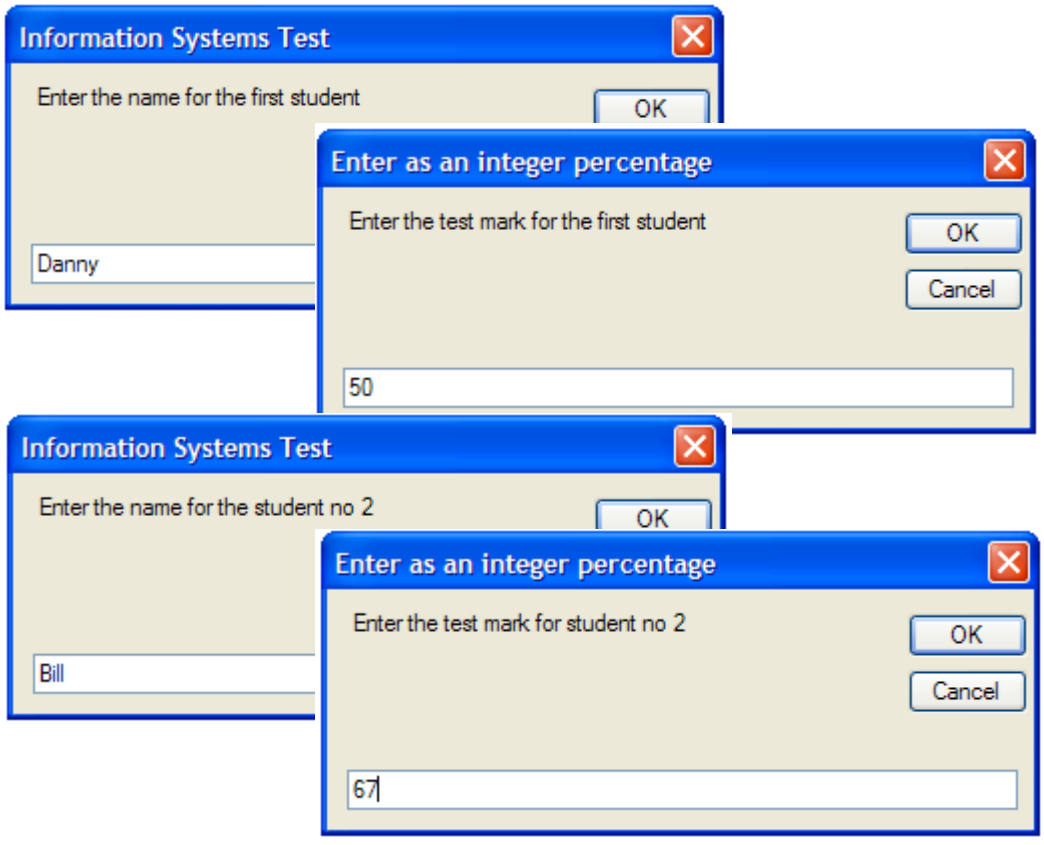


Figure 8-14

When the last test mark has been entered, the loop will automatically be terminated and the following output will be displayed on the label **(Figure 8-15).**

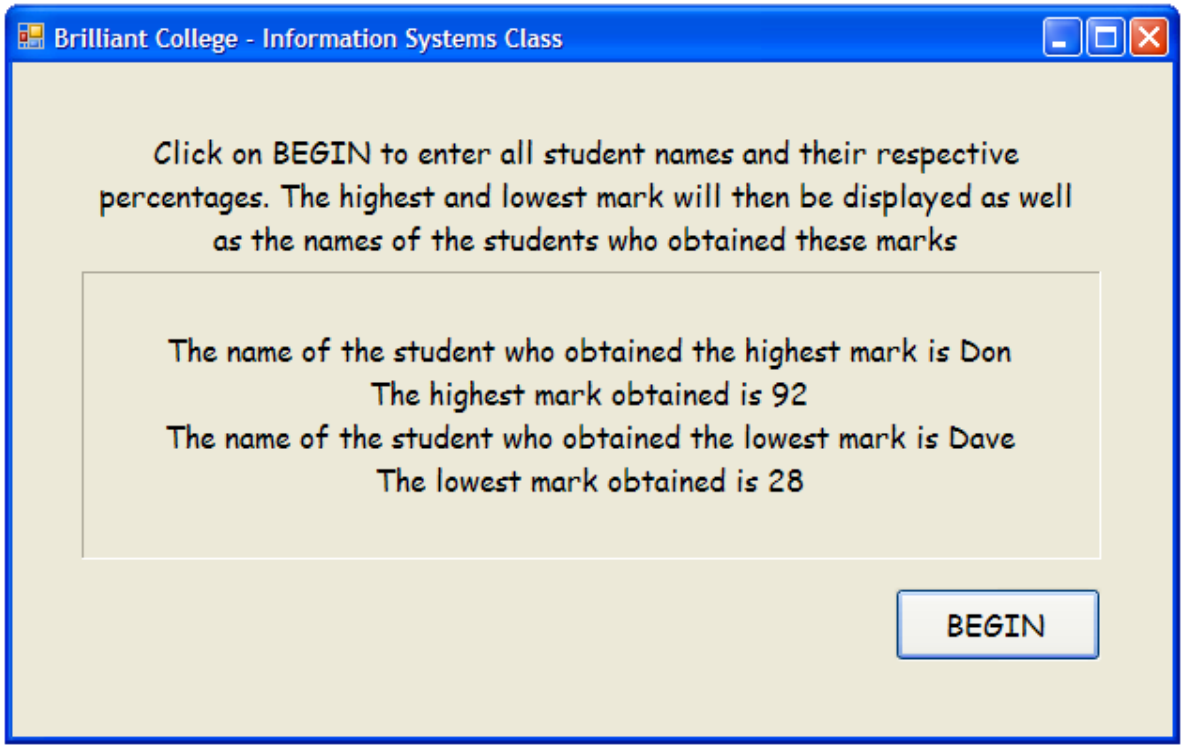


Figure 8-15

8.5.3 PRACTICAL EXAMPLE 3

In this program we are going to do division by subtracting values using a for-loop.

The user is asked to enter an integer total number that is greater than 400. Calculate what the result will be if the total number is divided by 5 without using division. The answer must be an integer.

8.5.3.1 PROGRAM PLANNING

	Description	Type	Name of variable
Input:	total number	integer	totNumber
Output:	result	integer	result

I	P	O
totNumber	Prompt for totNumber Enter totNumber Subtract 5 from totNumber until finished Display result	result

Algorithm:

```
DivisionProgram
~       Divide  by using subtraction
        ~ Initialize the result
        result = 0

        display "Enter the total number to be divided by 5"     ~ display on new line
        enter totNumber

        if      totNumber is Numeric then
                if totNumber > 5 then
                ~       Repeat subtracting until totNumber < 5
                        for y = totNumber to 0 step –5
                                totNumber = totNumber – 5
                                result = result + 1
                                if      totNumber < 5 then
                                        y = 0           ~  too small for another subtraction
                                endif
                        next y
                        display "The number can be divided by five ", result, " times"
                else
                        display "The number is less than 5, it cannot be divided by 5"
                endif
        else
                display "The number entered must be a positive integer"
        endif
end
```

Test the program using 543, 3 and x as the total number

Output:

```
The number can be divided by five 108 times

The number is less than 5, it cannot be divided by 5

The number entered must be a positive integer
```

8.5.3.2 IMPLEMENTATION IN VB.NET

The corresponding code in the code editor and output for the above mentioned test data is now provided.

CODE IN THE CODE EDITOR

```
Private Sub btnCalculate_Click(. . .) Handles btnCalculate.Click
   Dim intTotNumber As Integer
   Dim intResult As Integer = 0
   If Integer.TryParse(txtTotNumber.Text, intTotNumber) Then
      If intTotNumber > 5 Then
         For intY As Integer = intTotNumber To 0 Step -5
             intTotNumber = intTotNumber - 5
             intResult = intResult + 1
             If intTotNumber < 5 Then
                intY = 0
             End If
         Next intY
         lblAnswer.Text = "The number can be divided by five " & _
                          intResult & " times"
      Else
         lblAnswer.Text = _
         "The number is less than 5, it cannot be divided by 5 "
      End If
   Else
      lblAnswer.Text = "The number entered must be a positive integer"
   End If
End Sub
```

OUTPUT

Three examples of output are provided in **Figure 8-16** to **Figure 8-18.**

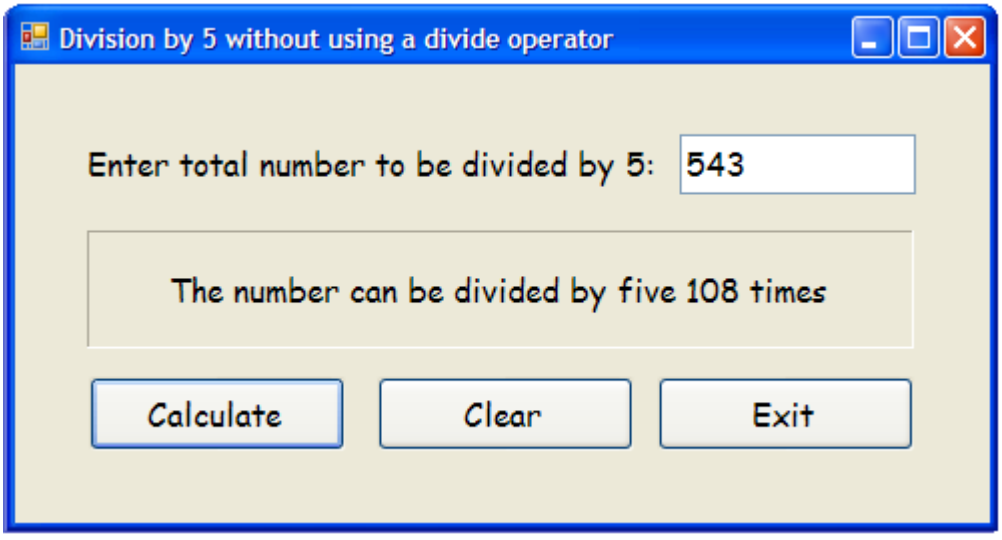


Figure 8-16

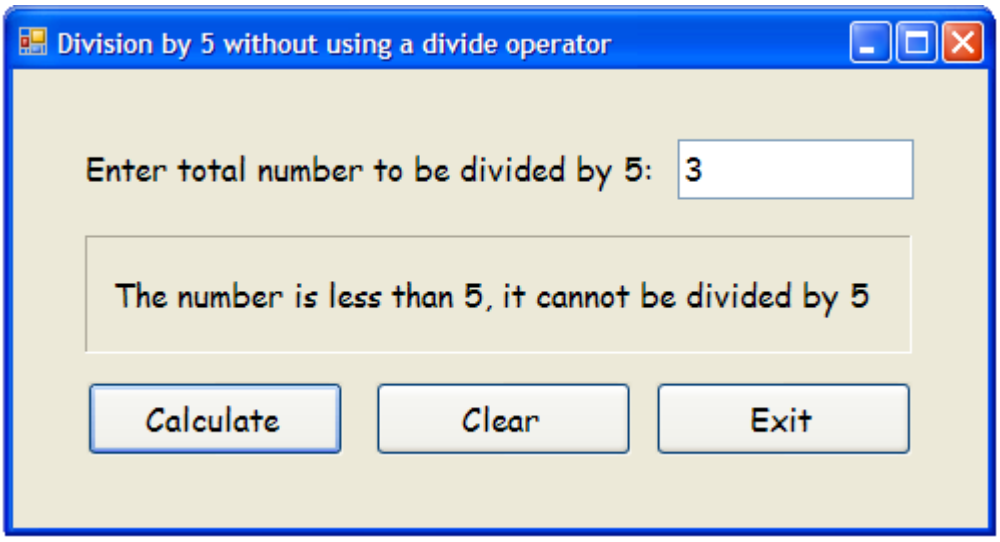


Figure 8-17

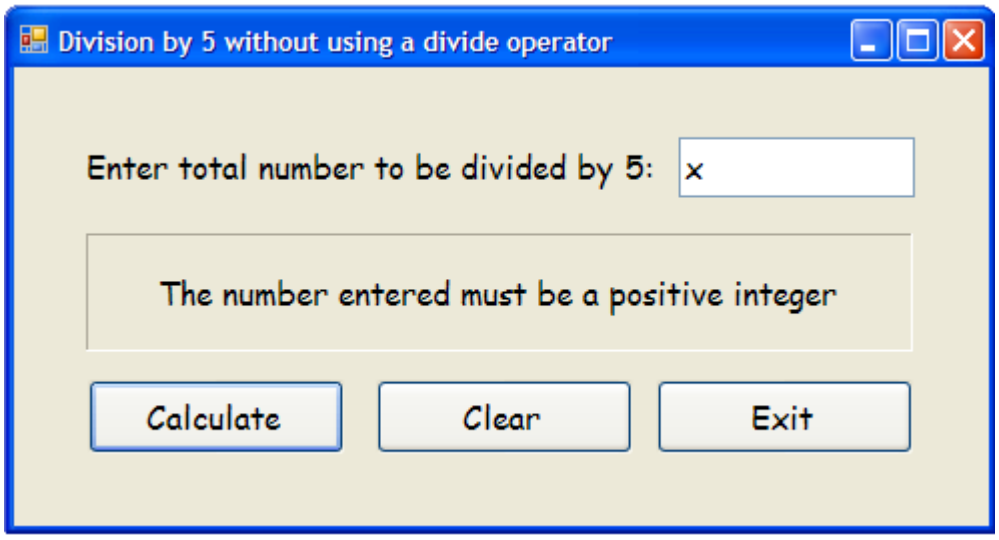


Figure 8-18



Do the necessary planning and write a solution in VB.NET for each of the following problems. Each solution should contain a For-loop.

1. You have been asked to display a multiplication table.

   The user has to enter which table (any integer from 1 to 12) he/she wants and then the program must produce the table.

   Provide for incorrect input and test your program with various tables. If the user entered a 5 the table should look like the example in **Figure 8-19**:

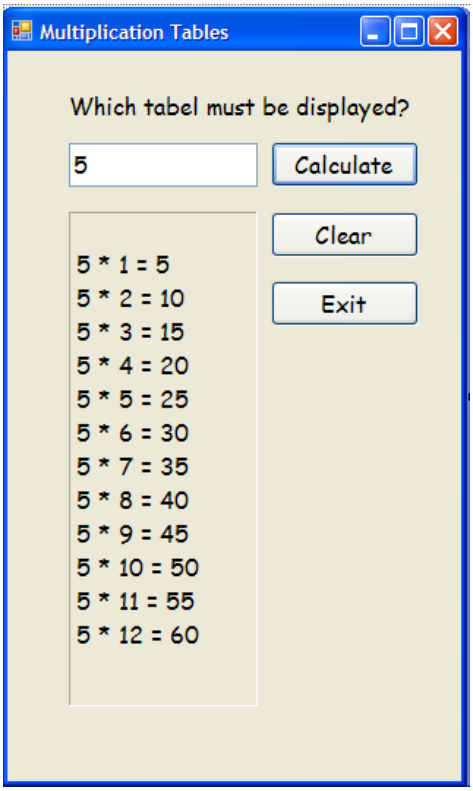


Figure 8-19

2. Rewrite example 4 in 8.3.4 (applied in VB.NET in 8.4.2) to display **any number** of multiples of five as indicated in **Figure 8-20**.

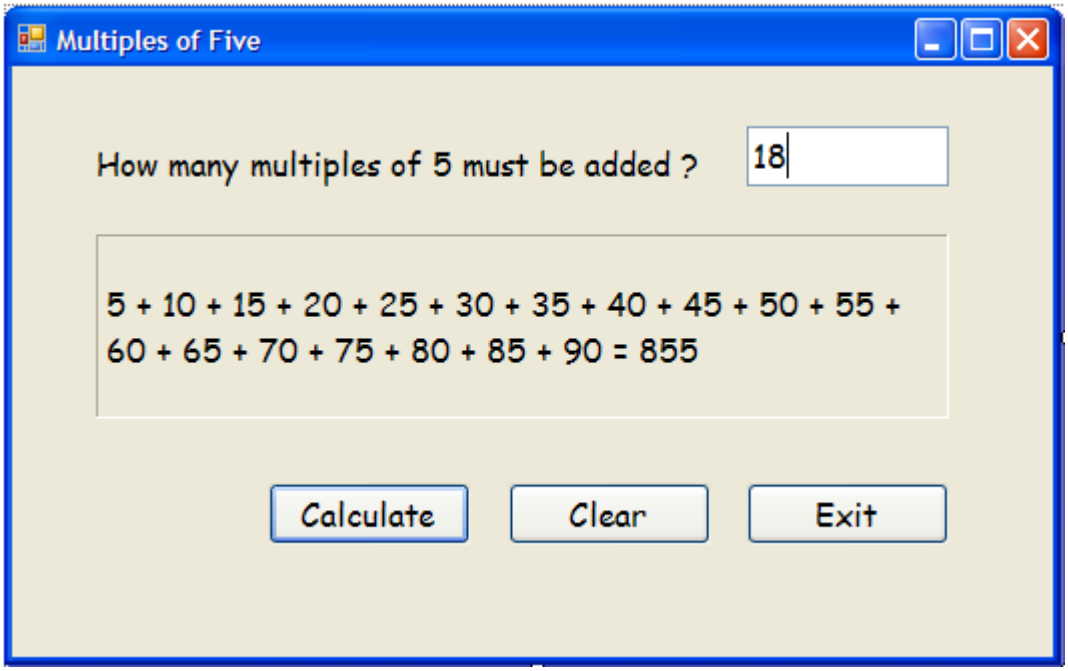


Figure 8-20

3. Rewrite practical example 2 in 8.5.2 to provide for incorrect input. Therefore, if the user enters an invalid test mark for the third student, he or she must be able to correct it before continuing to the next iteration of the loop. The program must also ensure that every student name is entered.

4. The distribution manager of a newspaper uses 25 boys to assist her to deliver the newspapers early in the morning. She has to know what is the average number of newspapers delivered daily and also how many boys deliver more than the average, how many deliver less than the average and how many deliver the number equal to the average. Use a for-next-loop to enter the number delivered by each boy. Display all the totals calculated. (hint: calculate the average using one for-next-loop and then use another for-next-loop to determine the totals).

5. The Direct Postal Service sends parcels to customers world wide. A customer often orders a number of items to be packed in different parcels. Every parcel is marked with the name of the customer, the weight of the parcel and the number of the parcel e.g. if 5 parcels are sent to the same customer, the second parcel will be numbered 2 of 5. The user is asked to enter the name of the customer and the number of parcels (maximum 20 parcels). This number is used to control the for-next-loop. The weight of every parcel (real number) must also be entered before the details for every parcel must be displayed as follows:

   Name of customer
   Weight of parcel in kg.
   Parcel x of y

   You have to write this program for one customer only.

6. Keep it Kleen is a company that sells vacuum cleaners. The company uses representatives to sell their stock. Every representative has a goal provided by the company. If the representative sells more vacuum cleaners than the goal, he receives R56.20 as a bonus per additional vacuum cleaner sold. If he has not met the goal, he has to pay R15.75 for every vacuum cleaner sold less than the goal. Enter the name, goal and number of vacuum cleaners sold and then calculate the bonus or amount that must be paid to the company. Display the name and amount for every representative. Repeat this procedure for 15 representatives.

For practical assignments 4 to 6, design a suitable, user friendly interface and code as effective as possible.

CHAPTER 9
ITERATION USING THE DO-LOOP

9.1 INTRODUCTION

In the previous chapter the for-loop was discussed where the number of times some statements had to be repeated, was known. However, the exact number of times that a loop must repeat is often not known. Therefore it will be necessary to study other types of loop-structures.

OUTCOMES

When you have studied this chapter you should be able to

- understand the difference between a pretest loop and a posttest loop,
- write a do while-loop in pseudocode and in VB.NET,
- write a do-loop-until-statement in pseudocode and in VB.NET,
- know what a sentinel is and how to use it to terminate a do-loop,
- know the different ways in which a do-loop can also be terminated in VB.NET,
- write algorithms and corresponding VB.NET solutions containing
  - do-while-loops,
  - do-loop-until-statements,
  - combinations of all structures previously covered, e.g. an if-statements within a loop etc.

