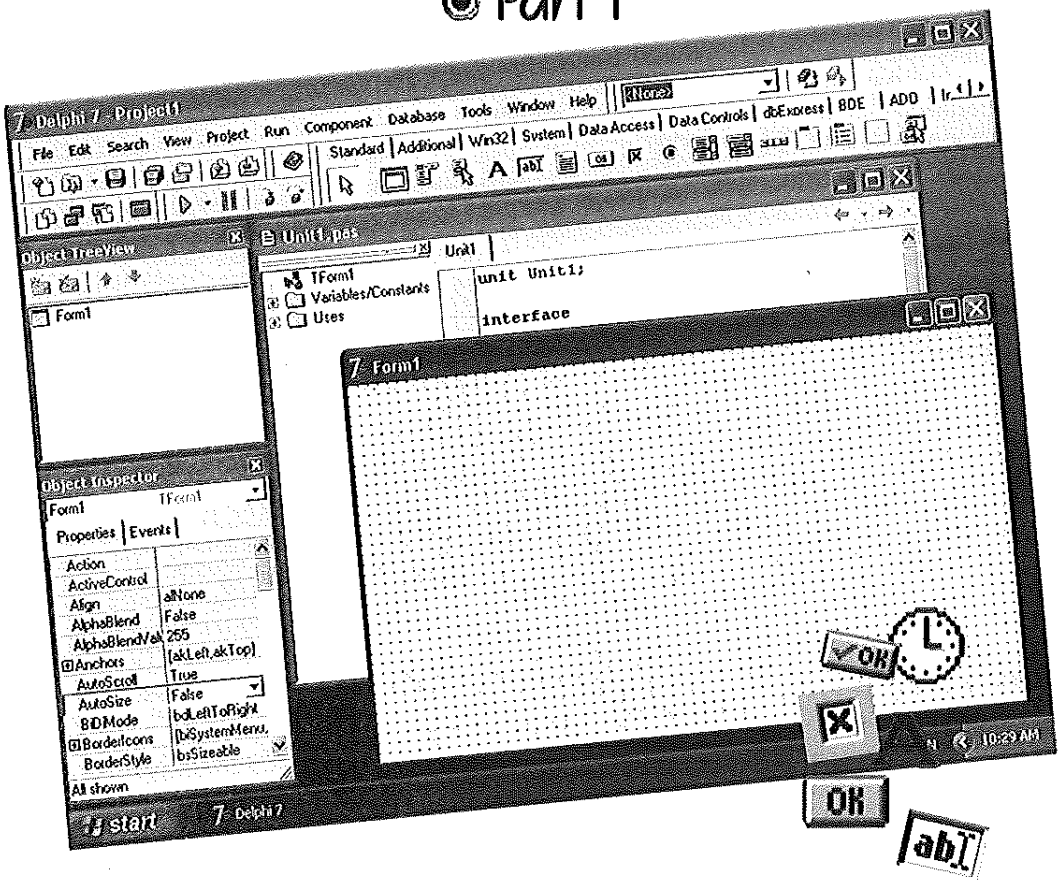
Enjoy Delphi

- A Guide to Problem Solving and Programming -

Part 1



Annette Bezuidenhout Keith Gibson Chris Noomè Malle Zeeman

Study OPPORTUNITIES

Published by Study Opportunities
PO Box 53000, Dorandia, 0188
Tel/Fax: (012) 565-6469
E-mail: study.opp@mweb.co.za

First edition - February 2004
Second impression - November 2004

ISBN 1-919867-36-8

Editor: Sandra Jacobs

DTP, layout and design by Ronelle Stoltz

Printed by MC Printers

Preface

Try to imagine your life without a computer in it somewhere. We play games on them, watch the special effects they create in movies and on TV, use them to browse for information on the internet, send messages using e-mail, store and edit photos – and so much more.

Using programs and accomplishing difficult tasks with the click of a mouse has become commonplace – something we are so used to doing that we can get it right in our sleep!

Now try to imagine just how wonderful it must feel to be the person who creates and writes the programs that allow hundreds and thousands of people to enrich their lives and increase their productivity. Just think how it must feel to create something that can be so useful – or bring so much fun and pleasure to others!

By learning to program with Delphi you will be writing programs that interact with users from the start. You will learn to create programs that make full use of the environment that most computer users work in today – i.e. Windows.

The programs that you write will look and feel like those that you buy in the shops – or download over the internet. You will learn how to think logically and solve problems - AND you will learn all the basic techniques and principles of good programming.

To make the most of this book we would like to encourage you to...

EXPLORE!

BE ADVENTUROUS!

You learn best how to program or use a computer by exploring – trying things out. ...

Being scared of the computer will get you nowhere fast!

The worst that can happen is that you can loose the data / work that you have done – and the solution to that is simply to do it again.

Sure, it's extra work – but it's also extra practice, and you probably won't make that mistake again!

This book also includes an appendix which helps you through aspects of program design and layout which you will find useful in helping you to create professional looking programs. PLEASE refer to Appendix A often – and make it a priority to learn and understand the work it covers as well as the normal chapters.