9.2 THE DO-LOOP

To illustrate the concept of a loop where we do not know the exact number of times the statements must be repeated, we can imagine a long line of people wanting to buy tickets for a football match. If there are 10 people in the queue, we might say that the loop will be repeated 10 times, however, some more people may join the line and it is not clear how many tickets will be sold. Certain conditions will stop this process e.g.

- All the available tickets have been sold
- The box office closed at 17:00
- All the people who were in the line have already bought their tickets
- The box office never opened

If all these conditions were included in a program, it is clear that we really have no idea when the repetition must stop. The program needs a statement to test these conditions otherwise it will run forever (endless loop).

There are two types of Do-loops. When planning a solution that contains a number of statements to be repeated several times, the programmer has to decide which type of loop must be used.

9.2.1 PRETEST LOOP (DO-WHILE-STATEMENT)

The first one is called the pretest loop where the condition is tested before the statements within the loops are processed. Looking at the box office selling the football tickets, the following conditions may occur: Nobody may stand in the line to buy tickets, all the tickets were sold out yesterday or the lady who sells the tickets did not open the box office. In this case no tickets will be sold.

An example of this type of loop is:

```
Do while time <= 1700
        Statement-1
        Statement-2
        :
        Statement-n
loop
```

The statements in the body of the do-loop will be processed while the condition is true. As soon as it becomes false, the next statement after the end of the loop will be processed. If it is false the first time when it is tested, it might happen that the statements in the loop will not even be executed once.

9.2.2 POSTTEST LOOP (DO-LOOP-UNTIL-STATEMENT)

The next type of do-loop is called the posttest loop. The statements in the body of the loop will be processed at least once. The condition will be tested only after the statements in the body of the loop have been executed. In this case we may consider an example where only 100 tickets are available but there are 150 people waiting to buy tickets. So, tickets will only be sold until 100 tickets have been sold.

An example of this type of loop is:

```
Do
        Statement-1
        Statement-2
        :
        Statement-n
loop until noTickets >= 100
```

The statements in the do-loop will repeatedly be processed until the condition at the end of the loop becomes true. It will then proceed to the next statement following the do-loop structure.

9.2.3 TERMINATING EXECUTION OF A LOOP

There must be a statement in the body of the loop for both the pretest loop and the posttest loop that enables the outcome of the condition tested in the loop control statement to change. If not present, the loop will never end.

For example, when sales must repeatedly be entered to determine the highest sales, the programmer may decide to enter a sales amount of -1 to end the loop. Any suitable value may have been chosen to indicate the end of the processing, as long as this value is not a possible valid input to the program. The value is called a **sentinel**.

In VB.NET, a programmer can use a sentinel to end the loop, or make use of the functions provided in the Input Box Function to end the loop (or even make use of both).

The input box function provides an easy way to terminate a loop if the user clicks on the Cancel-button or clicks on the Stop-icon.

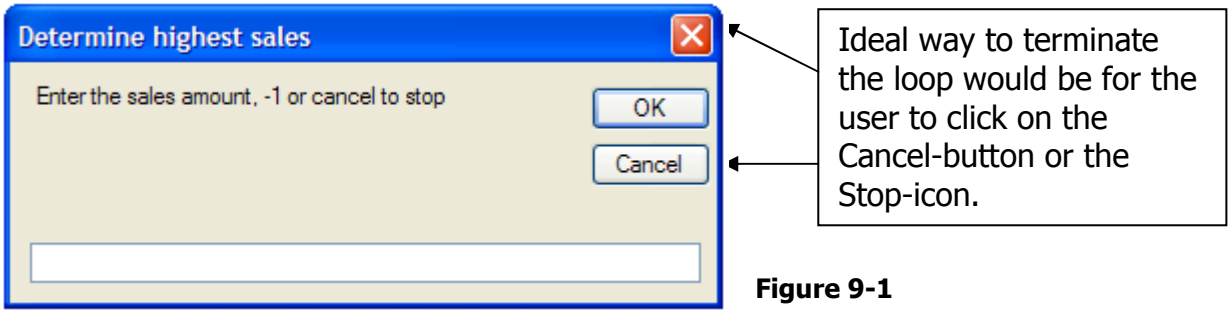


Figure 9-1

When the user clicks on either the Cancel-button, or on the Stop-icon, the input box will return an **empty string**. The programmer can therefore process the loop while the variable is not equal to an empty string.

For our previous example, the code could be as follows:

```
strAmount = InputBox_
     (“Enter the sales amount, -1 or cancel to stop“, _
      “Determine highest sales”)
Do While strAmount <> “” AndAlso strAmount <> “-1”
     :

     :
Loop
```

9.3 EXAMPLES OF THE DO-WHILE-LOOP

As discussed before, this type of loop tests a condition before processing the statements in a loop. It is therefore possible that the body of the do while loop will never execute.

9.3.1 EXAMPLE 1

Calculate the sum of all the consecutive integers starting at 24 while the sum is less than 23456. Display how many integers have been added to the sum.

CalcSum

```
~       Accumulate a counter of integers added to a sum
        sum = 0         ~ initialize sum                          } see
        number = 24                                               } comment 1
        count = 0       ~ initialize the counter
        do while sum < 23456
                sum = sum + number      ~ accumulate the sum
                number = number + 1     ~ proceed to the next consecutive no
                count = count + 1       ~ increment count
        loop

        display "The number of integers added is ", count       ~ see comment 2
end
```

At this stage it is necessary to understand which statements must be done before processing the loop, what statements must be within the loop and which statements must be placed after the loop to yield the correct results.

Comment 1: These statements prepare variables before going into the loop. They initialize the variables with the correct starting values.

In the do while control statement the sum is tested to be less than 23456. In the body of the while-loop, the statement *sum = sum + number* will increase sum. As *number* is a positive integer, sum will increase until it eventually reaches 23456 or more. At this stage the do while-loop will terminate. The count is also incremented to indicate that the body of the loop was processed once more.

Comment 2: It is only possible to print a final count when the condition in the do while control statement is no longer satisfied.

9.3.2 PRACTICAL EXAMPLE 1

We are now going to do a complete planning and write the program.

9.3.2.1 PROGRAM PLANNING

The fishing society had a competition to determine who is the best fisherman. The competition is won by the person who caught the most fish in the given time. For every fisherman, enter the name of the person and the number of fish caught. After all the data are entered a number of fish equal to -1 (sentinel) will be entered. Display the name of the winner and the number of fish caught.

	Description	Type	Name of variable
Input:	name of fisherman	string	fmName
	number of fish caught	integer	noFish
Output:	name of winner	string	winner

I	P	O
fmName noFish	Prompt for input values Enter fmName and noFish Determine winner Display winner, noFish	Winner noFish

Before we start writing the algorithm we have to think about the planning of the loop. The control statement in the do while-loop has a condition and reading the problem statement closely, it is clear that the number of fish is tested in this condition. Once the loop is entered the condition must already contain a value to be tested. The programmer must now realize that the number of fish caught by the first person must be entered before the loop. But there is no necessity to enter the name of the fisherman before the loop is entered.

This implies that the number of fish caught by the next person must be entered at the end of the body of the loop to be tested when returning to the do while-loop control statement.

Let us proceed with the algorithm:

```
FindFishermanWinner
~       Determine who caught the most fish!
        winnerNumber = 0          ~  choose low number for winner
        winner = "     "
        display "Provide the number of fish caught by the first fisherman"
                                  ~ display on new line
        display "Enter –1 to indicate no more input"
                                  ~ display on new line
        enter  noFish
```

```
    do while noFish <> -1       ~ test if loop must continue
        display "Provide name of fisherman"      ~ display on new line
        enter fmName
        if noFish is numeric
            if noFish > winnerNumber then
                winnerNumber = noFish
                winner = fmName
            endif
        else
            display "The number of fish caught must be numeric"
        endif
        display "Provide the number of fish caught by the fisherman"
                                ~ display on new line
        display "Enter –1 to indicate no more input"     ~ display on new line
        enter  noFish
    loop
    display "The name of the winner is ", winner          ~ display on new line
    display winner, " caught ", winnerNumber, " fish"     ~ display on new line
    display "Congratulations ", winner, "!!!"             ~ display on new line
end
```

Test the program using the following test data:

Name:	Sam	number of fish:	8
	Johnny		2
	Kevin		7
	Fred		10
	Bill		5
	Ted		12
	Paul		9

Output:

```
Provide the number of fish caught by the fisherman
Enter –1 to indicate no more input 8
Provide name of fisherman Sam
Provide the number of fish caught by the fisherman
Enter –1 to indicate no more input 2
Provide name of fisherman Johnny
:
:
Provide the number of fish caught by the fisherman
Enter –1 to indicate no more input 9
Provide name of fisherman Paul
Provide the number of fish caught by the first fisherman
Enter –1 to indicate no more input -1

The name of the winner is Ted
Ted caught 12 fish
Congratulations Ted!!!
```

9.3.2.2 IMPLEMENTION IN VB.NET

USER INTERFACE

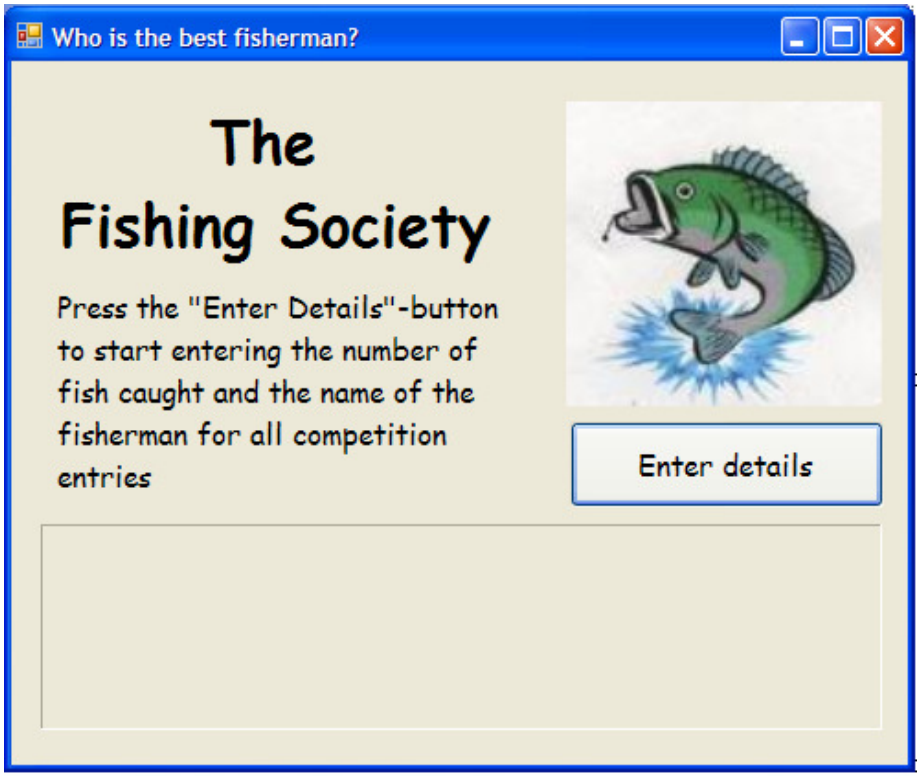


Figure 9-2

CODE IN THE CODE EDITOR

```
Private Sub btnEnter_Click(. . .) Handles btnEnter.Click

   Dim intWinnerNumber As Integer = 0
   Dim strWinner, strFmName As String
   Dim strNumFish As String
   Dim intNumFish As Integer

   strNumFish = InputBox _
    ("Provide the number of fish caught by first fisherman,_
     enter -1 or press cancel to stop", "Fishing competition")

   Do While strNumFish <> "" And strNumFish <> "-1"
      If Integer.TryParse(strNumFish, intNumFish) Then
         strFmName = InputBox("Provide name of fisherman", _
                              "Fishing competition")
         If intNumFish > intWinnerNumber Then
            intWinnerNumber = intNumFish
            strWinner = strFmName
         End If
```

User can terminate the loop by clicking on cancel or stop (in which case strNumFish will be an empty string, or the user can enter -1 for the number of fish caught.

```
strNumFish = InputBox_
           ("Provide the number of fish caught by the next
            fisherman, enter -1 or press cancel to stop", _
           "Fishing competition")
     Else
        MessageBox.Show_
           ("The number of fish caught must be numeric", "Error",_
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        strNumFish = InputBox("Re-enter the number of fish caught
           by this fisherman, enter -1 or press cancel to stop", _
           "Fishing competition")
     End If
  Loop
  lblWinner.Text = "The name of the winner is " & strWinner & _
       ControlChars.NewLine & strWinner & " caught " & _
       intWinnerNumber & " fish" & ControlChars.NewLine & _
       "Congratulations " & strWinner & "!!!"
End Sub
```

1

2

Take note:
If one statement is too long for one line, a line continuation character (_) must be used to indicate that the statement continues on a new line. However, a literal value (enclosed in quotes) may not be split in the middle of the value. Although our messages, indicated at point (1) and (2) are on one line in the code editor, it was impossible to fit them in one line in A4 format. Please don't see this as a syntax error.

EXAMPLES OF OUTPUT

When the "Enter details"-button is clicked, the following input boxes (as indicated in **Figure 9-3 to Figure 9-8**) will appear. The user must keep on providing the number of fish caught as well as the name of the respective fisherman for all fishermen who entered the competition. These values correspond to the data that was used for desk checking of the algorithm. Take note of **Figure 9-6** where an error message occurs and the user must rectify the mistake. When the details for the last fisherman have been entered, the user must enter -1 for the number of fish caught (or cancel or stop must be clicked). The process will then be terminated and the output will be displayed on the label as indicated in **Figure 9-9**.

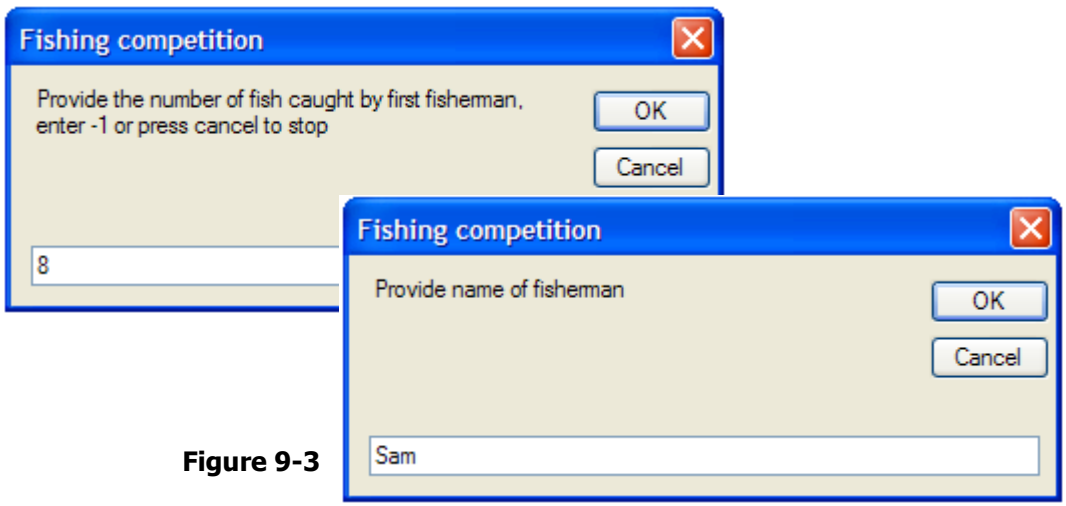


Figure 9-3

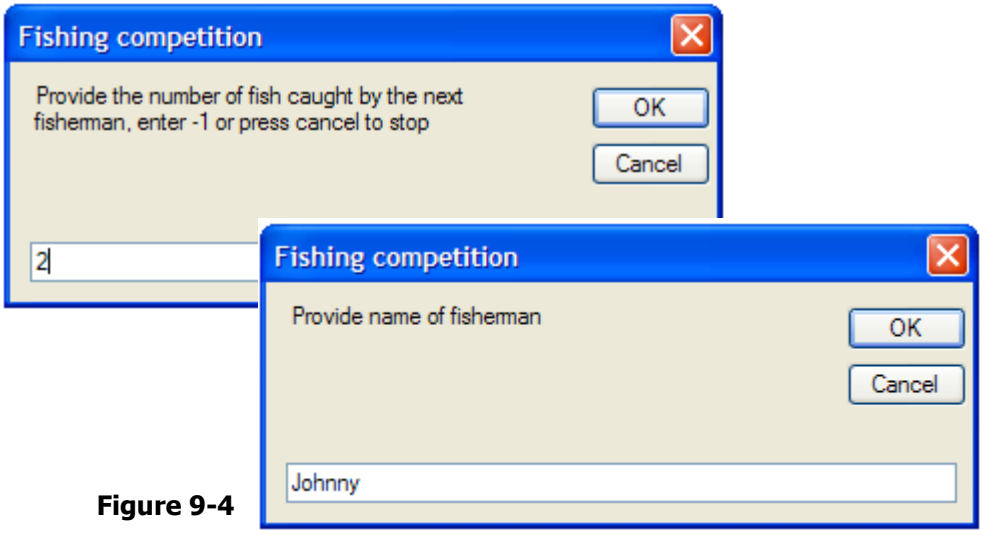


Figure 9-4

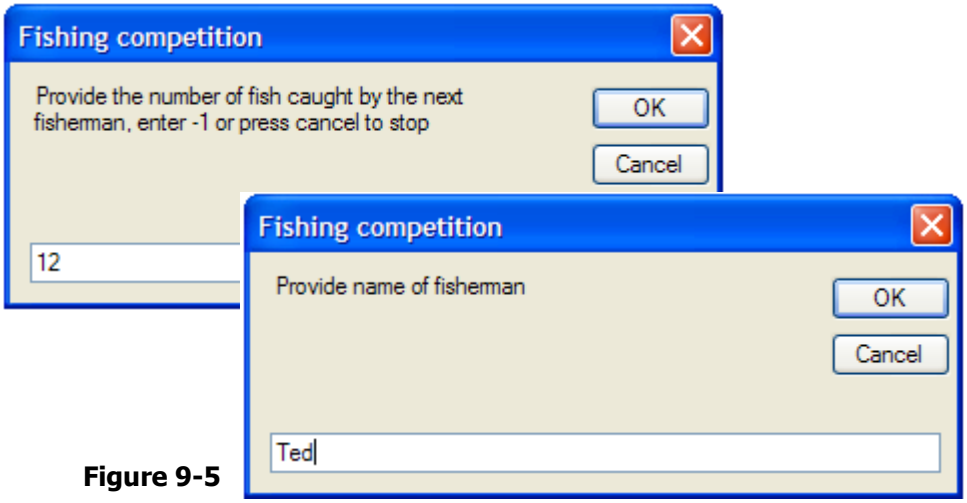


Figure 9-5

⋮

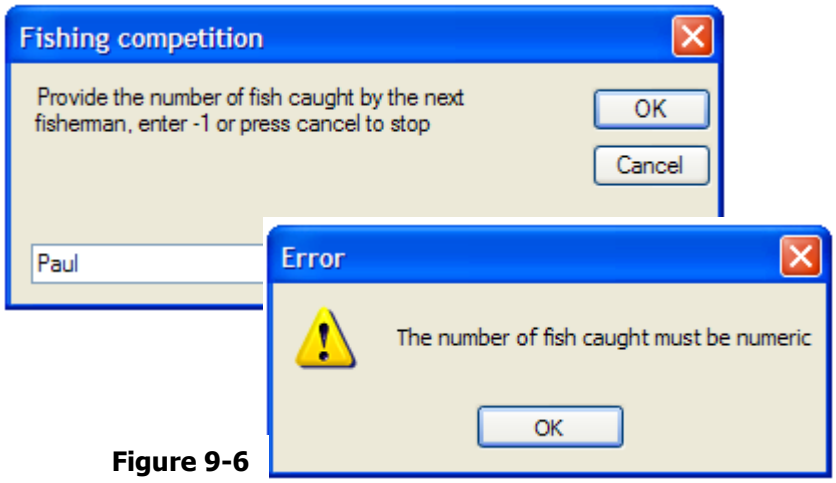


Figure 9-6

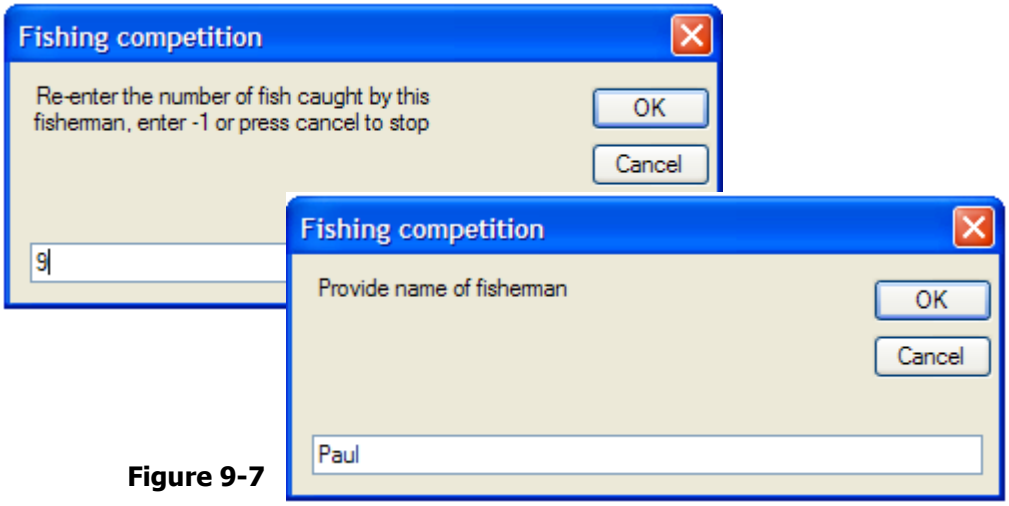


Figure 9-7

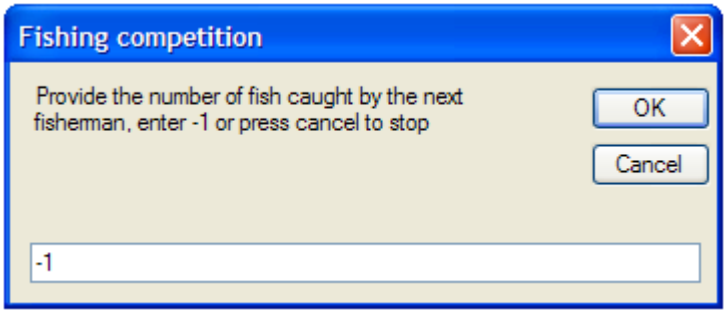


Figure 9-8



Figure 9-9

9.3.3 PRACTICAL EXAMPLE 2

9.3.3.1 PROGRAM PLANNING

Alexis went shopping and bought a number of different items. Enter the amount of money in her purse and the price of every item. Calculate the total amount. After the prices of all the items are entered, a price of 0 (zero) is entered to indicate that she has finished picking items to buy. If the total amount she spent is more than R100 she will receive a discount of 3.5%. If the money in her purse is enough to pay for her shopping, calculate and display how much money she will have left in her purse after she received any change. If her money is not enough, display a message to indicate how much more money she will need to pay for her shopping.

	Description	Type	Name of variable
Input:	price of item	real	price
	money in purse	real	purseMoney
Output:	money left in purse	real	change
	money still needed	real	shortMoney

I	P	O
price purseMoney	Prompt for input values Enter price and purseMoney Calculate totAmount Test purseMoney Display results	change shortMoney

Algorithm:

```
~  AlexisShopping
        total = 0
        ~ enter money in her purse once
        display "Enter the amount of money in Alexis' purse"  ~ display on new line
        enter purseMoney
        if purseMoney is numeric
                ~ enter prices for items to buy in a loop if money in purse is valid
                display "Enter the price of the first item"          ~ display on new line
                enter price
                do while price <> 0
                        if price is numeric
                                total = total + price
                        else
                                display "The price must be numeric"
                        endif
                        display "Enter the price of the next item, enter 0 to stop"
                        enter price
                loop
```

```
            ~ the total amount has now been calculated,
            ~ determine discount, if any
            if total > 100 then
                  total = total – total * 0.035
                  ~  total = total * 0.965 is equivalent to the statement above
            endif

            ~  clear the screen
            if purseMoney >= total then
                  change = purseMoney – total
                  display "Alexis has enough money, she now has R", change,
                        " in her purse"                    ~ display on new line
            else
                  shortMoney = total – purseMoney
                  display "Alexis needs R", shortMoney,
                        " more to pay for her purchases"  ~display on new line
            endif
      else
            display "The money in her purse must be a numeric amount"
      endif
end
```

Test the program using the following two sets of test data:

The amount in her purse is R709.55
Item 1 R150.50
Item 2 R285.70
Item 3 R397.42

Desk checking:

The total amount she spent is R833.62
This amount is more than R100, she receives 3.5% discount.
The amount due is now R833.62 – R29.18 = R804.44
She needs R804.44 – R709.55 = R94.89

The amount in her purse is R75.00
Item 1 R3.00
Item 2 R25.50
Item 3 R21.60
Item 4 R7.80

Desk checking:

The total amount she spent is R57.90
This amount is not more than R100, she receives no discount.
The amount in her purse is now R75.00 – 57.90 = R17.10

Output:

```
Alexis needs R94.89 more to pay for her purchases
Alexis has enough money, she now has R17.10 in her purse
```

9.3.3.2 IMPLEMENTATION IN VB.NET

USER INTERFACE

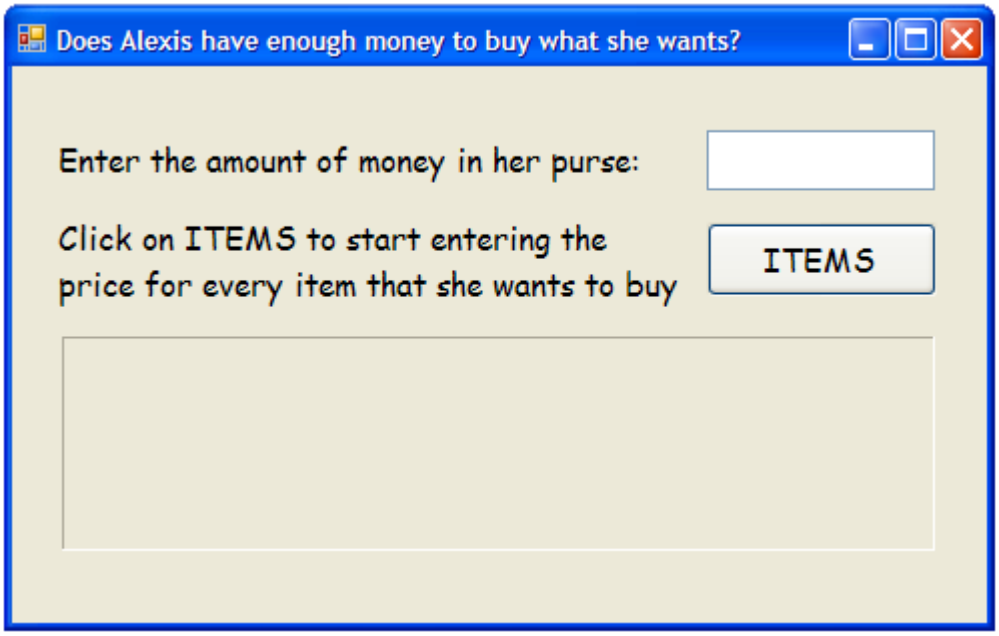


Figure 9-10

CODE IN THE CODE EDITOR

```
Private Sub btnItems_Click(. . .) Handles btnItems.Click
   Dim decTotal As Decimal = 0
   Dim decPrice As Decimal
   Dim strPrice As String
   Dim decPurseMoney As Decimal
   Dim decShortMoney As Decimal
   Dim decChange As Decimal
   If Not String.IsNullOrEmpty(txtPurseMoney.Text) Then
      If Decimal.TryParse(txtPurseMoney.Text, decPurseMoney) Then
         strPrice = InputBox("Enter price for first item", _
                             "Enter 0 or press cancel to stop")
         Do While strPrice <> "0" And strPrice <> ""
            If Decimal.TryParse(strPrice, decPrice) Then
               decTotal = decTotal + decPrice
            Else
               MessageBox.Show(strPrice & _
                " is not a valid amount", "Please re-enter", _
                MessageBoxButtons.OK, MessageBoxIcon.Information)
            End If
            strPrice = InputBox_
               ("Please enter the price for the next item", _
                "Enter 0 or press cancel to stop")
         Loop
```

```
            If decTotal > 100 Then
               decTotal = decTotal - decTotal * 0.035
            End If
            If decPurseMoney >= decTotal Then
               decChange = decPurseMoney - decTotal
               lblAnswer.Text = _
                    "Alexis has enough money to buy all items" & _
                   ControlChars.NewLine & "She now has R" & _
                   decChange.ToString("N2") & " change in her purse"
            Else
               decShortMoney = decTotal - decPurseMoney
               lblAnswer.Text = "Alexis cannot buy all these items" _
                    & ControlChars.NewLine & "She needs R" & _
                   decShortMoney.ToString("N2") & _
                   " more to pay for all her purchases"
            End If
         Else
            MessageBox.Show_
               ("The money in her purse must be a numeric amount",_
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            txtPurseMoney.Focus()
         End If
      Else
        MessageBox.Show(_
        "Enter the amount in her purse before you click on ITEMS",_
        "Retry", MessageBoxButtons.OK, MessageBoxIcon.Information)
        txtPurseMoney.Focus()
      End If
End Sub
```

OUTPUT

Figure 9-11 gives an example of an error message if the ITEMS-button was clicked before the amount in her purse was entered.

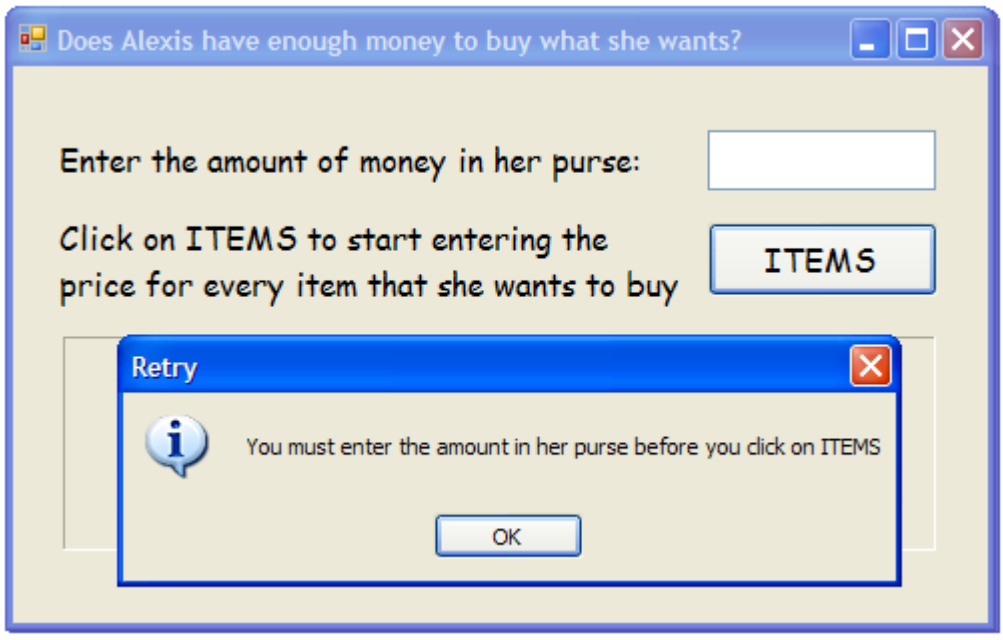


Figure 9-11

Figure 9-12 and **Figure 9-13** illustrate the example where Alexis has R709.55 in her purse and wants to buy 3 items that are more than this amount. After the third amount has been entered, the user clicked on cancel. The output result in **Figure 9-14** will then be displayed.

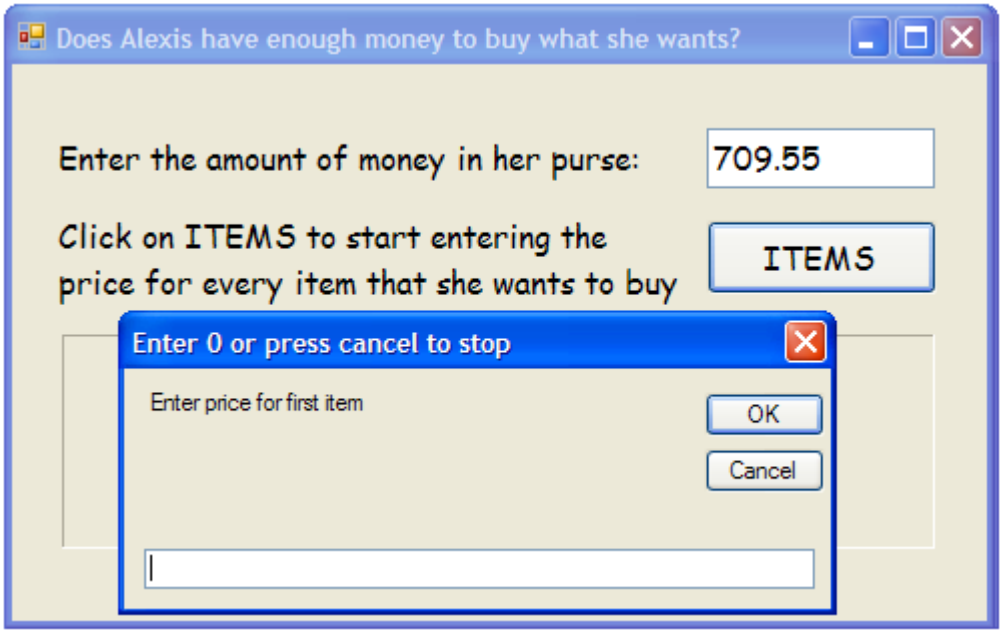


Figure 9-12

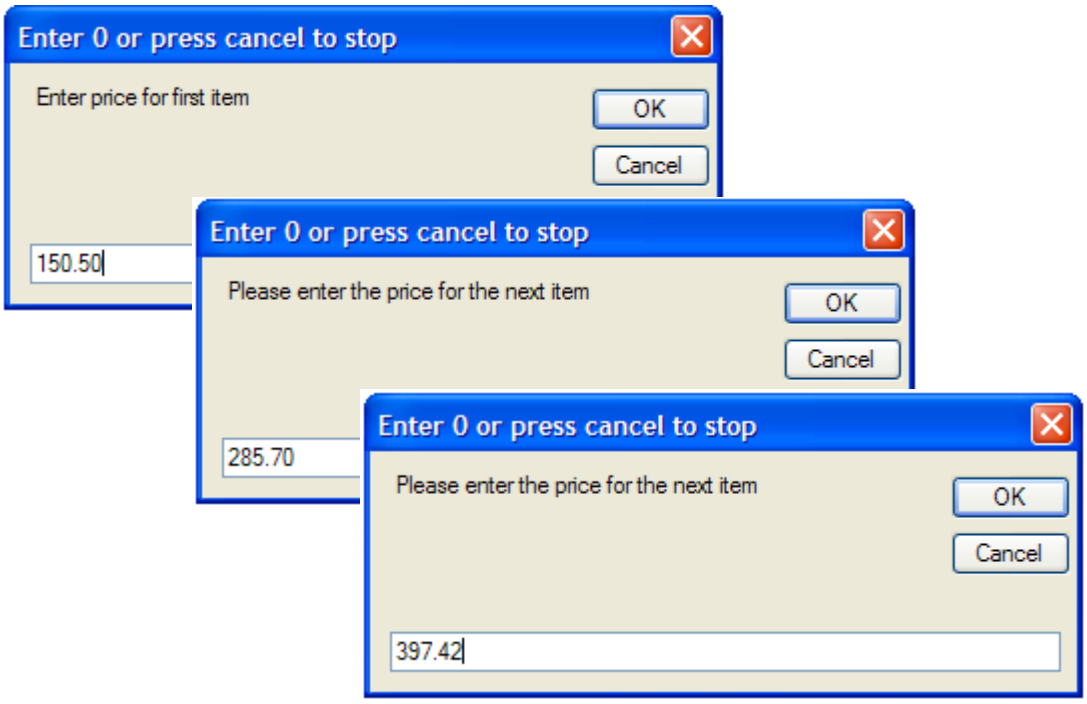


Figure 9-13

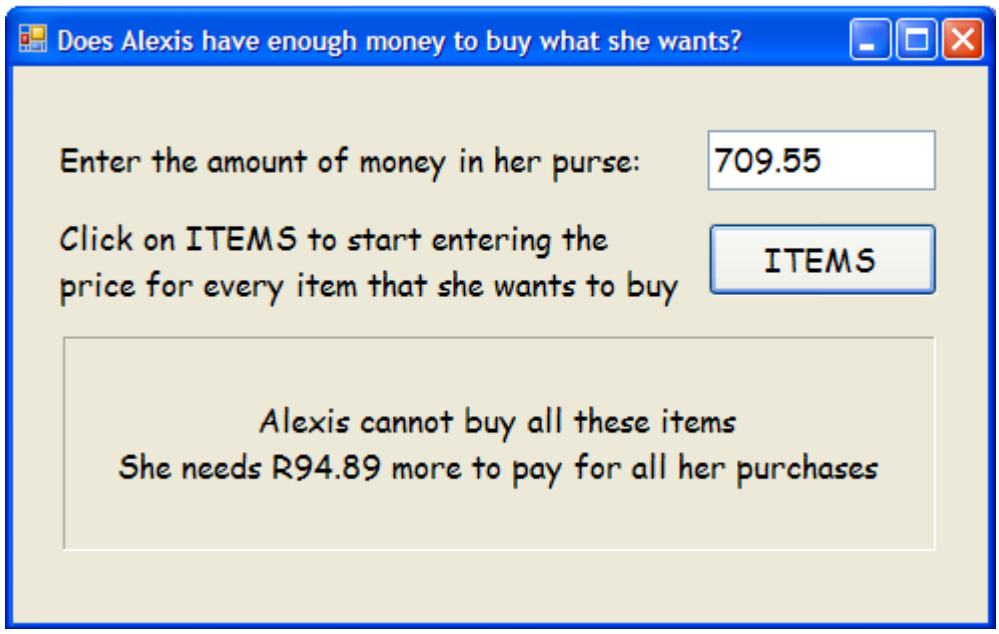


Figure 9-14

Figure 9-15 to **Figure 9-18** illustrate the example where Alexis has R75.00 in her purse and wants to buy items of R3, R25.50, R21.60 and R7.80. She has enough money to pay and will still have R19.13 change in her purse as the output in **Figure 9-19** indicates after the user has entered a 0 to end the loop. Take note, in **Figure 9-17** the user has entered a wrong amount, in which case a suitable error message was displayed.