Acknowledgements

The authors would like to thank

Bertie Buitendag

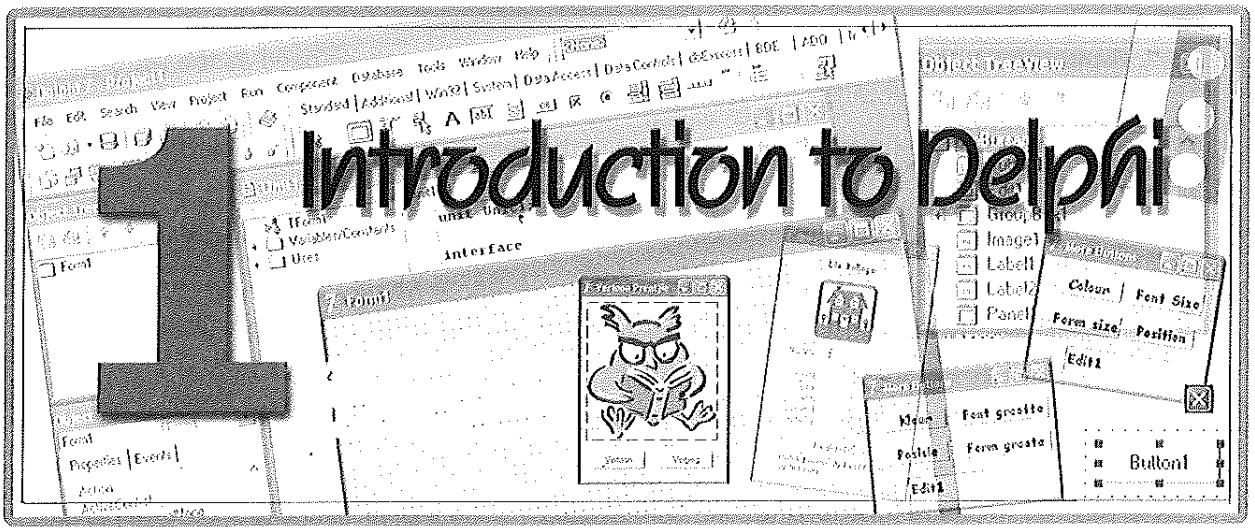
Aletta Forster

Wessel Jacobs

Ulza Wassermann

for their contributions to this book.

Contents



1 Introduction to Delphi

After you have completed this chapter, you should be able to

- use the basic terminology of Delphi
- identify the different parts of the Delphi Integrated Development Environment
- start a new Delphi application
- write and execute a simple Delphi program
- save a Delphi program
- explain what a syntax error is

Programming, programs and programming languages

Computers are perceived as almost mystical, powerful machines that can do almost anything that one can imagine. People are in awe of them. They think that they are incredibly complex and difficult to use, work with and to control. To some extent they are right.

What computers really are, however, is incredibly, moronically, indefatigably ***stupid!*** They are the dumbest things on the face of the planet (sometimes though, they are only marginally dumber than the people who use them). All that any computer can ever do is follow instructions quickly, without getting bored or tired. Anything that a computer can do has been worked out, planned and written for it as sets of instructions – often millions of lines of instructions – called a program.

The computer never, for one instant, deviates from these instructions. It blindly accepts everything it is told to do and is completely and utterly helpless whenever it is faced with something new or unexpected. It can't even remember how to find or even follow the instructions it is given – it has to have another set of instructions that tells it how to do that!

Remember: The computer's operation and 'intelligence' is only as good as the set of instructions it is following.

Programming is the art of explaining to the dumbest thing on the planet how to do whatever we want it to do.

A ***program*** is a series of instructions, written in a language understood by the computer and which tells it exactly what to do.

A ***programming language*** is nothing more than the tool we use to accomplish this task.

This year we will learn how to write our own programs using Delphi. While this book will teach you some aspects of Delphi's syntax and structures, its main focus is on developing an understanding of how computers work, thinking skills and solving problems.

What is Delphi?

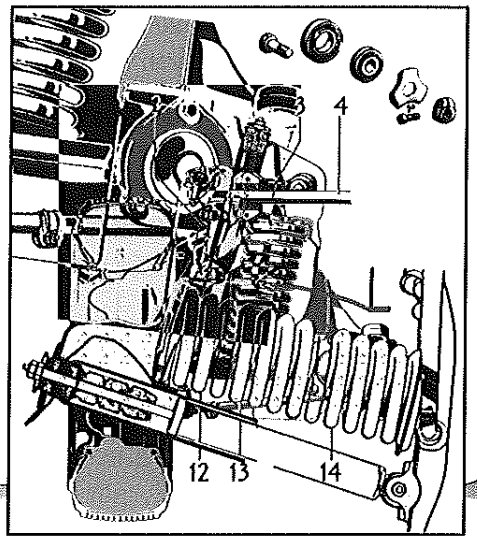
Delphi is a visual programming environment, designed for Rapid Application Development (RAD).

- A *Visual Programming Language* allows programmers to develop programs for the Windows operating system.
- *RAD* tools allow programmers to develop programs in a fraction of the time it used to take in the past. This is because existing, pre-designed components are implemented to develop new programs.

Consider the following scenario:

When a motor manufacturing company decides to develop a new car model, it often doesn't start from scratch. Sometimes many of the components (parts) that it uses were designed by other engineers or manufacturers. It may decide to use a trustworthy fuel injector system, maybe also the design of wheels that it has used previously. Some cars today may have different names but they share more than 80% of each other's components, e.g. Mazda Sting and Ford 1300.

Being able to use predesigned components drastically decreases the time it takes designers to design a new car. This is precisely what RAD entails.

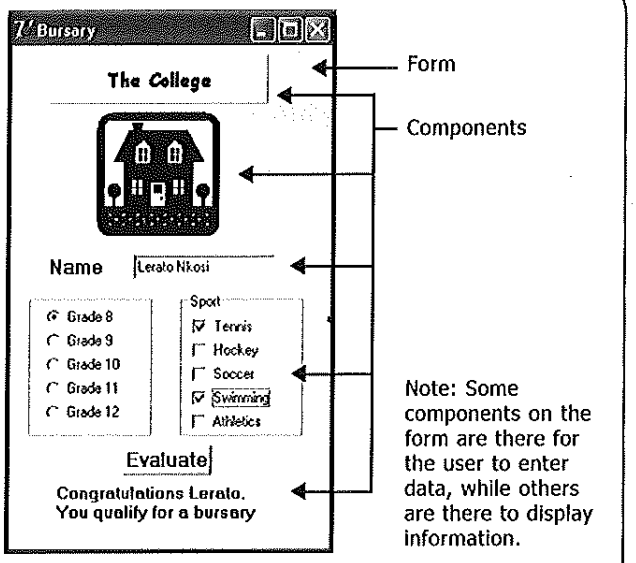


Example of a Delphi application

Consider the following example of an application written in Delphi:

> The College is offering bursaries for prospective learners. Grade 8 and 9 learners need to participate in at least two sports activities to qualify for a bursary while Grade 10 – 12 learners need to participate in at least one sports activity to qualify for a bursary.

The following interface (form) was designed in Delphi and needs to be filled in when a prospective learner arrives. When you run the Delphi program and click on [Evaluate], the program displays a message to indicate if the learner qualifies for a bursary.



In the example, when the program is run the following happens:

The user must supply certain data (Name, Grade and Sports Activities).

An action by the user (in this case clicking with the mouse on [Evaluate]) will result in something being 'triggered'. (In this case a message is displayed to indicate whether the learner qualifies for a bursary).