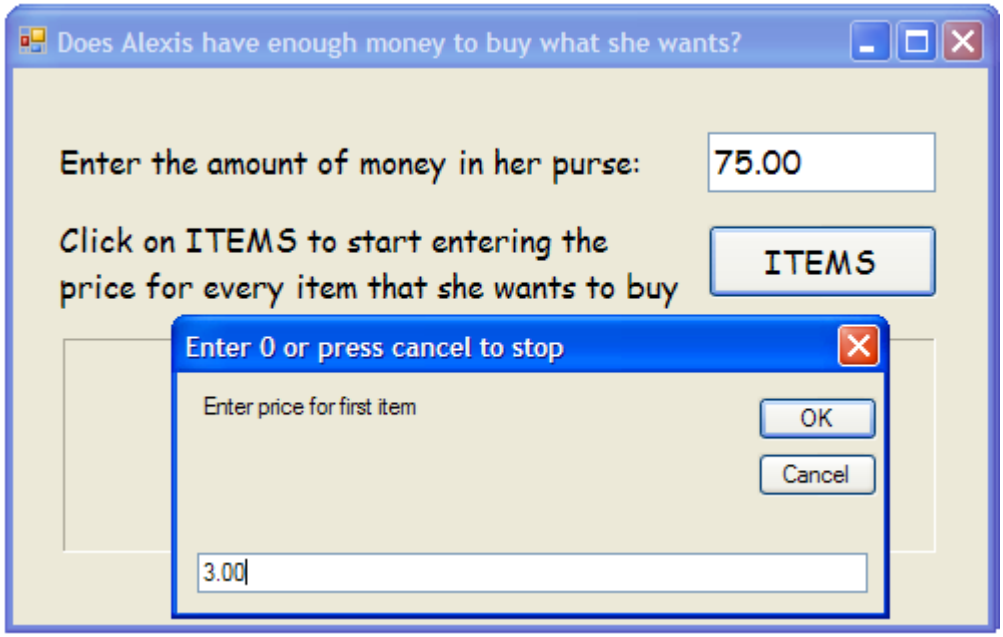


Figure 9-15

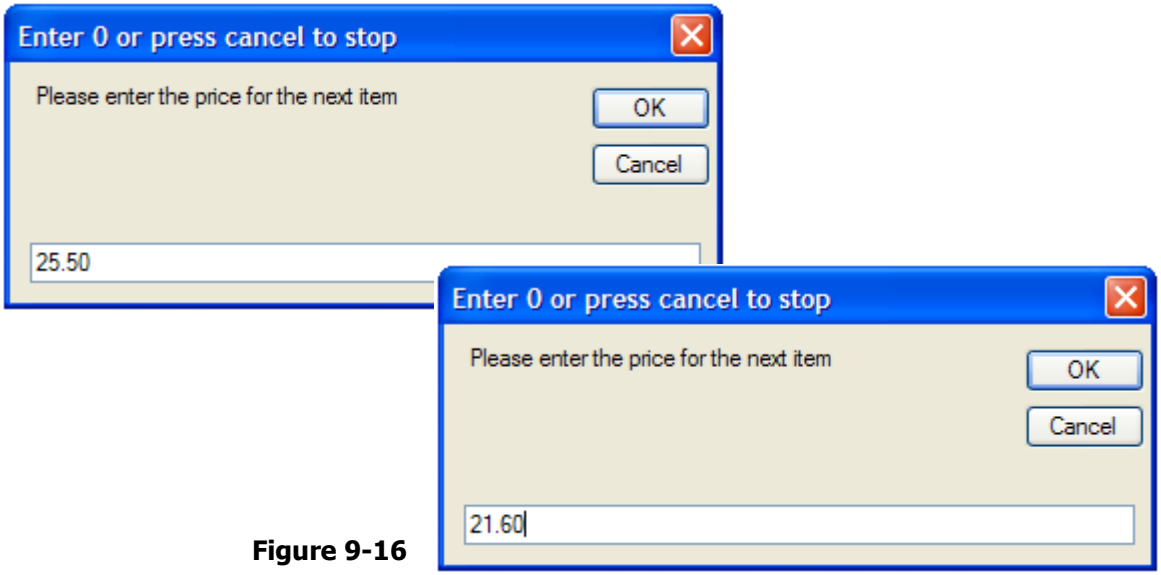


Figure 9-16

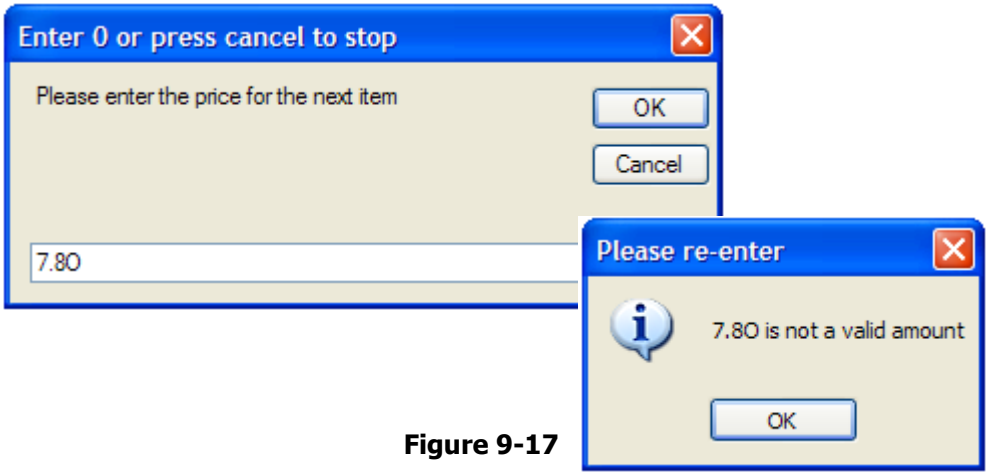


Figure 9-17

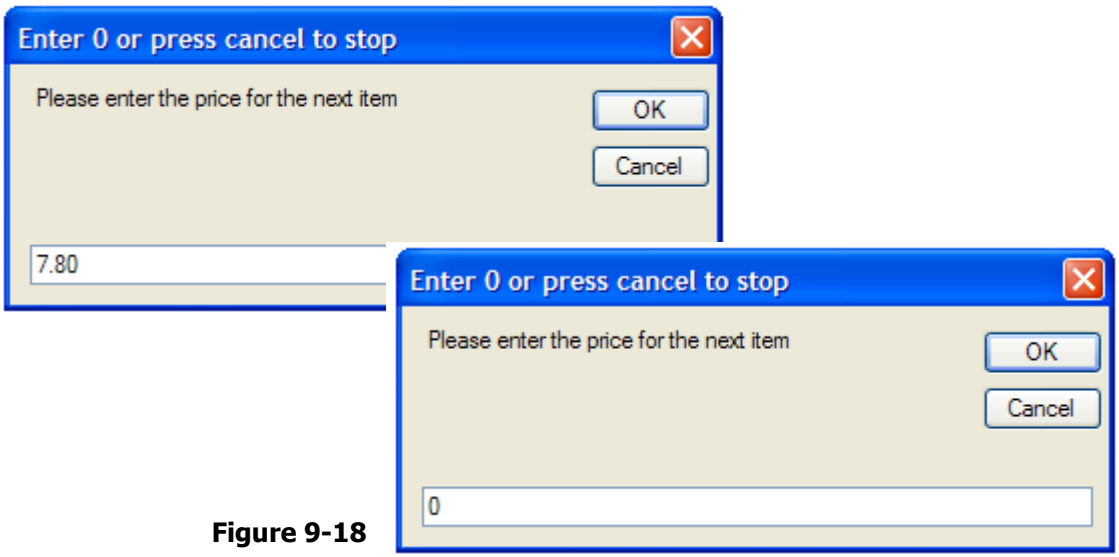


Figure 9-18

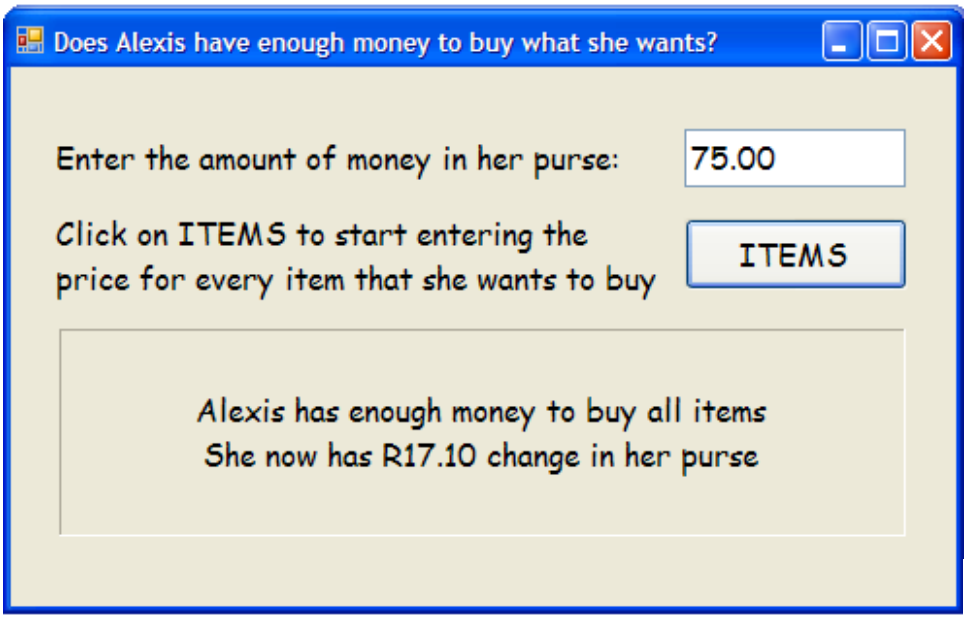


Figure 9-19

9.3.4 PRACTICAL EXAMPLE 3

9.3.4.1 PROGRAM PLANNING

Be Prepared is a company that sells electric and gas stoves to customers. Each of Prepared's salespeople receives a commission based on the total of his or her sales.

The algorithm should allow the user to enter the salesperson's name and amount of electric stoves' sales and the amount of gas stoves' sales. It should then calculate the commission on the sum of the sales amounts and display the name and commission for each of the salespeople. A salesman name of ZZZZZ is entered to indicate the end of the input data. The total sales and total commission on sales of all salespeople should also be calculated and displayed with appropriate messages.

Use the following information to code the algorithm:

Sales in Rand	Commission
1 – 100 000	4% on sales
100 000.01 – 300 000	R4 000 + 7.5% on sum of sales over R100 000
300 000.01 or more	R19 000 + 14% on sum of sales over R300 000

	Description	Type	Name of variable
Input:	name of salesman	string	name
	electric stove sales	real	elecSales
	gas stove sales	real	gasSales
Output:	names of salesman	string	name
	commission earned	real	commission
	total sales	real	totSales
	total commission	real	totComm

I	P	O
name elecSales gasSales	Prompt for input fields Enter input fields Calculate commission Calculate totals Display name and commission Display totals	name commission totSales totComm

Algorithm:

```
SalesmenCommission
        totSales = 0
        totComm = 0
        display "Enter the name of the first salesman, enter ZZZZZ to stop"
                                                    ~ display on new line
        enter name
        do while name NOT = "ZZZZZ"
                display "Enter the electric stove sales"    ~ display on new line
                enter elecSales
                display "Enter the gas stove sales"         ~ display on new line
                enter gasSales
                if elecSales is numeric AND gasSales is numeric
                        totalSales = elecSales + gasSales
                        ~ calculate commission
                        if totalSales <= 100000 then
                                commission = totalSales * 0.04
                        else
                                if totalSales <= 300000 then
                                        commission = 4000 +
                                                (totalSales – 100000) * 0.075
                                else
                                        commission = 19000 +
                                                (totalSales – 300000) * 0.14
                                endif
                        endif
                        ~ calculate totals
                        totSales = totSales + totalSales
                        totComm = totComm + commission
                        ~ display details of one salesman
                        display "The commission for ", name, " = R", commission
                                                    ~ display on new line
                        ~ proceed to the next salesman
                        display "Enter the name of the next salesman, ZZZZZ to stop"
                        enter name
                else
                        display "The sales values were invalid"
                endif
        loop
```

```
        ~  Display the final results
        display "The total amount sold by all the salespeople is R", totSales
                                                        ~ display on new line
        display "Total commission earned by all the salespeople is R",
                totComm                                 ~ display on new line
end
```

Possible input values with respective output results:

```
Enter the name of the first salesman, enter ZZZZZ to stop    Nicolas
Enter the electric stove sales  R150799.50
Enter the gas stove sales R200085.50

The commission for Nicolas = R26123.90

Enter the name of the next salesman, ZZZZZ to stop    Thomas
Enter the electric stove sales  R50000.00
Enter the gas stove sales R40000.00

The commission for Thomas = R3600.00

Enter the name of the next salesman, ZZZZZ to stop    ZZZZZ

The total amount sold by all the salespeople is R440885.00
Total commission earned by all the salespeople is R29723.90
```

9.3.4.2 IMPLEMENTATION IN VB.NET

USER INTERFACE

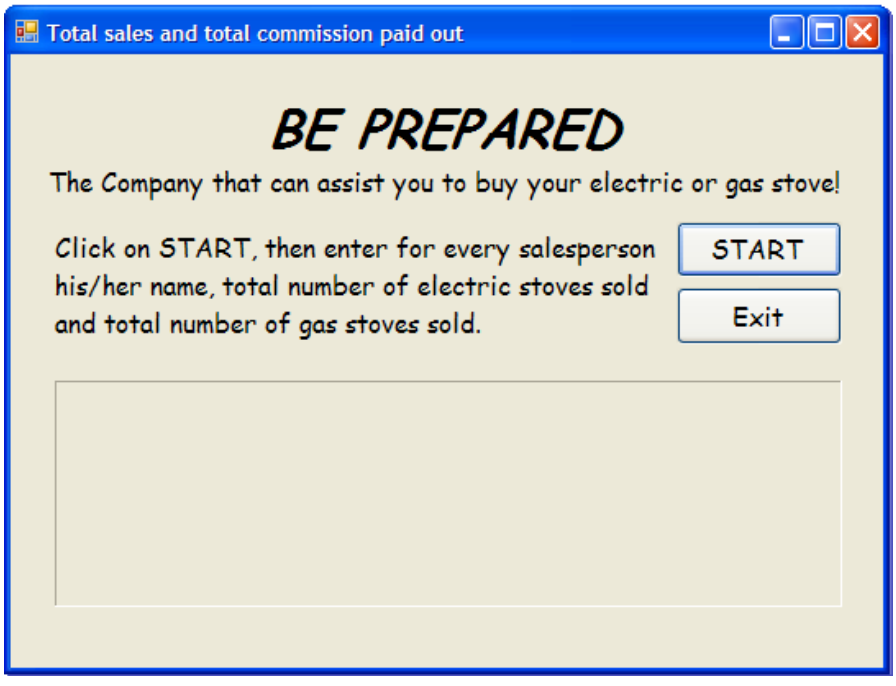


Figure 9-20

CODE IN THE CODE EDITOR

```
Private Sub btnStart_Click(. . .) Handles btnStart.Click
   Dim decTotSales As Decimal = 0
   Dim decTotComm As Decimal = 0
   Dim strName, strElecSales, strGasSales As String
   Dim decElecSales, decGasSales, decTotalSales As Decimal
   Dim decCommission As Decimal
   strName = InputBox_
           ("Enter the name of the first salesperson", _
           "Enter ZZZZZ or click on cancel to stop ")
   Do While strName.ToUpper() <> "ZZZZZ" And strName <> ""
      strElecSales = InputBox_
           ("Enter the electric stove sales", strName)
      strGasSales = InputBox_
           ("Enter the gas stove sales", strName)
      If Decimal.TryParse(strElecSales, decElecSales) _
         AndAlso _
         Decimal.TryParse(strGasSales, decGasSales) Then
         decTotalSales = decElecSales + decGasSales
         If decTotalSales <= 100000D Then
            decCommission = decTotalSales * 0.04D
         Else
            If decTotalSales <= 300000D Then
               decCommission = 4000D + _
                           (decTotalSales - 100000) * 0.075D
            Else
               decCommission = 19000D + _
                           (decTotalSales - 300000) * 0.14D
            End If
         End If
         decTotSales = decTotSales + decTotalSales
         decTotComm = decTotComm + decCommission
         MessageBox.Show("The commission for " & strName & _
            " = R" & decCommission.ToString("N2"), " ", _
            MessageBoxButtons.OK, MessageBoxIcon.Information)
         strName = InputBox_
           ("Enter the name of the next salesperson", _
            "Enter ZZZZZ or click on cancel to stop ")
      Else
         MessageBox.Show("The sales values were invalid", _
                         "Please re-enter")
      End If
   Loop
   lblAnswer.Text = _
   "The total amount sold by all the salespeople is R" & _
   decTotSales.ToString("N2") & ControlChars.NewLine & _
   "Total commission earned by all the salespeople is R" _
   & decTotComm.ToString("N2")
End Sub
```

OUTPUT

Figure 9-21 to **Figure 9-23** indicate the values entered in the input boxes and respective commission displayed in a message box for every sales representative. When a name of ZZZZZ is entered, the total sales and commission will be displayed as indicated in **Figure 9-24**. Take note of the error message that will be displayed if an incorrect amount is entered, as indicated in **Figure 9-25**.

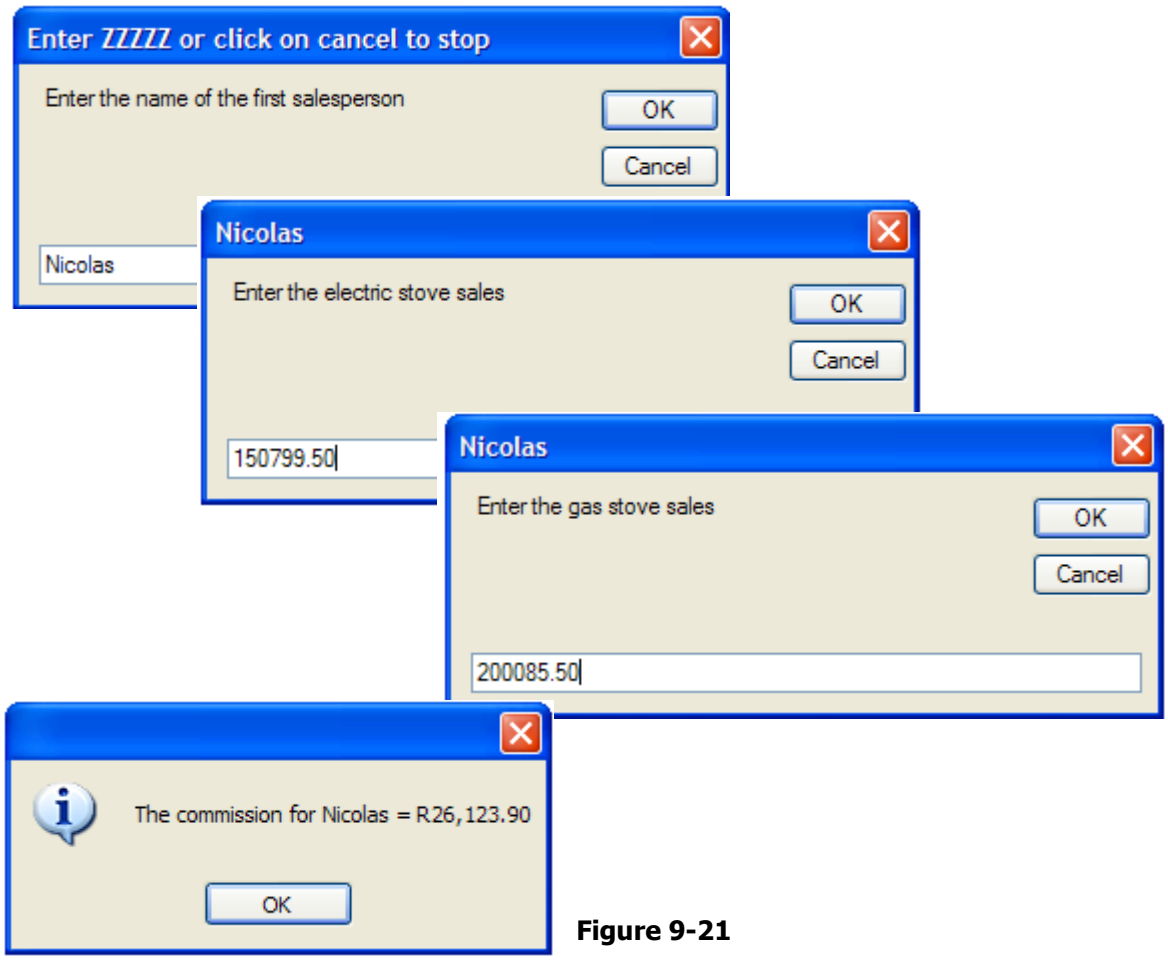


Figure 9-21

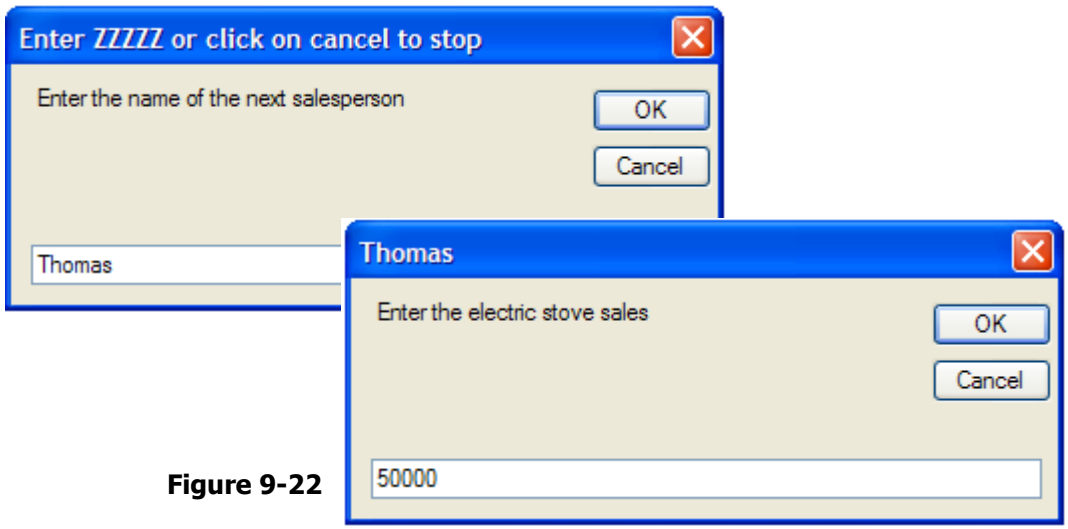


Figure 9-22

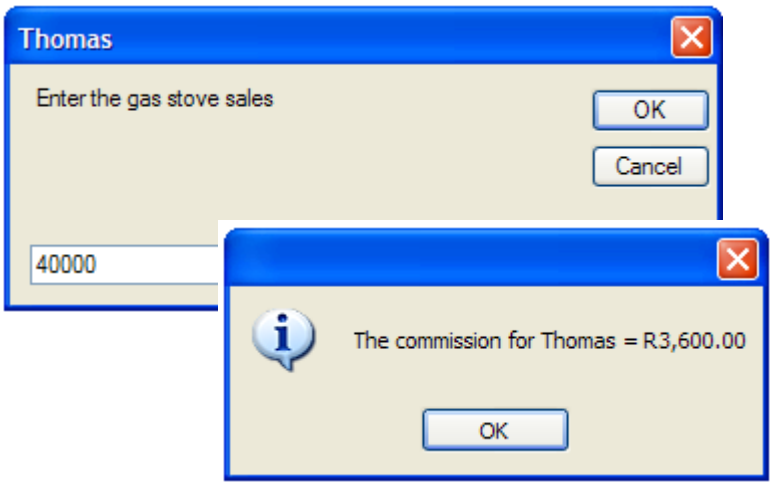


Figure 9-22 (Continued)