First acquaintance with Delphi

Before you can start solving problems and programming, you must first create a base to work from. That means that you must first familiarise yourself with the programming environment.

In Delphi the work is carried out in the so-called Integrated Development Environment (IDE). The IDE acts like an artist's canvas, allowing you to visually design the way that the user will interact with your program (i.e. the user interface – what the user will see when your program is run).

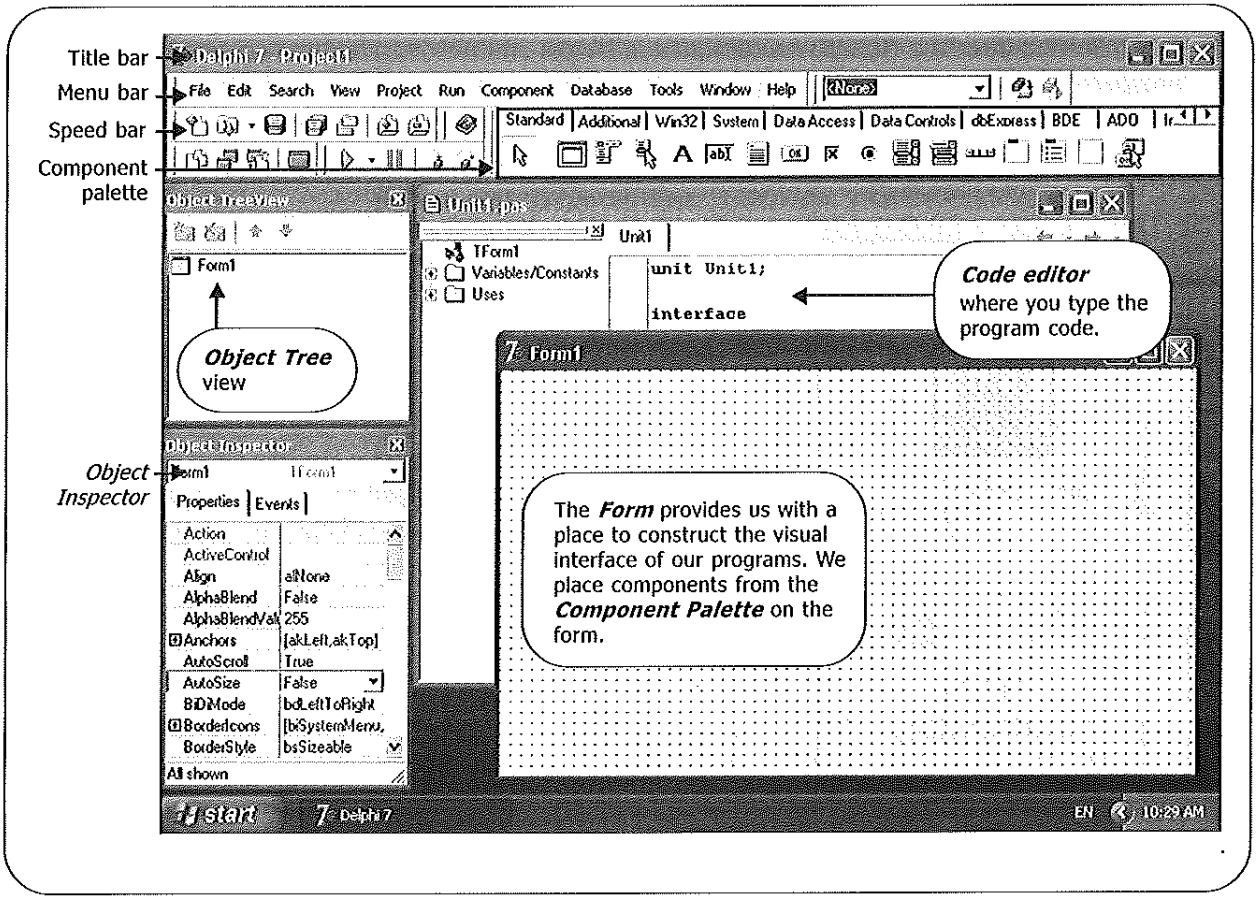
Starting Delphi

Starting Delphi up is as simple as double-clicking the icon on your desktop or on the **start** menu.

If necessary a new application can be created by simply choosing ***File, New Application*** from the drop-down menu.

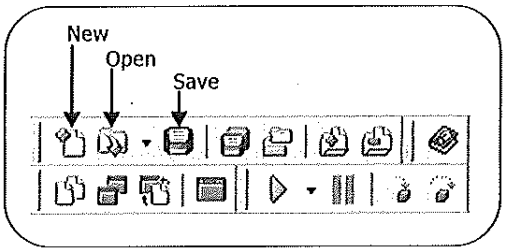
The Delphi programming environment

Once you have started Delphi, you will see the following screen. This screen, which consists of many windows, is referred to as the Delphi IDE (Integrated Development Environment). It contains all the necessary design tools to develop a Delphi Application. A new, blank application is normally created and is ready for you to work on.



The title bar and the menu bar are similar to that found in any Windows application.

The *speed bar* consists of several toolbars and provides quick access to common operations and commands. It saves the programmer the time of having to go to the menu all the time. Move the mouse over the speed bar to see some of the options.



The form

Remember our example of a Delphi application? We saw that the user interface is basically created on a *form* - that is the basis on which other components are placed.

Let's explore the Delphi IDE by looking at what can be done with a *form*.

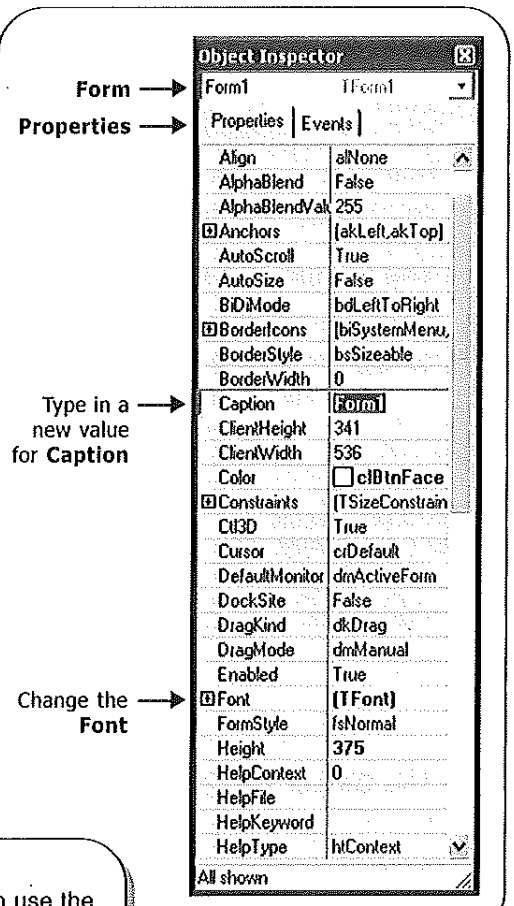
Activity

Exploring the IDE and the form

- Start Delphi.
- Start a new application (***File, New Application***).
- Click on the form.
- Notice that the name **Form1** is now displayed at the top of the **Object Inspector**.
- Click on the **Properties** tab of the **Object Inspector**.
- Look for the **Caption** property and type in a new value.
- See how the *caption* of the form changes.
- Now look for the **Color** (American spelling for an American program) property.
- Change it and watch how the *colour* of the form changes.
- Change the height and the width of the form. (You can do this by changing the values of the properties or by dragging the sides of the form.)

You get the idea..

- Don't save the form.



Tip
If any of the parts of the IDE are not visible you can use the **View** command on the menu bar to display it again.

EXPLORE!

You learn best how to program or use a computer by exploring – trying things out. ...

Change properties just to see what happens, see what the menu commands do, right click everywhere just to see what happens and then **read** the menus that pop up.

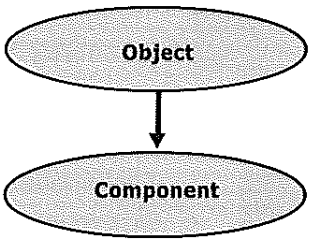
The form has a certain colour, width and height. These are just some of the *properties* or characteristics of the form. Each property has a certain value. These can be changed by manipulating the values in the Object Inspector. This is quite cool – because as you change the values in the Object Inspector, the changes are immediately reflected in the form.

- The form is the basis of the User Interface on which all other components can be placed.
- The form has properties that can be changed in the Object Inspector.

Components

Two of the windows in the Delphi IDE are the *Object* Inspector and the *Object* Tree View. For now you only need to know that forms and other components are all called *objects*.

Components are the building blocks for your application. Delphi has pre-designed components that we will use. Examples of components are Buttons, Labels, Edits, Menus, etc. Just like the form object, components have properties.

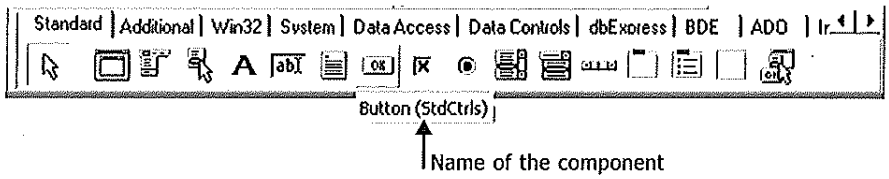


In Delphi we find a hierarchical structure.
All components will therefore also be objects.

The Component Palette

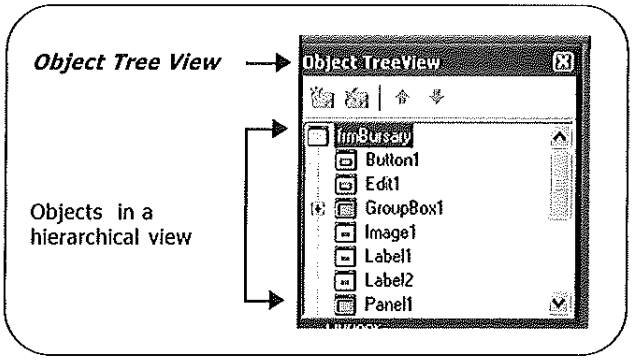
A toolbar called the Component Palette is provided. The component palette is divided into different pages/tabs (Standard, Additional, System, etc.) that contain a library of ready-made components that can be used in your application.

If you hold the mouse pointer over an icon you can see the name of the component.



The Object Tree View

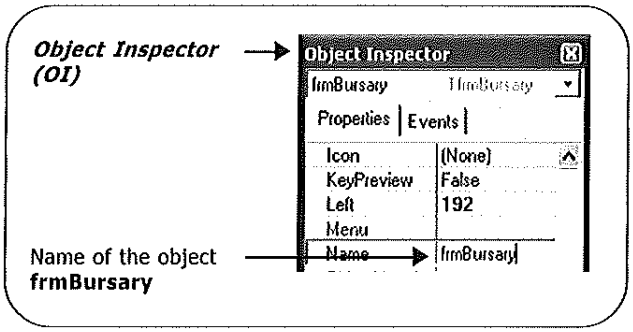
You can see the current objects and their names in a hierarchical view.



The Object Inspector (OI)

The Object Inspector gives you an overview of a particular object's *properties* and *events*.

- An example of a property is the *name* of the object (in this case **frmBursary**). Different objects will have different properties.
- An *event* is an action that is linked to the specific object. More about this when you start writing your own programs!



Note
From now on we will refer to the Object Inspector as the OI.

Exploring components

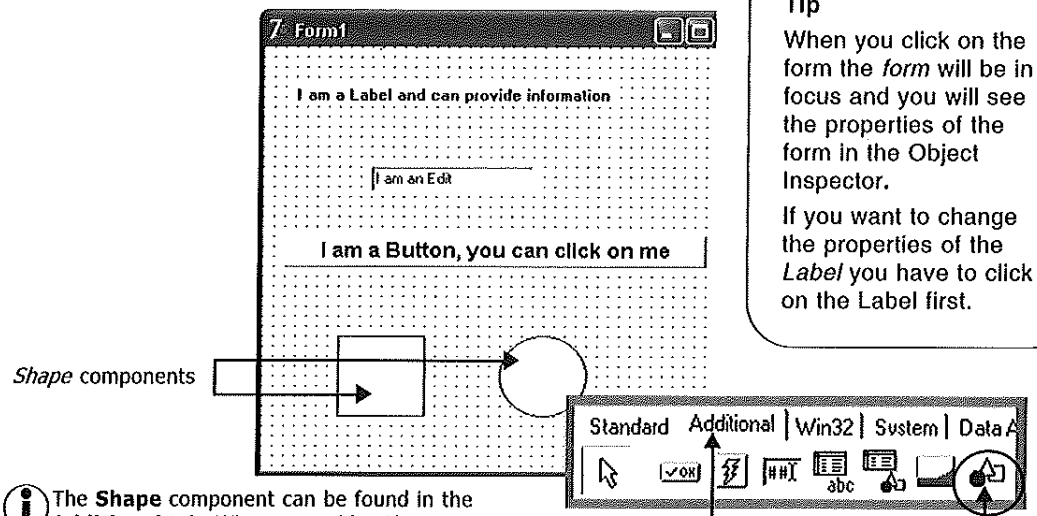
Activity

We are going to *explore different components and their properties* by placing components on the form and changing their properties.