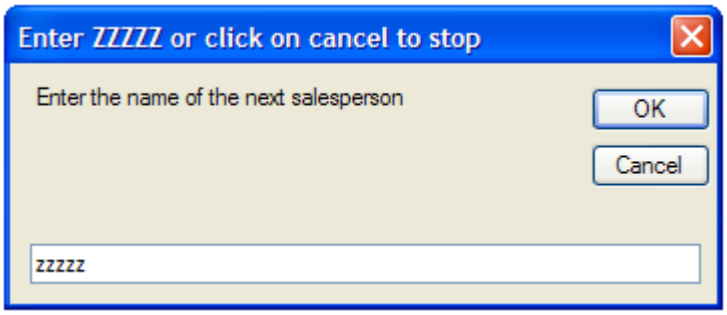


Figure 9-23

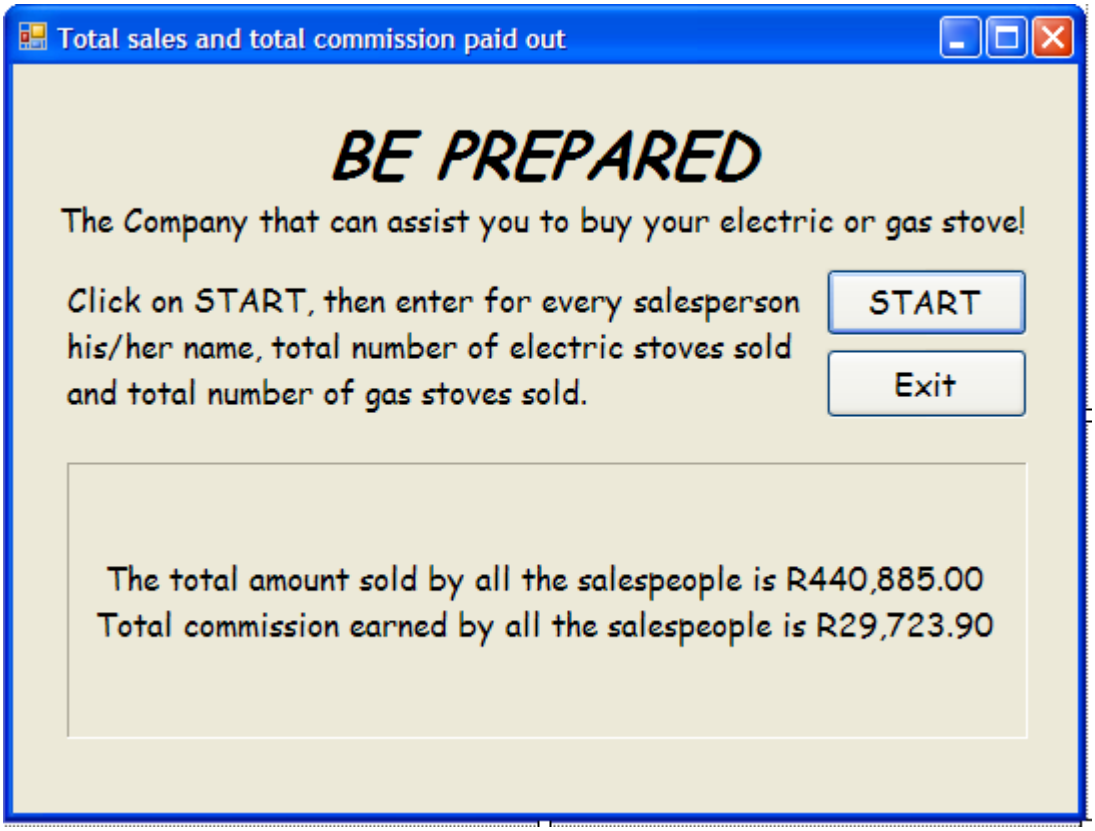


Figure 9-24

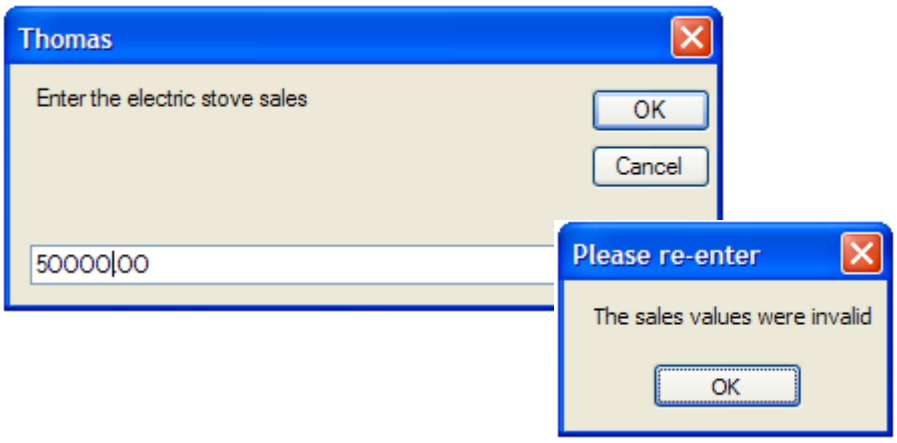


Figure 9-25

9.4 EXAMPLE OF THE DO-UNTIL-LOOP

We have already discussed the do-until-loop which is a posttest loop that tests the condition at the end of the loop.

Let us study an example:

9.4.1 PROGRAM PLANNING

Problem statement:

Angel has offered to pick strawberries for her mother who wants to cook jam. She needs between 4.5 and 5.5 kilograms of strawberries for the jam. Angel, who is only a little girl can pick and bring between 400 and 900 grams of strawberries at a time to empty her basket into her mother's container on the scale. The user is asked to enter the weight of the strawberries in grams every time Angel brings strawberries. The program must calculate and display how many times she has to go to the garden to pick strawberries before her mother has enough to cook. The program must also display the total weight of the strawberries picked.

	Description	Type	Name of variable
Input:	weight in grams	real	weight
Output:	number of times	integer	noTimes
	total weight in kg	real	totWeight

I	P	O
weight	Prompt for weight Enter weight Count noTimes Accumulate totWeight Display noTimes, totWeight	noTimes totWeight

Algorithm

```
CalculateNumberTimes
~       Calculate number of times Angel picked strawberries
        noTimes = 0
        totWeight = 0

        do
                display "Enter the weight of the strawberries in grams Angel picked"
                                                        ~ display on new line
                enter weight

                if weight is numeric
                        totWeight = totWeight + weight / 1000
                                ~ convert to kilogram and add to total weight
                        noTimes = noTimes + 1           ~accumulate count
                else
                        display weight, " is not a valid input value, please re-enter"
                endif

        loop until totWeight >= 4.5

        ~ number of times has been calculated
        display "The number of times Angel picked strawberries is ", noTimes
                                                        ~ display on new line
        Display "She picked a total of ", totWeight, "kg of strawberries"
end
```

Test data:
Possible input values with respective output results:

```
Enter the weight of the strawberries in grams Angel picked 500
Enter the weight of the strawberries in grams Angel picked 88O
88O is not a valid input value, please re-enter
Enter the weight of the strawberries in grams Angel picked 880
Enter the weight of the strawberries in grams Angel picked 720
Enter the weight of the strawberries in grams Angel picked 450
Enter the weight of the strawberries in grams Angel picked 570
Enter the weight of the strawberries in grams Angel picked 770
Enter the weight of the strawberries in grams Angel picked 360
Enter the weight of the strawberries in grams Angel picked 320

The number of times Angel picked strawberries is 8
She picked a total of 4.570kg strawberries
```

9.4.2 IMPLEMENTATION IN VB.NET

USER INTERFACE:



Figure 9-26

CODE IN THE CODE EDITOR

```
Private Sub btnStart_Click(. . .) Handles btnStart.Click

   Dim intNoTimes As Integer = 0
   Dim decTotWeight As Decimal = 0
   Dim intWeight As Integer
   Dim strWeight As String

   Do
       strWeight = InputBox_
      ("Enter the weight of the strawberries in grams Angel picked",_
       "Angel picks strawberries")

      If Integer.TryParse(strWeight, intWeight) Then
         decTotWeight = decTotWeight + _
                        Convert.ToDecimal(intWeight / 1000)
         intNoTimes = intNoTimes + 1
      Else
         MessageBox.Show(strWeight & " is not a valid input value",_
         "Please re-enter", MessageBoxButtons.OK, MessageBoxIcon.Error)
      End If

   Loop Until decTotWeight > 4.5


   lblAnswer.Text = "The number of times Angel picked strawberries is " _
           & intNoTimes & ControlChars.NewLine & _
           "She picked a total of " & decTotWeight.ToString("N3") & _
           "kg of strawberries"
End Sub
```

OUTPUT

Figure 9-27 to **Figure 9-29** correspond to the test data on page 241. When the total weight that has been accumulated in the loop, exceeds 4.5 kg, the loop will terminate and the output, as indicated in **Figure 9-30**, will be displayed.

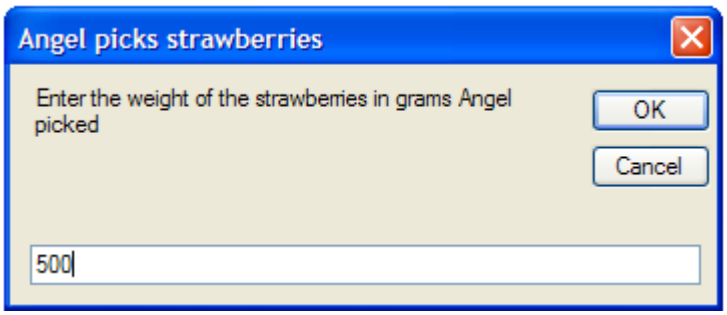


Figure 9-27

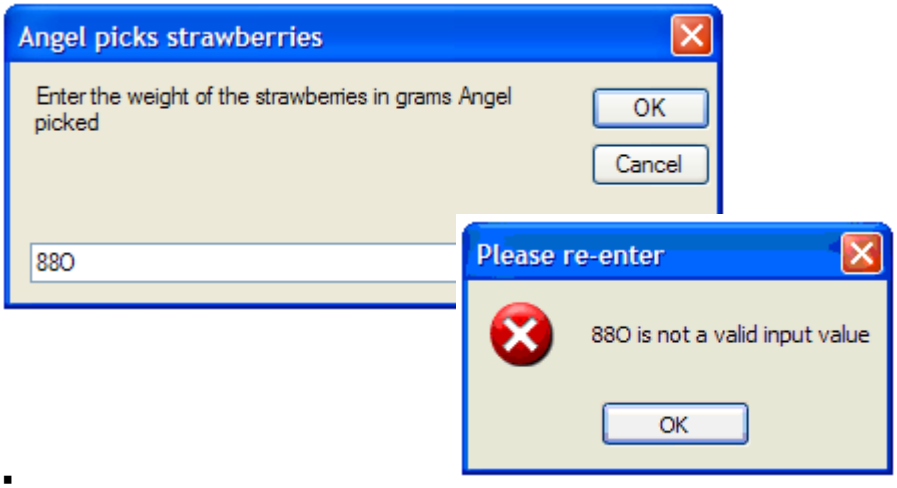


Figure 9-28

:

:

:

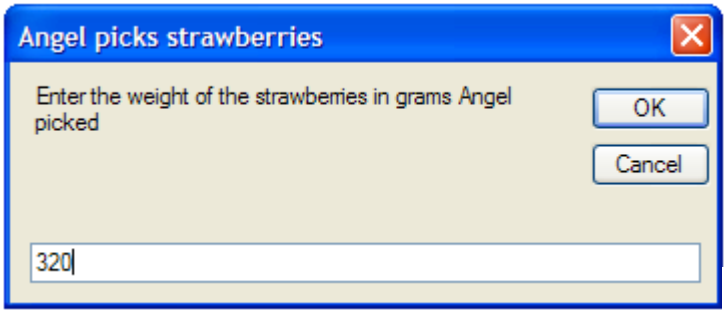


Figure 9-29



Figure 9-30



Every one of the following problems contain do-loops. When planning your program you have to decide which type of loop will be the best suited to solve this particular problem. If the problem contains decisions to be made, you may also decide whether you are going to use an if-statement or the select case statement.

1. A number of people joined the Streamline Club during the first week of January and the weight of each member was written on their membership card. During the last week of March everybody was weighed again and their new weights were also written on their cards. Plan and write a program to enter the name, first weight and last weight. Calculate the loss of weight (or gain, if they gained weight) and display the results i.e. name and weight loss or gained with a suitable message. The program will terminate when a beginning weight of 0 is entered. The average weight loss must be display at the end.

2. Jerry is not sure how much money he has, but he knows he has between R100 and R135 in R1, R2 and R5 coins. The user will enter the price of the item he wants to buy. The price does not exceed R100. Jerry pays for the item bought giving one coin at a time to the salesperson. The value of this coin is entered every time he presents the coin and it is subtracted from the

amount still due. Determine how many coins he used to pay for the item and display this number. Remember that you do not know how many of what coins he has. There may be an equal number of all the coins, no R1 coins or no R5 coins, etc. Also calculate and display the amount of change, if any.

3. The Walk-In restaurant is very popular amongst employees in the area because they provide "home-cooked" meals consisting of meat and vegetables already dished up. They provide different meals for men (code = M) and women (code = L). Children (code = C) pay half price of a ladies' meal. The prices are as follows:

Type of meal	**Price per meal**
Men	R20.50
Ladies	R16.80

The user enters the number of different meals available in the beginning of the program i.e. for men, ladies and children. Every time a person orders meals (He/she may also buy for friends) the type of meal and the number is entered. Firstly, if the meals are available, this number is subtracted from the relevant total and then the amount due is calculated and displayed. If there are no meals of that specific type available, a suitable message must be displayed. If fewer meals than the customer orders are available, a new transaction is started. When all three types of meals are finished, the program will terminate. You may assume that for the sake of this program, all the meals will be bought.

(If you want to change the program so that not all the meals are sold, you may include another condition to terminate the program. In this case you can also display how many meals and of what type, are not sold).

4. A florist wants to send coupons to her 420 regular customers based on the number of orders the customer placed during the past year. The amount on the coupon depends on the number of orders according to the following table:

Number of orders	**Amount on coupon**
2 – 6	The number of orders times R10
7 – 15	The number of orders times R11.50
16 – 30	The number of orders times R14
31 or more	The number of orders times R17

The user is asked to enter the name of the customer and the number of orders placed. Determine the value of the customer's coupon and then

display the following message on the screen (In the example the test data name is Carol and the coupon is worth R50).

Thank you Carol! You receive a coupon worth R50 to collect lovely flowers!

5. The learners in the CLEVER SCHOOL pay school fees according to their grade. The name and the grade of the learner must be entered. The learners in the first grade pay R14 for the months February to November (excluding July). The fee increases by R4.50 for every next grade (e.g. 2nd grade pays R18.50, 3rd grade pays R23.00, etc.). Calculate and display the name and annual school fee of every learner in the CLEVER SCHOOL. At the end the average annual school fee must be displayed. A name of ABCD is entered after the details of all the school's learners have been entered.

6. The manager of the Nutcracker Hotel needs a program to print invoices for the guests who stayed at his hotel. The input to this program is the name of the guest, the number of nights he/she spent in the hotel and also the number of breakfasts taken. The number of nights and the number of breakfasts need not be the same. A guest receives one free night and one free breakfast for every night spent in the hotel. You may assume that the number of free breakfasts will never exceed the number of the input breakfasts.

   The price of a night is R340 and a breakfast costs R65. The invoice for every guest must contain the name of the guest, number of nights stayed, number of breakfasts, number of free nights and the final amount due.

   At the end of the program the total amount paid by all the guests must be displayed.

   The programmer has to decide on the sentinel used to end the program.

7. People who want to hire a machine to wash their carpets usually go to the WishyWashy Company where they can hire any one of 3 types of machines i.e. type A, B or C for heavy duty, ordinary and light washes. The prices for using the machines are as follows:

Type of machine	Initial cost	Additional cost per hour or part thereof
A	R50.00	R30.00
B	R62.50	R36.25
C	R74.87	R40.50

   The input to this program is the name of the client, the code of the type of machine, the time the machine was used (in minutes).

The amount must be calculated by adding the initial cost to the calculated cost for the time used. Vat of 14% must be added to the amount to be able to present the final amount due to the client to pay.

At the end of the program, a total amount, total vat amount and the final total amount due to WishyWashy must be displayed.

Choose your own sentinel.

For each of these practical assignments, design a suitable, user friendly interface and code as effective as possible.

APPENDIX A
TOOLS FOR PLANNING PROGRAMS

1. INTRODUCTION

Pseudocode has been used throughout these notes when planning programs. There are however, different methods that can be used when representing an algorithm. The use of flowcharts and Nassi-Shneiderman methods will now briefly be discussed.

We will first write the algorithms for solving the two problems in pseudo-code and then repeat the process using flowcharts, followed by the algorithm using a Nassi-Shneiderman diagram.

2. PSEUDOCODE

EXAMPLE 1:

Write an algorithm to enter 2 numbers that are not equal and display the bigger number.

```
FindTheBigger

        display "Enter a number"
        enter num1
        display "Enter the next number – not equal to the first number"
        enter num2

        if num1 > num2 then
                display "num1 is bigger than num2"
        else
                display "num2 is bigger than num1"
        endif
end
```

EXAMPLE 2:

Write an algorithm to enter integers between 5 and 20 until their sum is greater than 200. Display how many integers were entered.

```
CountTheNumber

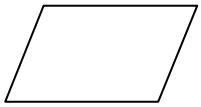
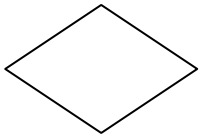
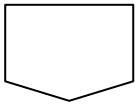
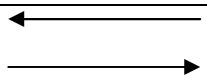
        sum = 0
        count = 0
        do while sum <= 200
                display "Enter an integer between 5 and 20"
                enter num
                sum = sum + num
                count = count + 1
        loop
        display "The number of integers is ", count
end
```

3. FLOWCHARTS

A flowchart is a schematic representation of an algorithm. It illustrates the steps in a process and it consists of a number of specific diagrams joined in a specific manner. It graphically represents the program logic through a series of standard geometric symbols and connecting lines. Different flowchart symbols are used for different aspects of the process.