- Start a new application (***File, New Application***).
- See if you can place a Label, Edit and a Button on the form.
- Change the Caption and Font properties of the Label and the Button.
- Change the Text property of the Edit.
- Add some Shape components and change the Shape *property*.

EXPLORE!

The best way to learn about a component is to place it on the form and play with its properties.



Tip

When you click on the form the *form* will be in focus and you will see the properties of the form in the Object Inspector.

If you want to change the properties of the *Label* you have to click on the Label first.

The **Shape** component can be found in the **Additional** tab. When you add a Shape component, a square appears on the form. By changing the *shape property* of the Shape component, the square can be changed to a circle, rectangle, etc.

Tip

To place a component on a form click on the component selected from the Component Palette and then simply click on the form.

You can also click on components and drag them around the form to change their position.

You can resize them by clicking and dragging the handles (the black boxes).

To remove a component just click on it and press <Delete>.

NOTE: At no stage in this book will you be taught about all the components. You can, however, feel free to experiment and play with them because that is the best way to learn about what they can do.

Practical example of a program that actually does something

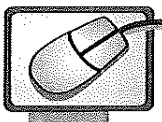
Up to now we have played around with the form and components but have not actually written a working program. Because a computer is really dumb you have to give it very clear instructions on what to do and when to do it!

Nothing can happen in a Delphi application if an action *(event)* has not been triggered.

The most obvious actions by the user that will trigger an event that a computer can respond to, are those generated by the user's interaction with the computer. For example

- clicking a mouse button
- moving the mouse
- pressing a key on the keyboard.

These *events* indicate *when* something needs to happen in a Delphi application. In the following activity you will learn how you will tell the application *what* needs to happen.



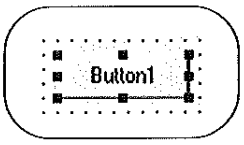
Adding code

A few very important concepts are illustrated in the example. Make sure you recognise and understand them.

Activity

We are going to place a Button on a form and add code to make the following happen when you click on the Button:

- the colour, caption, height and width of the form must change
- the Button must disappear.



Create an interface for your program

- Start a new application.
- Place a Button on the form.
- Change the caption of the Button to [Click on me] and display it in a legible font.

We now have to write instructions that will be executed automatically when the user clicks on the *Button*.

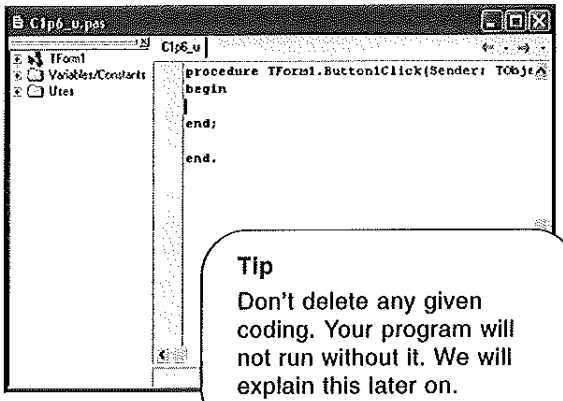
- Double-click the Button.

The *Code Editor* window now appear with the skeleton code.

- Type the following code between the **begin** and the **end;**

```
Form1.Caption := 'Change to aqua';
Form1.Height := 200;
Form1.Width := 200;
Form1.Color := clAqua;
Button1.Visible := False;
```

Code Editor



Tip

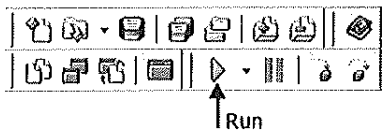
Don't delete any given coding. Your program will not run without it. We will explain this later on.

Tip

Use <F12> to toggle between the *form* and the *unit*. (Don't do it while the program is running.)

Compile and run the program

- Click the Run icon.
- Now click the Button on the form and see what happens.
- Click in the top right-hand corner to stop the program.
  (This will bring you back to the design view where you worked initially.)



TIP

To run a program you can

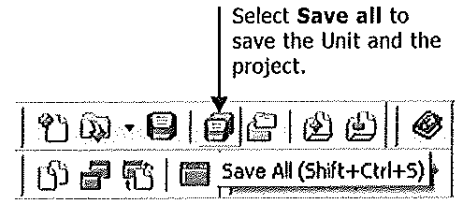
- select **Run** from the Run menu at the top of the screen
- press the F9 function key
- click on the green Run icon on the speed bar.

Delphi is a strictly defined programming language and its rules must be followed to the letter. If you do not type the code exactly as indicated your program will not run.

Save the program

- Create a folder where you want to store all your Delphi applications.
- If necessary, close the program before saving it. (You will not be able to save your program while it is still running.)

Select **Save All** to save the Unit and the project.



- Save your Unit as **c1MyFirst_u** and the project as **c1MyFirst_p**.

Explanation

A link is necessary between the *event* and the code that needs to be executed when the event takes place. In the activity you had to double-click on the component to link the event with the code. If you just added the code yourself, the program would not have worked.

Delphi provides a framework (skeleton) that needs to be completed. The skeleton code and the code added by the programmer, are called the ***Event handler***.

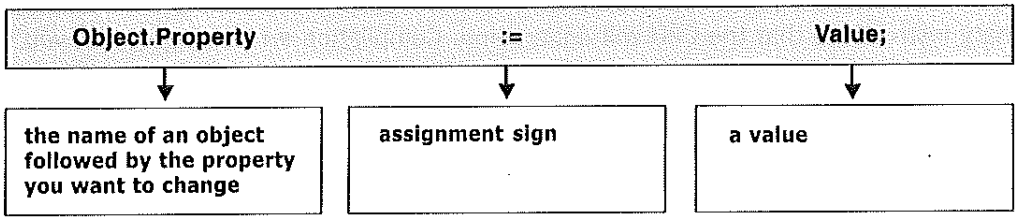
Look at the following statements again:

```
Form1.Caption := 'Change to aqua';
Form1.Height := 200;
Form1.Width := 200;
Form1.Color := clAqua;
Button1.Visible := False;
```

Assignment statements are used to assign values to the properties of the form.

This statement makes the Button 'disappear'.

These statements all have the following structure:



Delphi programs

By default the Unit files are called Unit1, Unit2 etc. and the Projects are called Project1, Project2, etc.

During the year you will write many programs and save them to disk. Keeping track of all these files could become quite difficult.

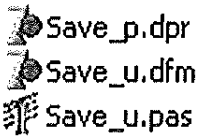
We are therefore using the following naming convention:

- For problems at the end of the chapter: the chapter and the problem's number
- For examples and activities in the chapter: the chapter's number and a name describing the problem

We add a **_u** for a **Unit** and a **_p** for a **Project**.

Delphi creates eight files when you save and run your program. The following three files however are the only ones you need to copy when you want to transfer your work to another computer. Delphi will recreate the other files when you run the program again.

- The project file (.dpr)
- The form file (.dfm)
- The unit file (.pas)



Syntax errors

Languages have rules that specify how the language can be written or spoken so that people can understand what is being communicated. These rules control all sorts of aspects of the language – how we use punctuation, the order of the words in a sentence, etc. For example in spelling in English we have rules such as 'i before e, except after c' and we have rules that tell us a '?' at the end of the sentence means a question is being asked.

Syntax is a word which basically includes all the spelling, grammar and punctuation rules of a language.

Just like human languages, computer languages (like Delphi or any other programming languages) also have syntax rules.

People can think – so when we make a mistake in communicating and break the rules of the language, the person we are communicating with can often guess at what we mean. Remember that *computers* are stupid. They can't guess at our meaning if we break the rules of syntax. They just get confused, stop what they are doing and show us where the mistake is. They cannot go any further until we fix the mistake.

That is why Delphi syntax is very strict. If you break the rules of the language the compiler (the program that translates the Delphi code into binary computer code) will not accept the program.

When you compile a program, Delphi takes the code you have written and translates it into binary instructions that the CPU will understand. Compilation errors occur when you have broken one of the rules of the language – such as forgetting a ';' at the end of a line. If a compilation error should occur whilst translating in the IDE, Delphi will activate the *Edit Window* and position the cursor at the error in the source code. Fortunately, error messages are displayed to help you correct syntax errors. The message is, unluckily, not always self-explanatory.

Activity

Dealing with syntax errors

To open an existing project you select ***File, Open Project*** and then look for the project in the correct directory. The following icon is used for projects:

Lets make some errors to see what Delphi does:

- Open **c1MyFirst_p.**
- Delete all the code after the begin of the procedure and enter the following code:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1 := Change to aqua;
  Form1.Height := 200
  Form1.Width = 200;
  Form1.Colour := Aqua;
  Button.Visible := False;
end;
```

Add a space between the : and the =

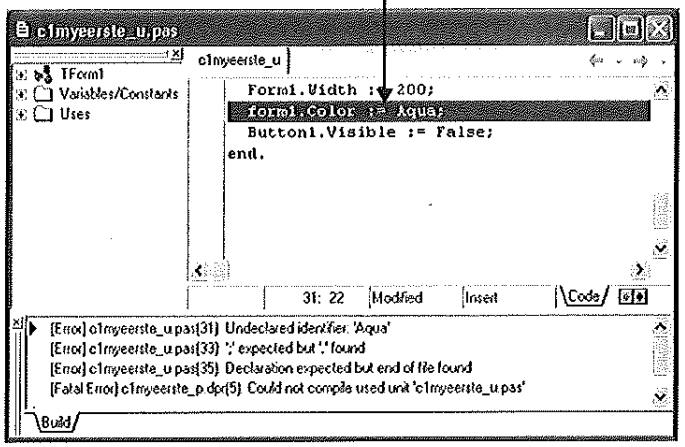
- Run the program.

A maroon line will appear on the first line of code with an error. The message **Undeclared identifier 'Change'** appears.

The reason for this that **Change to aqua** should have been between inverted commas.

Correct the error and run the program again.

A maroon line will appear on the first line of code.



A whole list of errors will now appear:

[Error] c1myfirst_u.pas(28): Incompatible types: 'TForm1' and 'String'
[Error] c1myfirst_u.pas(30): Missing operator or semicolon
[Error] c1myfirst_u.pas(30): ':=' expected but '=' found
[Error] c1myfirst_u.pas(31): Undeclared identifier: 'Colour'
[Error] c1myfirst_u.pas(32): Undeclared identifier: 'Button'
[Error] c1myfirst_u.pas(32): ':=' expected but ':' found
[Error] c1myfirst_u.pas(35): Statement expected but end of file found
[Fatal Error] c1myfirst_p.dpr(5): Could not compile used unit 'c1myfirst_u.pas'

- The first error does not make sense at this stage. Look carefully at what you have typed for the first statement.

  Form1 is the name of a component. The **.property** has been left out.

Change the line to