Flowchart symbols:

The following symbols are used:

Symbol	Description
	**Terminal symbol** This symbol indicates the starting or stopping point in the logic. Every flowchart should begin and end with this symbol.
	***Input/Output symbol*** This symbol represents an input or output process in the algorithm. It is used for reading, writing, displaying, etc.
	***Processing*** This symbol is used for types of processing such as arithmetic statements and assigning values.
	***Decision symbol*** This symbol is used to compare different variables / values that may change the flow of the logic. It may cause the logic to branch in another direction.
	***Module symbol*** This symbol represents another module that must be processed. This module will have its own flowchart.
	***Connector*** The connector joins 2 parts of the flowchart e.g. from one page to the next page.
	***Connecting lines*** These lines connect flowchart symbols with one another

Examples:

1. Draw a flowchart to enter 2 numbers that are not equal and display the bigger number.

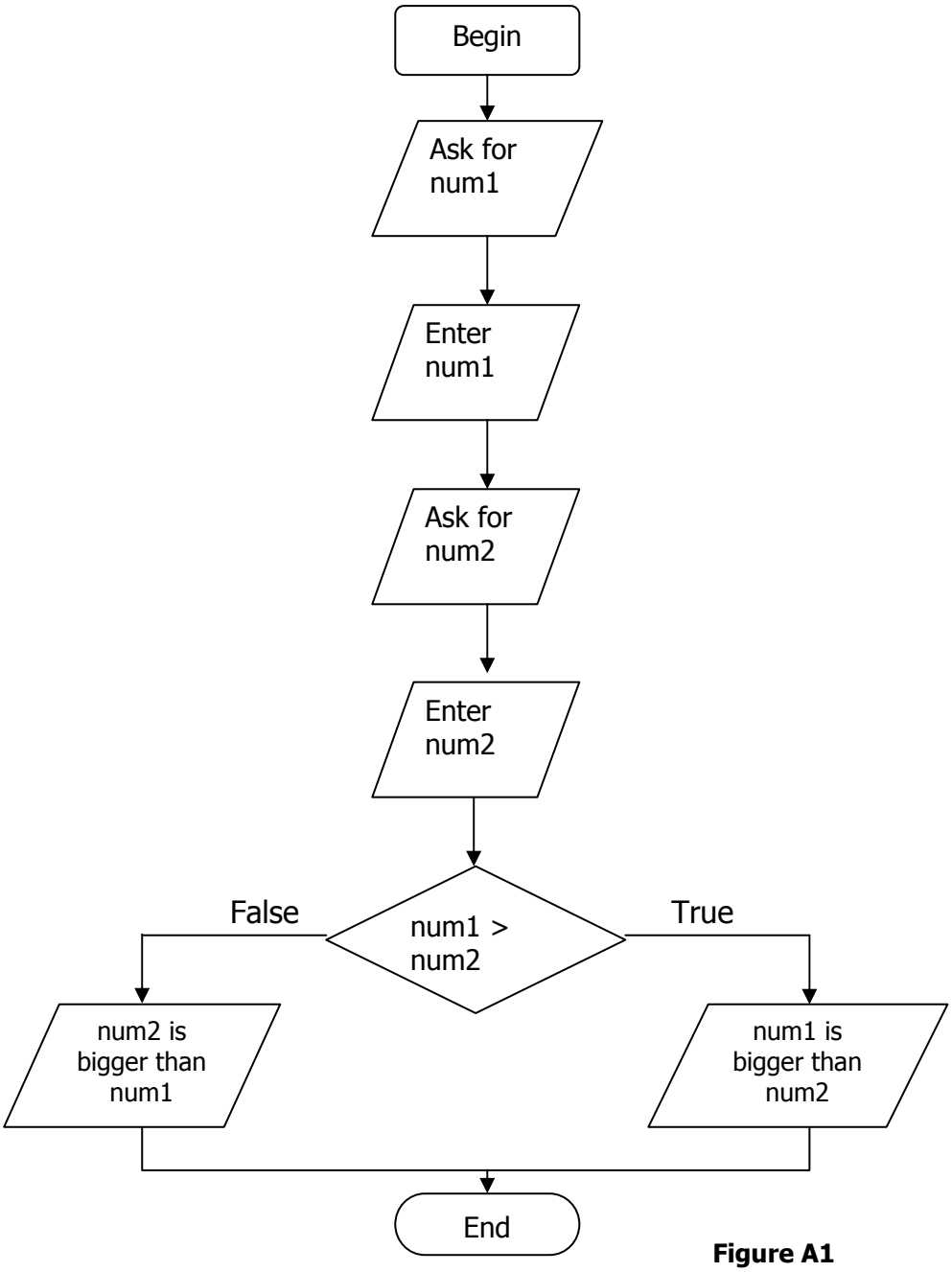


Figure A1

2. Draw a flowchart to enter integers between 5 and 20 until their sum is greater than 200. Display how many were entered.

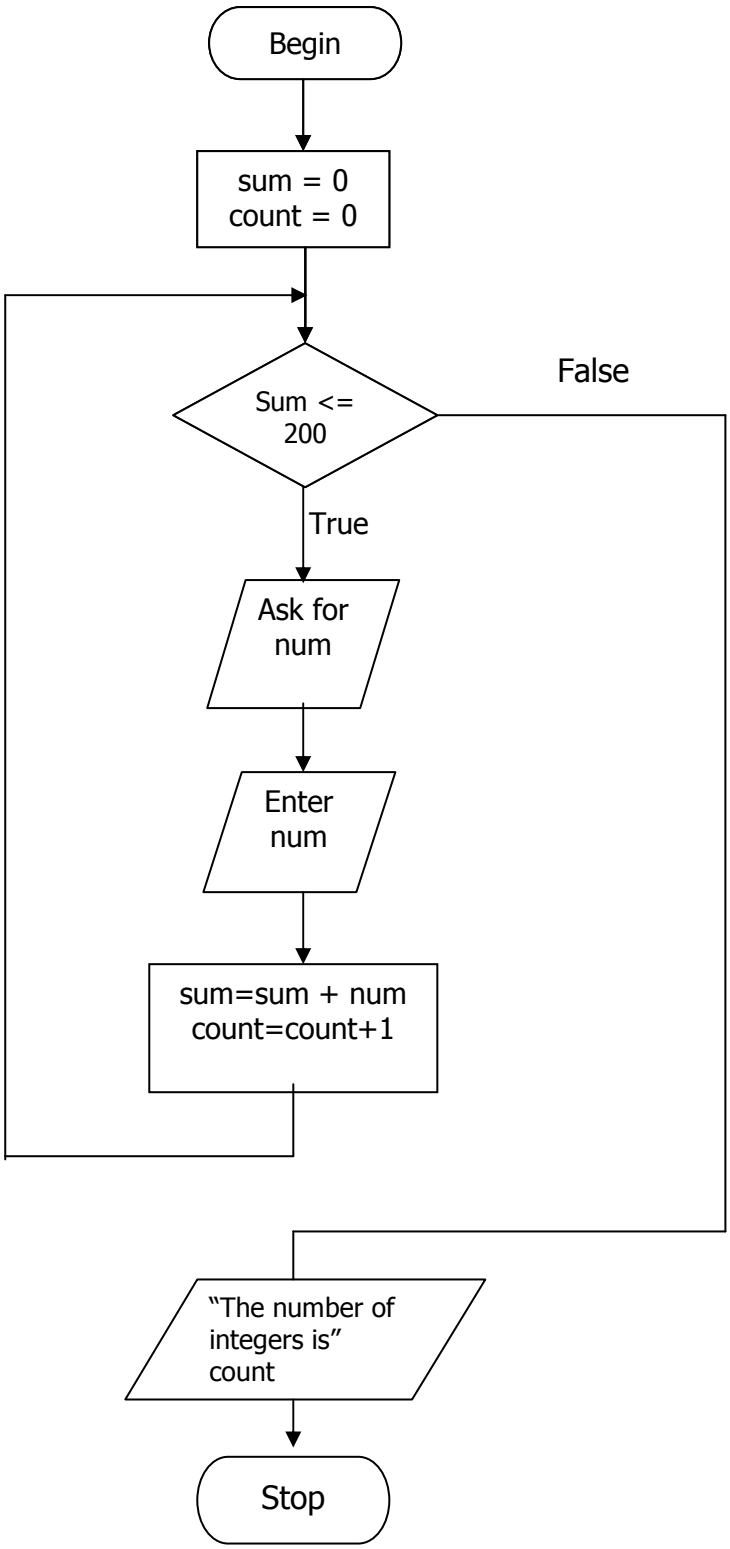


Figure A2

4. NASSI-SHNEIDERMAN

A Nassi-Shneiderman diagram (or NSD) is a graphical design representation for structured programming. Developed in 1972 by Isaac Nassi and Ben Shneiderman, these diagrams are also called *structograms*, as they show a program's structures.

Everything you can represent with a Nassi-Shneiderman diagram you can also represent with a flowchart. For flowcharts of programs, just about everything you can represent with a flowchart you can also represent with a Nassi-Shneiderman diagram.

4.1 SYMBOLS USED TO CONSTRUCT A NASSI-SHNEIDERMAN DIAGRAM

Process Block:

The process block represents the simplest of steps. When a process block is encountered, the statements within the block is processed. When it has completed these statements, it proceeds to the next block.

Decision Block:

This block enables the programmer to include a condition in the diagram which will produce different branches or paths for true and false processing.

Iteration Block:

The iteration block allows the programmer to repeat a block of instructions a number of times while a specific condition is true or until the specific condition is true.

We will now draw Nassi-Shneiderman diagrams to solve our two problems:

1. Draw a Nassi-Shneiderman diagram to enter 2 numbers that are not equal and display the bigger number.

FindTheBigger

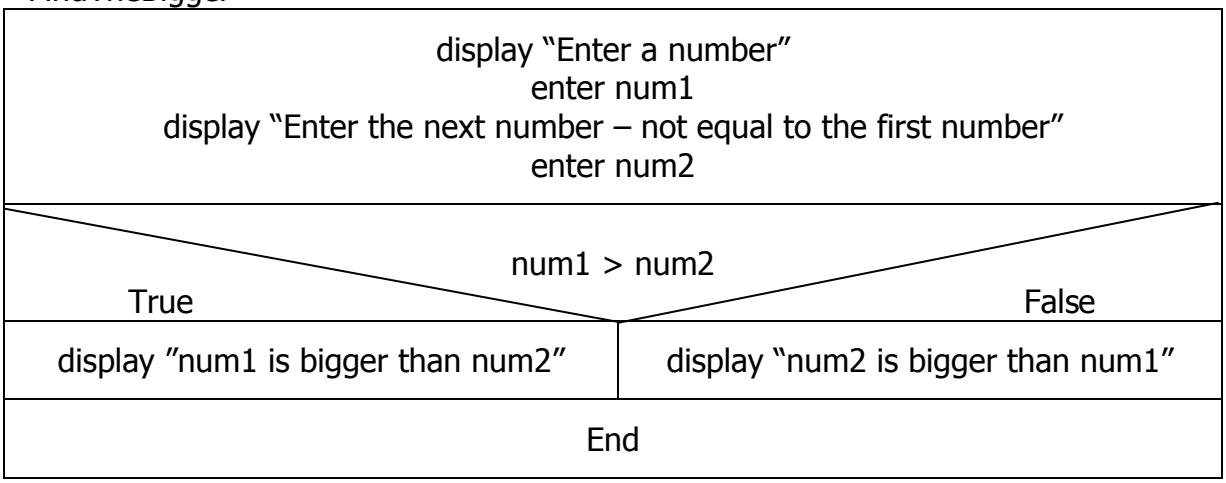


Figure A3

2. Draw a Nassi-Shneiderman diagram to enter integers between 5 and 20 until their sum is greater than 200. Display how many were entered.

CountTheNumber

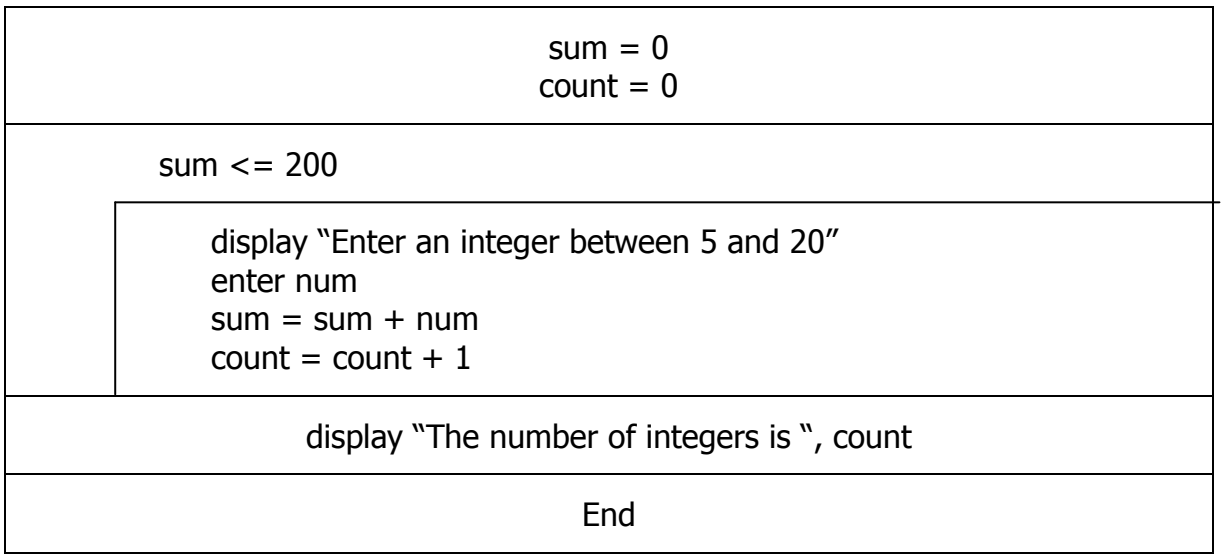


Figure A4

5. INTEGRATED EXAMPLE

Note:
The following integrated example illustrates the use of various program documentation tools and concepts. The complete program is modeled using UML (See appendix B). The complete solution is documented using various methods including:

- ☺ Pseudocode
- ☺ Nassi Shneiderman diagrams
- ☺ Flowcharts

It is advisable to study each of these documented solutions in order to obtain a better understanding of each of the documentation methods.

Problem:

Develop an application that will allow a customer to pay his parking ticket at a shopping mall. The customer will enter his ticket into a ticket machine. This machine consists of a card reading module that determines how long the car has been parked. This value (parking duration in minutes) is sent to the ticket application to determine the amount due and to print a receipt after the customer has paid.



For our purposes you may assume that the total minutes supplied will be entered into a text box. The amount due will then be calculated as follows:

Time parked	**Amount due**
<= 1 hour	Free
> 1 hour, but <= 2 hours	R2.50
> 2 hours, but <= 3 hours	R4.00
> 3 hours	R4.00 + R1.00 for every full hour and 50c for every part of an hour

If the customer parked for free, his time and a suitable message must be displayed on the label (that serves as a receipt).

E.g. You parked for only 45 minutes
Your parking is free

If the parking is not for free, the amount due must be displayed on a message box and the user must then enter every amount paid into an input box until the total amount paid is more or equal to the amount due. After every amount has been entered into the input box, the total amount paid and remaining amount due (if any) must be displayed in a message box. When the full amount has been paid,

the amount due, total paid and change (if any) must be displayed on the label to serve as a receipt.

E.g. Total amount due: R8.50
First amount paid: R5
Message: You paid R5.00. You still need to pay R3.50
Next amount paid: R2
Message: You paid R7.00. You still need to pay R1.50
Next amount paid: R2
Receipt (displayed on label)
You parked for 7 hours and 20 minutes
Amount due for parking = R8.50
Amount paid = R9.00
Change = R0.50

5.1 METHOD 1: ALGORITHM IN PSEUDO-CODE

	Description	Type	Name of variable
Input:	minutes parked	integer	minutes
	amount paid	real	amtPaid
Output:	hours parked	integer	hours
	minutes parked (less than 1 hour)	integer	min
	amount due	real	amount
	change to receive	real	change

I	P	O
minutes amtPaid	Prompt for input fields Enter input fields Calculate time, amount, amount paid, change Display output fields	hours min amount change

Algorithm:

CalcParking

~ Calculate the amount due for parking

```
totPaid = 0
display "Provide minutes parked"        ~ display on new line
enter minutes

if minutes numeric then
    hours = minutes \ 60
    min = minutes mod 60
```

```
            if minutes <= 60 then
                amount = 0
            else
                if minutes <= 120 then
                    amount = 2.5
                else
                    if minutes <= 180 then
                        amount = 4
                    else
                        extra = hours - 3
                        amount = 4 + extra
                        if min > 0 then
                            amount = amount + 0.5
                        endif
                    endif
                endif
            endif
            if amount > 0 then
                display "Minutes = ", minutes, "  Amount  =  ", amount
                                ~ display on new line
            else
                Do
                    display "Enter money to pay for parking"
                                                ~display on new line
                    enter amtPaid
                    if amtoPaid is numeric then
                        totPaid = totPaid + amtPaid
                    endif
                    if totPaid < amount then
                        display "Amount still to pay ",
                            (amount - totPaid)   ~display new line
                    endif
                until totPaid >= amount
                if totPaid > amount then
                    change = 0
                else
                    change = totPaid - amount
                endif
                display "Hours = ", hours, "  minutes = ", min,
                    "  Amount = R", amount, "  change = R", change
                                ~ display on new line
            else
                display "Minutes parked ", minutes, "  Amount due = 0"
                                ~ display on new line
        else
            display "Invalid minutes entered" ~ display on new line
end
```

METHOD 2: FLOWCHART

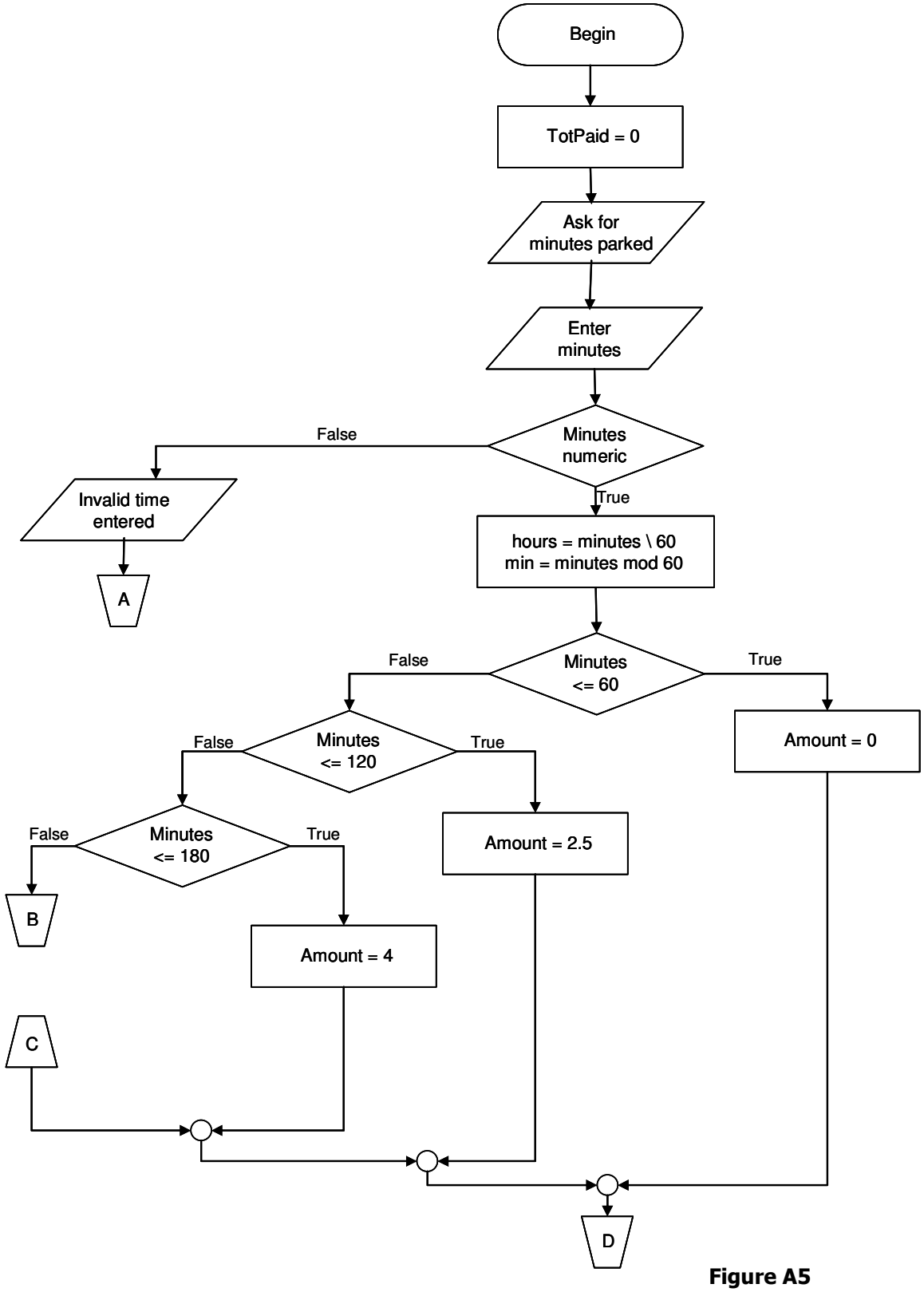


Figure A5

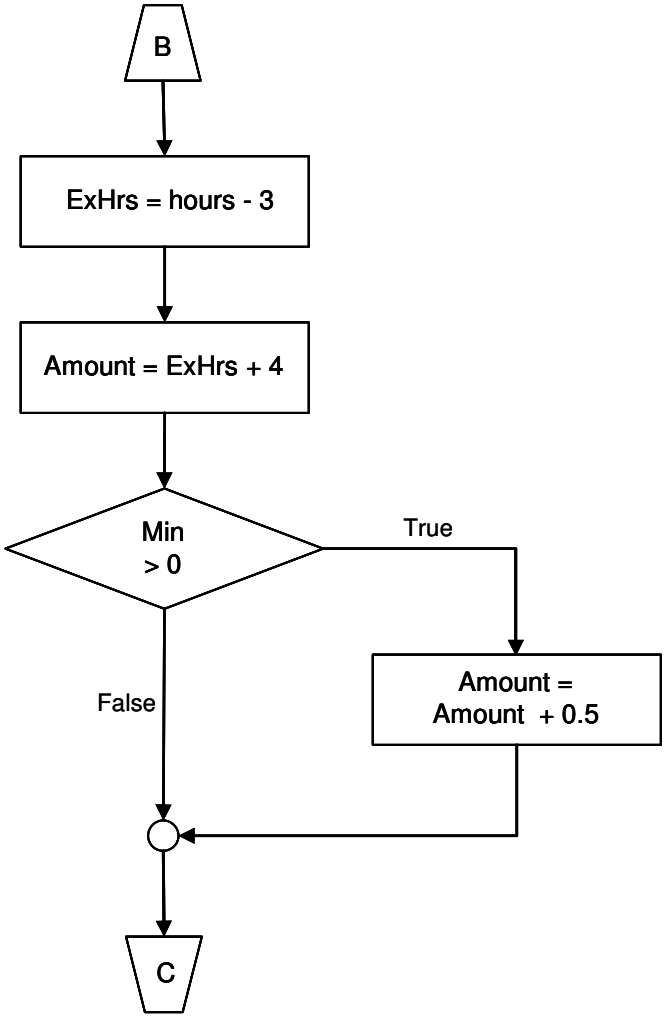


Figure A5 (Continued)

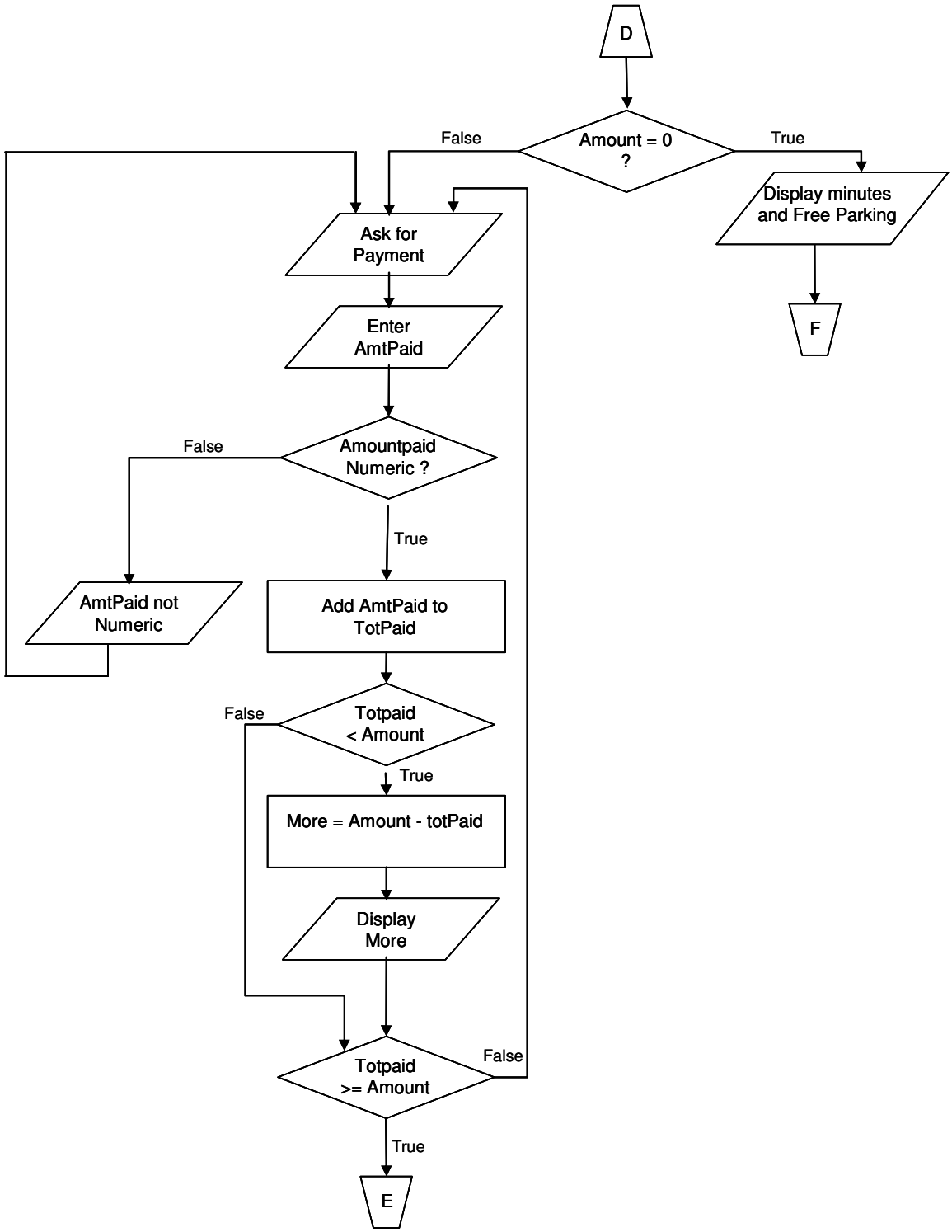


Figure A5 (Continued)

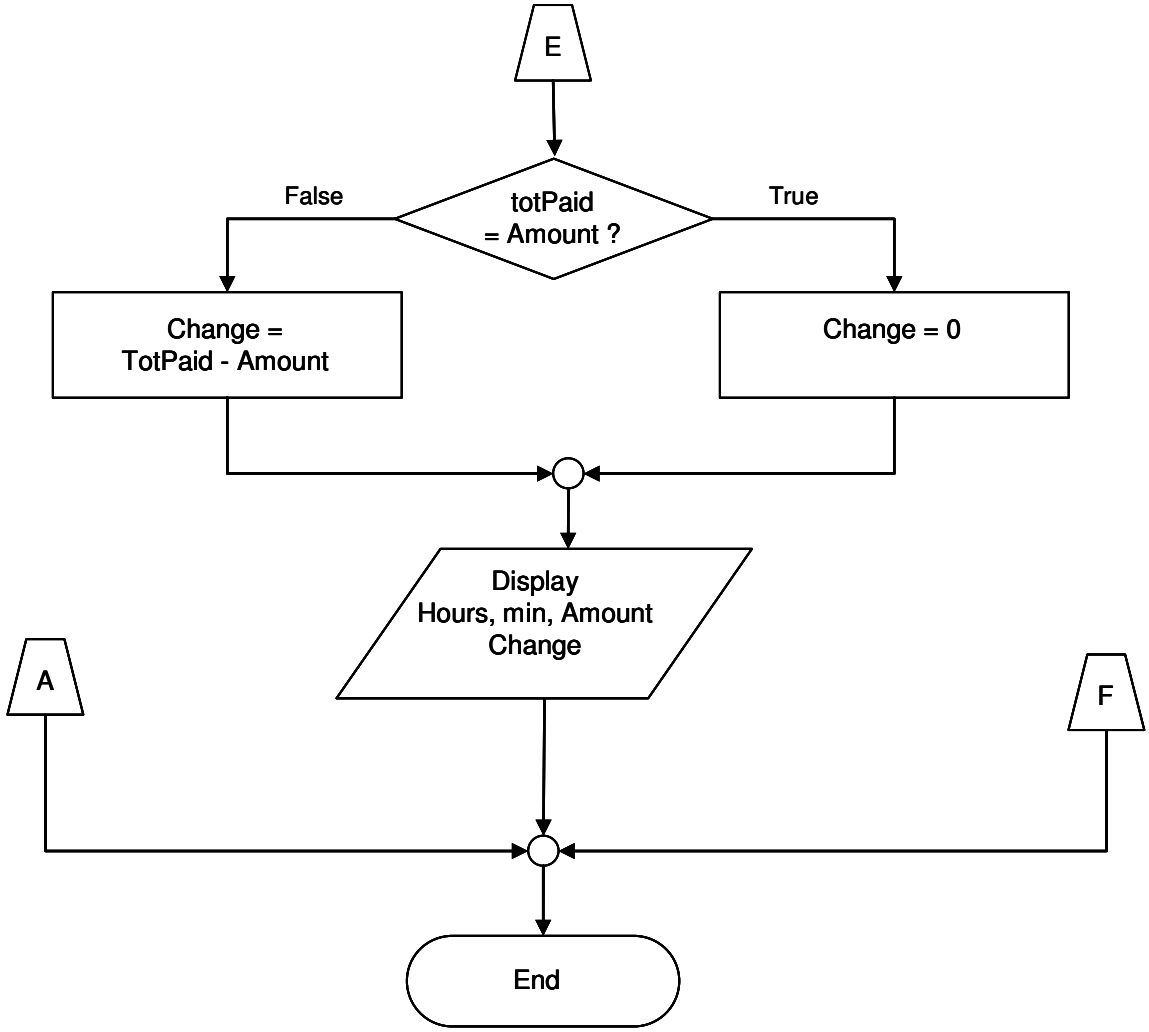


Figure A5 (Continued)

5.3 METHOD 3: NASSI SHNEIDERMAN DIAGRAM

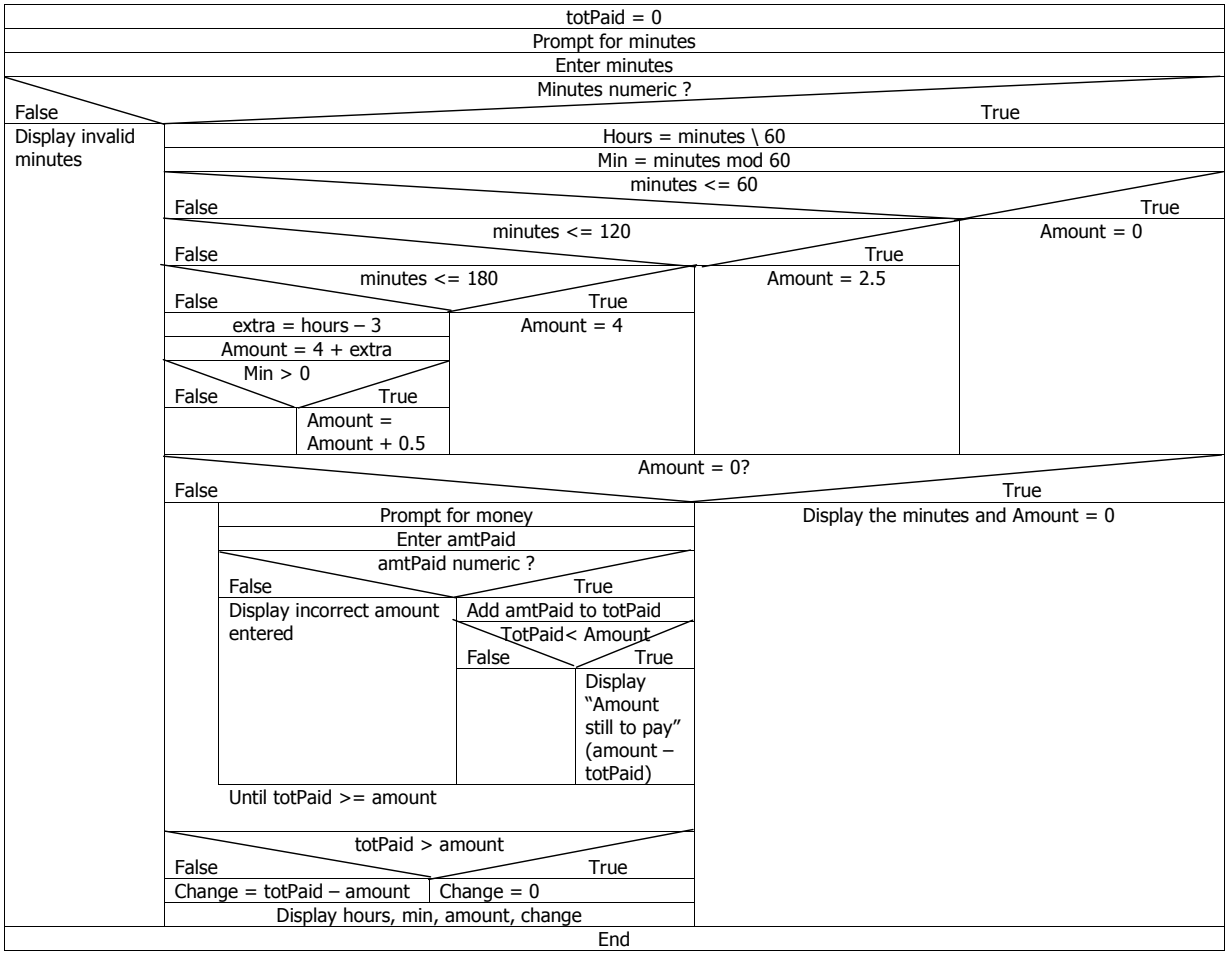


Figure A6

5.4 PROGRAM SOLUTION IN VB.NET

USER INTERFACE:

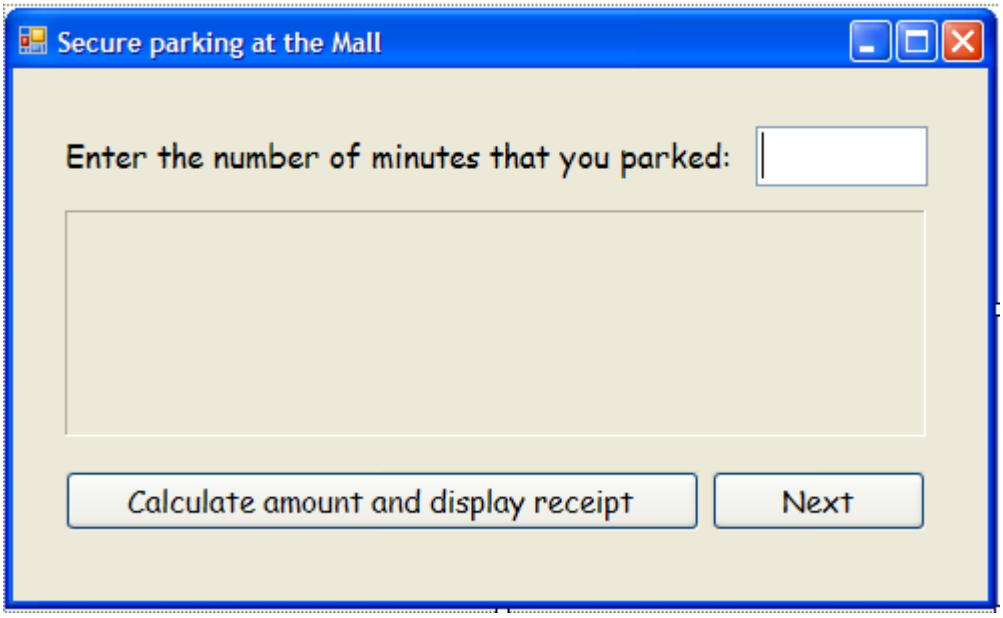


Figure A7

CODE IN CODE EDITOR:

```
Private Sub btnCalculate_Click(. . .) Handles btnCalculate.Click
   Dim intMinutes, intHours As Integer
   Dim strAmtPaid As String
   Dim intExtraHours, intMin As Integer
   Dim decAmount, decAmtPaid, decChange As Decimal
   Dim decTotalPaid As Decimal = 0
   If Integer.TryParse(txtHours.Text, intMinutes) Then
      intHours = intMinutes \ 60
      intMin = intMinutes Mod 60
      If intMinutes <= 60 Then
         decAmount = 0
      ElseIf intMinutes <= 120 Then
         decAmount = 2.5
      ElseIf intMinutes <= 180 Then
         decAmount = 4.0
      Else
         intExtraHours = (intHours - 3)
         decAmount = 4 + intExtraHours
         If intMin > 0 Then
            decAmount = decAmount + 0.5
         End If
      End If
```

```
      If decAmount = 0 Then
         lblReceipt.Text = "You parked for only " & intMinutes &_
                           " minutes" & ControlChars.NewLine & _
                          "Your parking is free"
      Else
         MessageBox.Show("Your amount due = R" & _
                         decAmount.ToString("N2"), " ", _
               MessageBoxButtons.OK, MessageBoxIcon.Information)
         Do
             strAmtPaid = InputBox("Enter money to pay", " ")
             If Decimal.TryParse(strAmtPaid, decAmtPaid) Then
                decTotalPaid = decTotalPaid + decAmtPaid
             Else
                MessageBox.Show("Sorry, I cannot accept that",_
                        "Error in money", MessageBoxButtons.OK, _
                        MessageBoxIcon.Exclamation)
             End If
             If decTotalPaid < decAmount Then
                MessageBox.Show("You paid R" & decTotalPaid & _
                ". You still need to pay R" & _
                (decAmount - decTotalPaid), " ",_
                MessageBoxButtons.OK, MessageBoxIcon.Information)
             End If
         Loop Until decTotalPaid >= decAmount
         If decTotalPaid = decAmount Then
            decChange = 0
         Else
            decChange = decTotalPaid - decAmount
         End If
         lblReceipt.Text = "You parked for " & intHours & _
               " hours and " & intMin & " minutes" & _
               ControlChars.NewLine & _
               "Amount due for parking = R" & _
               decAmount.ToString("N2") & _
               ControlChars.NewLine & "Amount paid = R" & _
               decTotalPaid.ToString("N2") & _
               ControlChars.NewLine & "Change = R" & _
               decChange.ToString("N2")
      End If
   Else
      MessageBox.Show("You entered an invalid value for hours",_
                     "Please correct", MessageBoxButtons.OK, _
                     MessageBoxIcon.Exclamation)
      txtHours.Text = ""
      txtHours.Focus()
   End If
End Sub
```

OUTPUT:

Example 1:
The customer parked for only 45 minutes and the parking is for free.



Figure A8

Example 2:
The customer parked for 440 minutes. It is equivalent to 7 hours and 20 minutes and the amount due is R8.50. He pays with a R5 coin and 2 R2 coins.

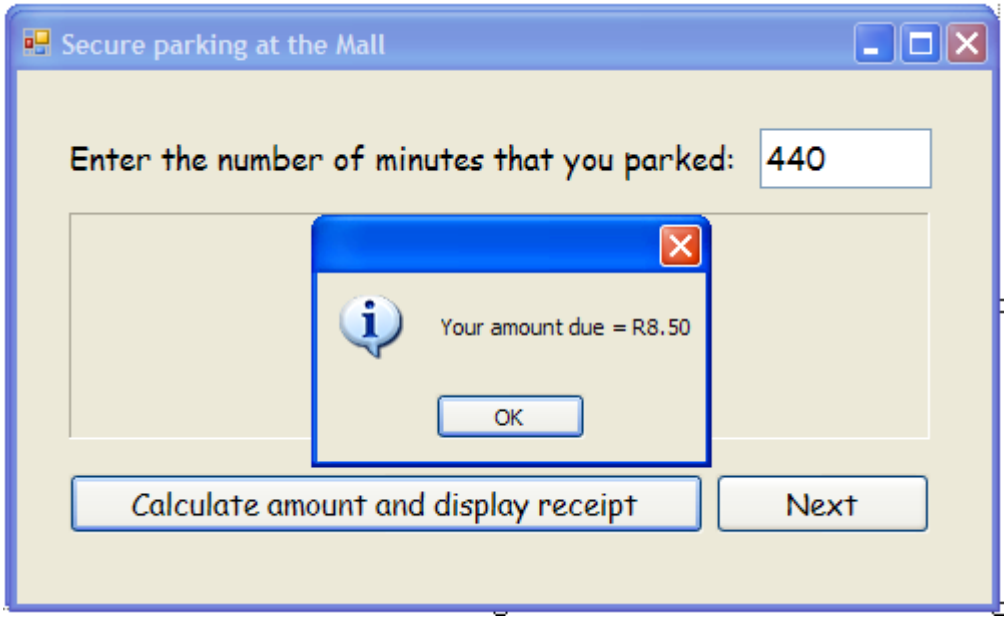
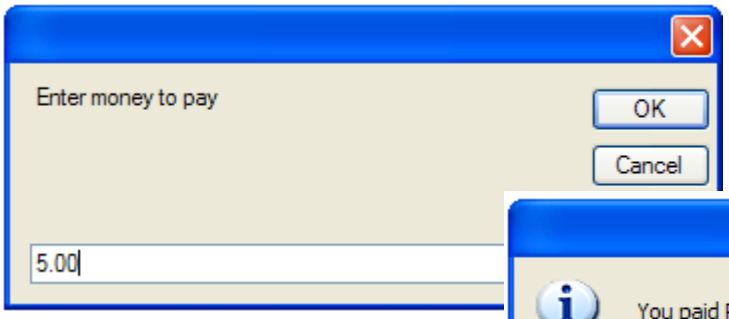


Figure A9



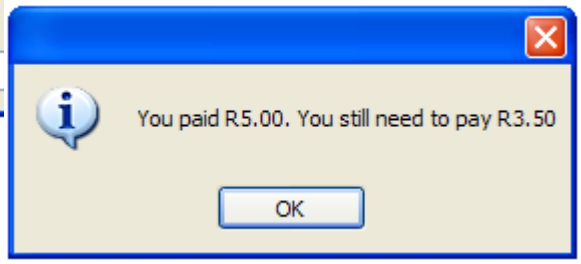
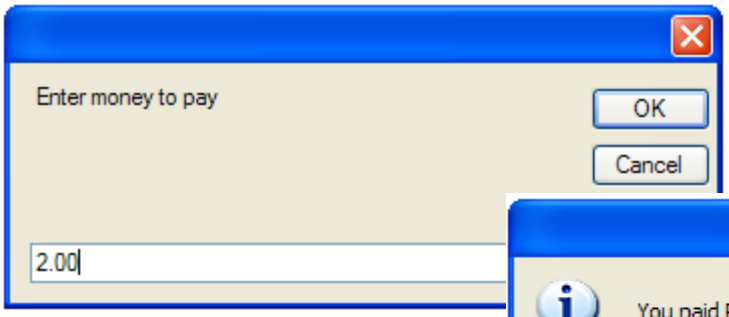


Figure A10



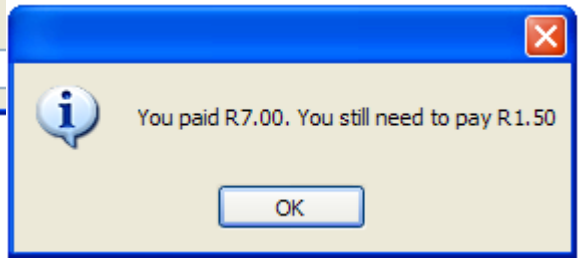


Figure A11

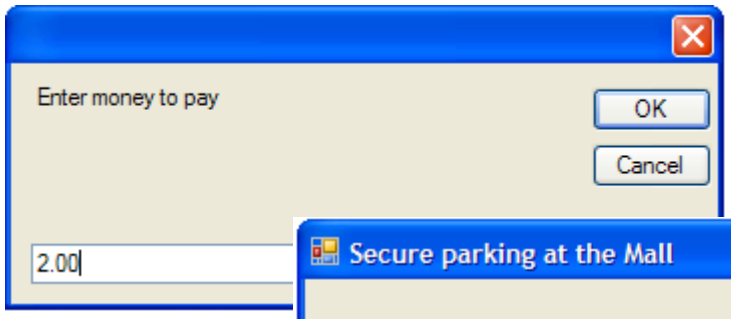




Figure A12

EXAMPLE 3:
The customer parked for 80 minutes. It is equivalent to 1 hour and 20 minutes and the amount due is R2.50. He pays with a R2 coin and a 50c coin.



Figure A13

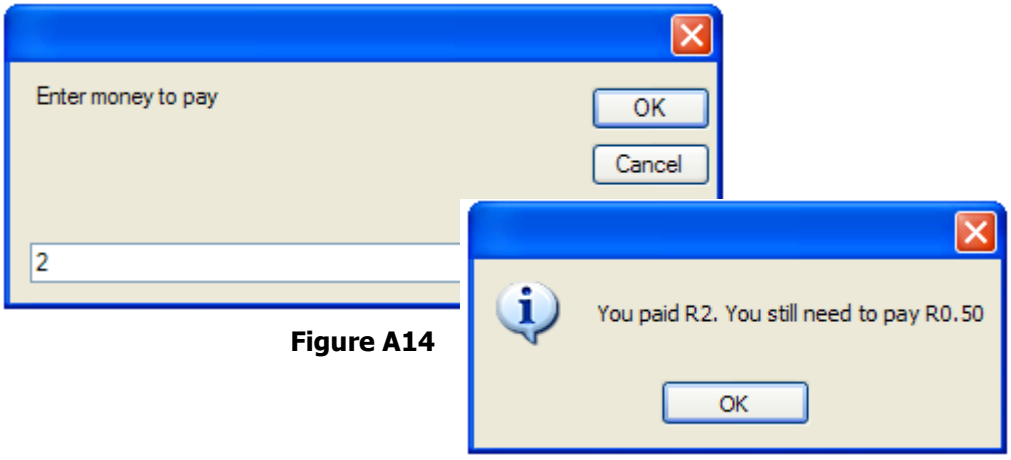


Figure A14

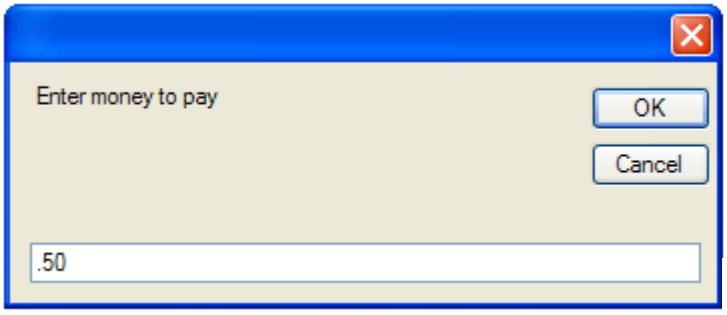


Figure A15


Secure parking at the Mall
Enter the number of minutes that you parked:
80
You parked for 1 hours and 20 minutes
Amount due for parking = R2.50
Amount paid = R2.50
Change = R0.00
Calculate amount and display receipt
Next

Figure A16

APPENDIX B
UNIFIED MODELLING LANGUAGE

TAKE NOTE: The authors suggest that these notes are studied in conjunction with the prescribed textbooks section on UML.

1. INTRODUCTION

In previous sections of these notes you have been introduced to various techniques for documenting the flow and logic for a specific problem's solution.

The Unified Modelling Language (UML) is a symbolic language that implements symbols to depict and describe how certain systems behave. The OMG group (http://www.uml.org/ 2008) defines UML by stating that:

> *"The Unified Modelling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artefacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."*

UML consist of various types of diagrams that could be used to represent systems behaviour as well as its functioning. These diagrams include amongst others:

- Class Diagrams
- Use Case Diagrams
- Sequence Diagrams
- Interaction Diagrams
- State Diagrams
- Component Diagrams
- Deployment Diagrams

As part of this course we will only cover the first three diagrams as listed above.

2. CLASS DIAGRAMS

Class diagrams are used to depict the various classes used within a system. Classes and objects form part of a programming paradigm called OOD (Object Oriented Design) which includes OOP (Object Oriented Programming).

NOTE: VB.NET is a true OOP language which conforms to OOD principles. OOP is a programming methodology that mimics the interaction and behaviour between various real-world objects. OOP as a programming methodology is covered within your normal prescribed textbook.

Class diagrams are used to represent objects in terms of its attributes and behaviour (methods). The diagram below depicts the three sections of a class diagram. The object's attributes represents values or other objects that are used to describe the new object. The objects methods represents the actions or tasks that may be performed by the object.

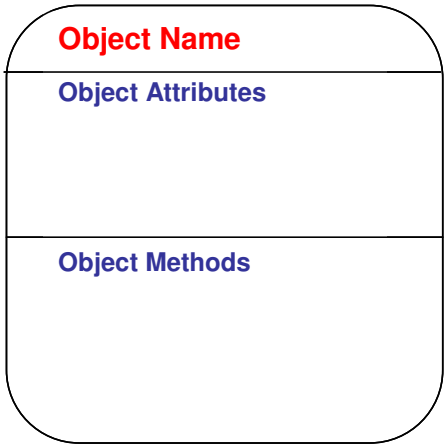


Figure B1

Example of a class diagram

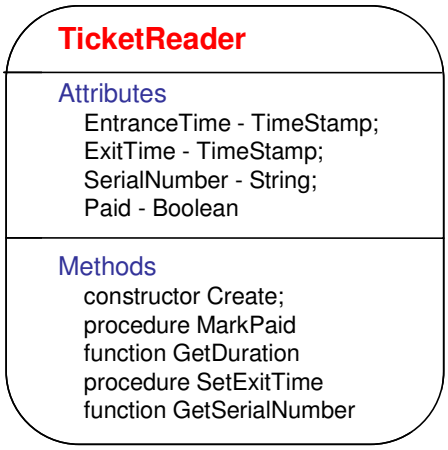


Figure B2

3. USE CASES AND USE CASE DIAGRAMS

A use case diagram represents the uses of a specific solution. In many solutions actors play an important role. An actor is a person or entity that interacts with the system. Actors are represented in UML using a symbol as given below. A use case represents the specific steps or processes in a certain solution. Each use case could also represent a single task. A use case task or process is represented in oval shapes.

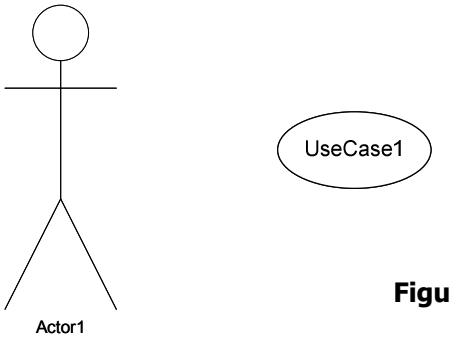


Figure B3

4. A USE CASE DIAGRAM

The Use Case diagram example below depicts the various use cases of a user (customer) paying a parking ticket at a parking vendor machine. (refer to the integrated in Appendix A).

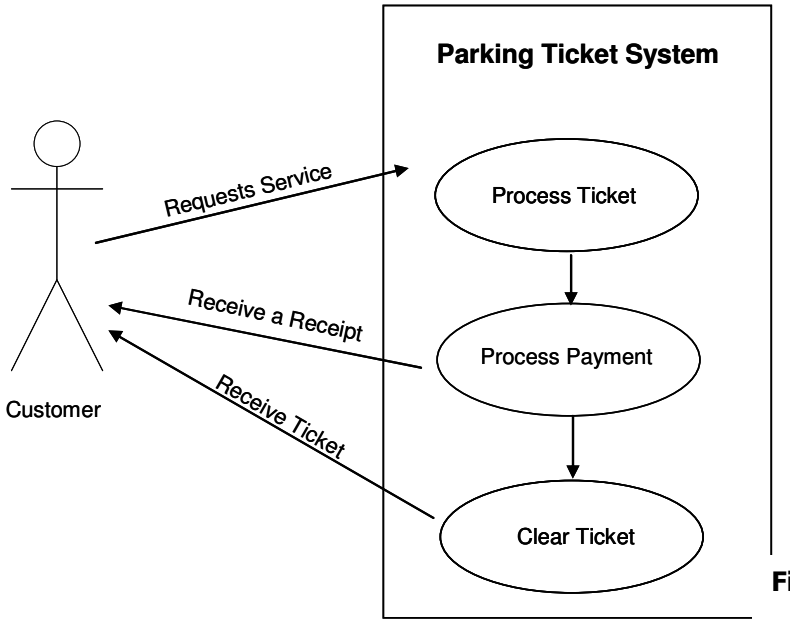


Figure B4

The rectangle is used to indicate what use cases forms part of the systems boundary.

The Process Ticket use case could represent a number of steps or a single step in the system.

5. SEQUENCE DIAGRAMS

A sequence diagram depicts the interaction between the various objects and actors in the sequence in which a process or solution may take place. Where a use case diagram depicts the various use cases (uses) of the system a sequence diagram are used to depict the sequence of processes and the interaction between the objects and actors to perform a specific task. The duration of each of the processes are also indicated.

In the example sequence diagram below a customer (Actor) interacts with a parking ticket system. The ticket system needs to interact with a ticket reader in order to obtain the duration of the stay in a parking bay.

Sequence diagrams are made up of the following symbols:

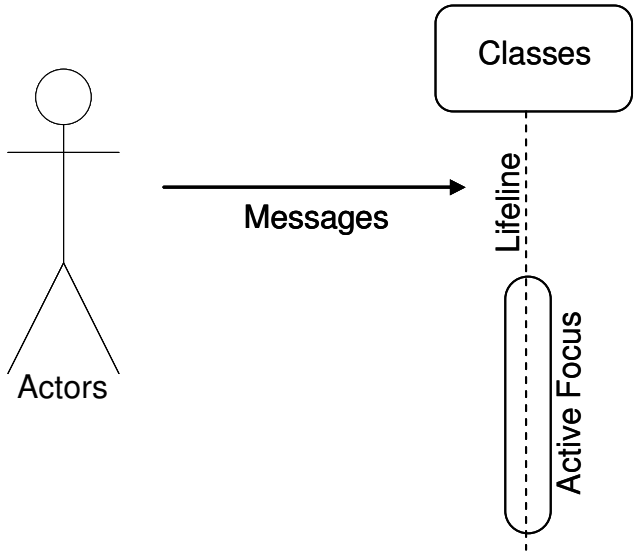


Figure B5

The following sequence diagram could depict the interaction between the customer (actor) with the Ticket System.

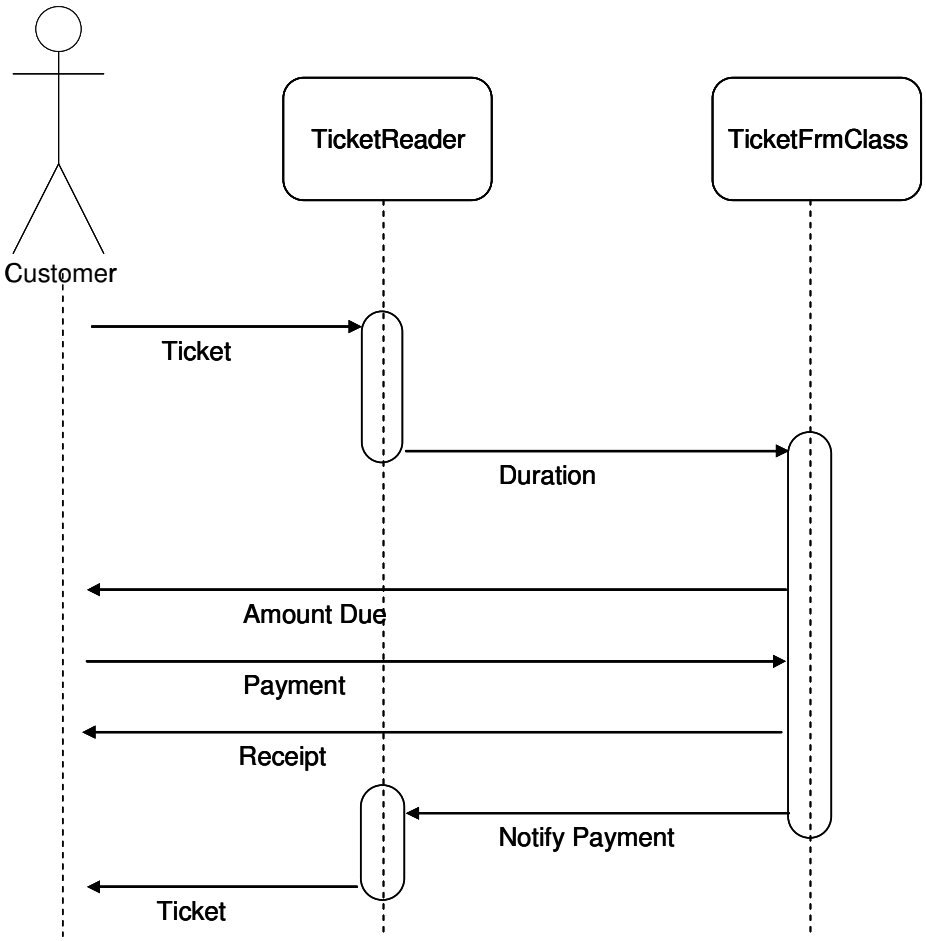


Figure B6

The customer(actor) will insert the parking ticket into the ticket reader the ticket reader will send the duration to the Formclass and activate the process to calculate the amount due. The customer will be prompted for the payment and pay for the parking. A receipt will be printed and the notification of the payment will be sent to the ticket reader. The ticket reader would clear the ticket and return the ticket to the customer.

BIBLIOGRAPHY

BURROWS WE, LANGFORD JD: 2003, Learning Programming using Visual Basic.NET, McGraw-Hill, ISBN 0072451963.

ERASMUS HG, FOURIE M, PRETORIUS CM: 2002, Basic Programming Principles, Heinemann, ISBN 0796202222.

FARRELL, JOYCE: 2004, Programming Logic and Design, Thomson Course Technology, ISBN 061926913.

KONEMAN, PHILIP A: 2004, Visual Basic.NET, Programming for Business, Pearson: Prentice Hall, ISBN 0130473685.

OMG Group 2008 [Online] Available at http://www.uml.org/ Accessed [22/04/2008]

ROBERTSON, LA: 2004, Simple Program Design, Fouth Edition, Thomson Course Technology, ISBN 0619160462.

SHELLY, G.B. CASHMAN, T.J. ROSENBLATT, H.J. (2006) *Systems Analysis and Design 6th Edition*. USA: Thompson- Course Technology

TSAY, JEFFREY J: 2004, Visual Basic.NET Programming, Business Applications with a Design Perspective (2nd Edition), Pearson: Prentice Hall, ISBN 0130094218.

WIKIPEDIA. 2008. From Wikipedia, the free encyclopedia - Nassi-Shneiderman diagram [Online] Available at: http://en.wikipedia.org/wiki/Nassi-Shneiderman_diagram Accessed [22/04/2008]

ZAK, DIANE: 2002, Programming with Microsoft Visual Basic.NET, Thomson Course Technology, ISBN 0619016620.

ZAK, DIANE: 2007, Microsoft Visual Basic 2005 RELOADED (2nd Edition), Thomson Course Technology, ISBN 1418836230.