```
Form1.Caption := 'Change to aqua';
```

- The second mistake indicates a missing operator or semicolon. Check the end of all the lines to see where a semicolon has been left out.

  Corrrect the error.
- The next mistake indicates that a ':=' was expected but a '=' found.

  This error message clearly indicates the error. Correct it.
- The next message indicates that the word **Colour** is unknown.

  The reason for this that the property of the form is **Color** spelt the American way. Correct the error.
- The next message also indicates that **Button** is unknown.

  The name of our component was **Button1**. Correct it.
- We then see that the computer found a ':' when it expected a ':='.

  The space between the ':' and the '=' caused this problem. Correct it.
- The next message tells us that the program does not end correcly. All programs must end with a **end.** Add the **end.**

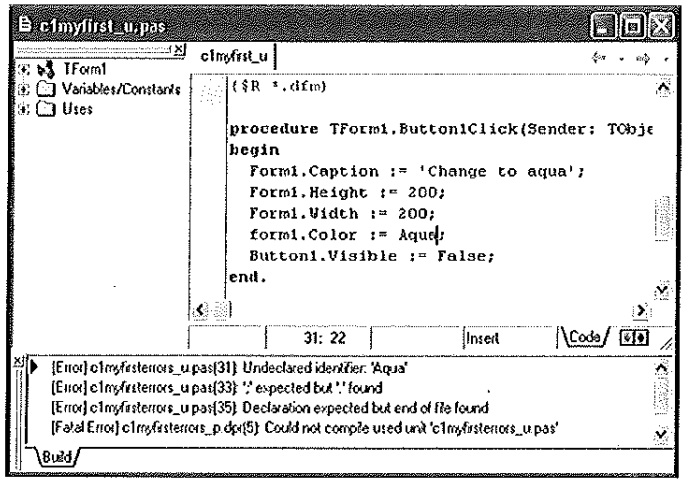
Tip

If you can't see all the errors correct the ones you can and run the program again.

The computer marks the place where it realises that an error has occurred. That may help you although the error may be in a previous line.

Now run your program again.

The next list of errors will now appear:



- The message **Undeclared identifier 'Aqua'** means the computer does not recognise the colour Aqua. Change to **clAqua.**
- **';' expected but '.' found** was a error because the **end;** to end the procedure was missing before the end. That ends the program. Add the **end;**

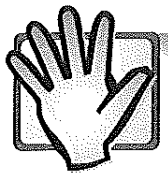
Your program will now compile.

By now we should see that, until we have fixed all the errors, the program will not run. Some error messages clearly indicate the problem, while others are more difficult to correct.

Tip

If the green run-button is grey use ***Run, Program reset*** to enable you to run the program again.

When you write programs on your own, try and fix them yourself before you call the teacher. This way you will learn to work very accurately and will also learn how to interpret error messages yourself.



Checkpoint: Write your own Event handlers

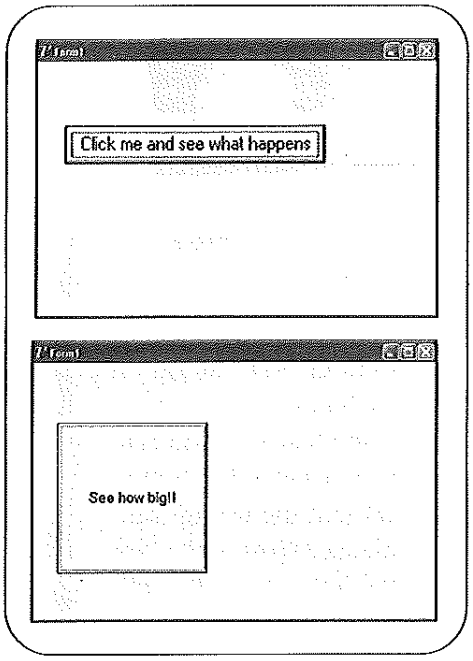
1. Write the correct statement to change the *properties* of the following components as indicated:
   a) Write the code to change the properties of **Form1** as follows:
      (i) Change the height to 200.
      (ii) Change the colour to blue.
      (iii) Change the caption to "Form Colour".
   b) Write the code to change the properties of **btnShow** as follows:
      (i) Change the height to 30.
      (ii) Disable the Button.
      (iii) Change the caption of the Button to "Show".
      (iv) Hide the Button.
   c) (i) Change the caption of **frmSchool** to the name of your school.
      (ii) Change the colour of **frmSchool** to yellow.
      (iii) Change the words on **btnClick** to "Click".
      (iv) Change the words on **lblOutput** to "output".
      (v) Make **frmSchool** "500" wide.

2. Start a new application. Place one Button on the form and change the caption to [Click me and see what happens].

   Now write the Event handler to let the following happen:

   When the user clicks on the Button it should become larger and [See how big!] should appear on the Button in a clear, large font.

   Save the application as **c1Buttonsize** (i.e. the unit as **c1Buttonsize_u** and the project as **c1Buttonsize_p).**



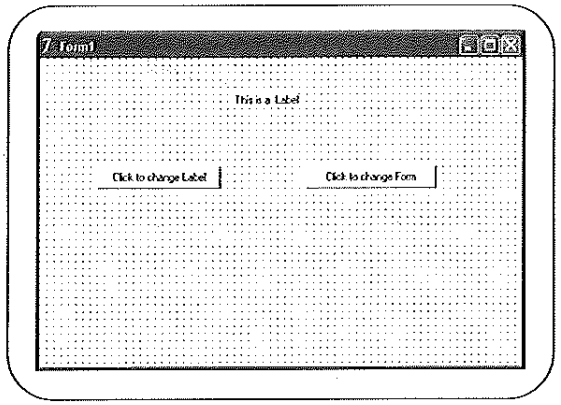
> An alternative method to obtain the framework of the *Event Handler* is the following:
> Click on the component, e.g. the Button; go to the component's *events* in the *Object Inspector*; click on the *OnClick event*; double-click on the white window to the right of the event; the *Event Handler* skeleton will appear.

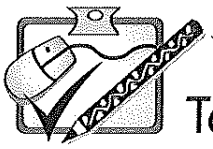
3. Start a new application. Place one Label and two Buttons on the form.

   Write the Event handlers so that the following happens:

   - The Label must turn red and show an appropriate message when the Button on the left is clicked.
   - The form's colour must change when the Button on the right is clicked.

   Save the application as **c1LeftRightClick** (i.e. the unit as **c1LeftRightClick_u** and the project as **c1LeftRightClick_p).**

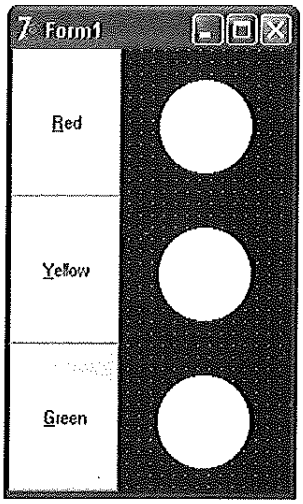




Test, improve and apply

Practical Exercises

1. Imitate a robot



Start a new application.

- Place the indicated components on the form and change the properties in the Object Inspector as follows:

Component	Property	Value
Form1	Color	clBlack
	Height	335
	Width	200
Button1	Caption	&Red
	Left	0
	Top	0
	Height	100
	Width	75
Button2	Caption	&Yellow
	Left	0
	Top	100
	Height	100
	Width	75
Button3	Caption	&Green
	Left	0
	Top	200
	Height	100
	Width	75
Shape1	Shape	stCircle
	Left	100
	Top	20
	Height	65
	Width	65
Shape2	Shape	StCircle
	Left	100
	Top	120
	Height	65
	Width	65
Shape3	Shape	StCircle
	Left	100
	Top	220
	Height	65
	Width	65

Accelerator (Hot) Keys:

By placing an "&" in front of "Red" in the caption we have created a shortcut key. You can now activate the button without the use of the mouse. Press <Alt> and the letter e.g. <R>. (If the button is in focus you only need to press <R>.)

Tip

To change the colour of a shape you need to change the brush property of the shape e.g. Shape1.brush.color:=clRed

To place multiple copies, hold the <Shift> down when you click on the component. Every time you click on the form you will place another instance of the component on the form.

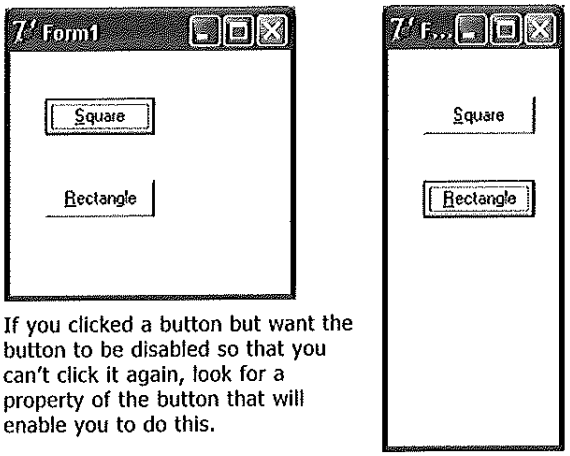
To stop placing components, click the mouse pointer at the start of the palette.

For each of the Buttons write an OnClick Event handler.

- For the Red Button the Event handler must change the colour of **Shape1** to red, for the Yellow Button the Event handler must change the colour of **Shape2** to yellow and for the green Button the Event handler must change the colour of **Shape3** to green.
- Save the file and the project as **c1p1_u** and **c1p1_p**.
- Run and test the program.

2. Change the dimensions of a form

- Start a new application.
- Place two Buttons on the form.
- Change the captions of the Buttons to Square and Rectangle.
- Complete the OnClick Event handlers. The [Square] Button must change the height and the width of the form to 200. The [Rectangle] Button must change the height of the form to 300 and the width to 100.
- After the [Square] Button is clicked, the [Square] Button must be *disabled* and the [Rectangle] *enabled* and vice versa.



If you clicked a button but want the button to be disabled so that you can't click it again, look for a property of the button that will enable you to do this.

Tip

There are three ways to change the dimensions of an object namely:

1. Click on the object and use the handles to drag and resize the object.
2. Change the properties in the Object Inspector.
3. Change the properties through code in the Unit.

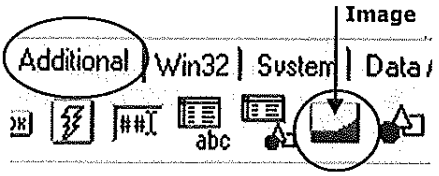
3. Adding a picture to our application

- Start a new application.
- Place two Buttons on the form.
- Place an Image on the form (in the Additional tab).

- Complete an OnClick Event handler to do the following:

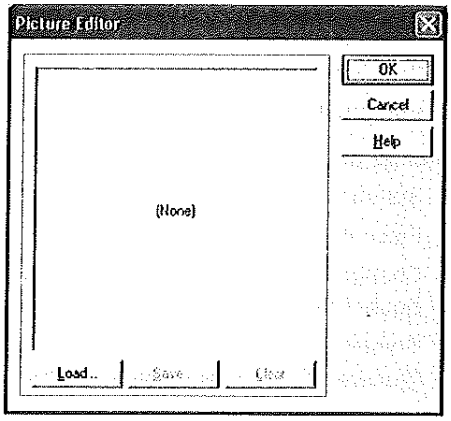
  When Button1 is clicked the Image must disappear and when Button2 is clicked the Image must be visible again. Supply the Buttons with appropriate captions.

To see the whole picture, change the Stretch property of the Image to TRUE.



To insert a picture, click on the three dots next to the picture property of the Image in the Object Inspector.

The following screen will appear:



If you click on **Load** you can choose a picture to insert.

4. More Buttons

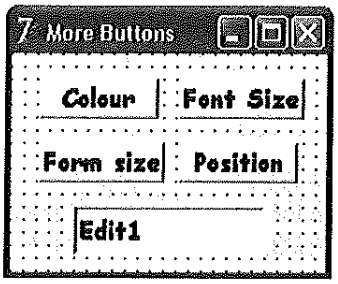
- Place four Buttons and an Edit on your form.
- Change the captions of the form and the components as indicated.
- Change the Font of the form to Comic Sans MS, the Font Style to Bold and the Size to 12.
- Write the following OnClick Event handlers for the Buttons:

  Button1 : Change the colour of the form to SkyBlue.
  : Change the colour of the Edit to MoneyGreen.
  : Change the colour of the font of the Edit to red.

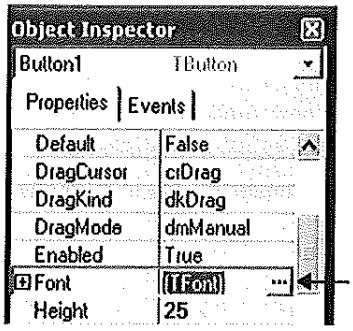
  Button2 : Change the font size of the form to 10.

  Button3 : Change the width of the form to 300.
  : Change the height of the form to 200.

  Button4 : Change the positions of the components on the form.



When you change the font on the form, the font of the objects on the form will also change. This is called ***inheritance***.



Tip
Click on the three dots to sellect **Font, Font Style** and **Size.**

Test yourself

5. Check that you know / can do the following:

		✓
Knowledge	**I can explain in my own words**	
	what a RAD tool is	
	what a Form is	
	what a component is	
	what a property is	
	what an interface is	
	what a Visual Programming Language is	
Skills	**I can**	
	start a new application in Delphi	
	add components such as Buttons, Edits, Labels and Shapes to a form	
	change the properties of objects	

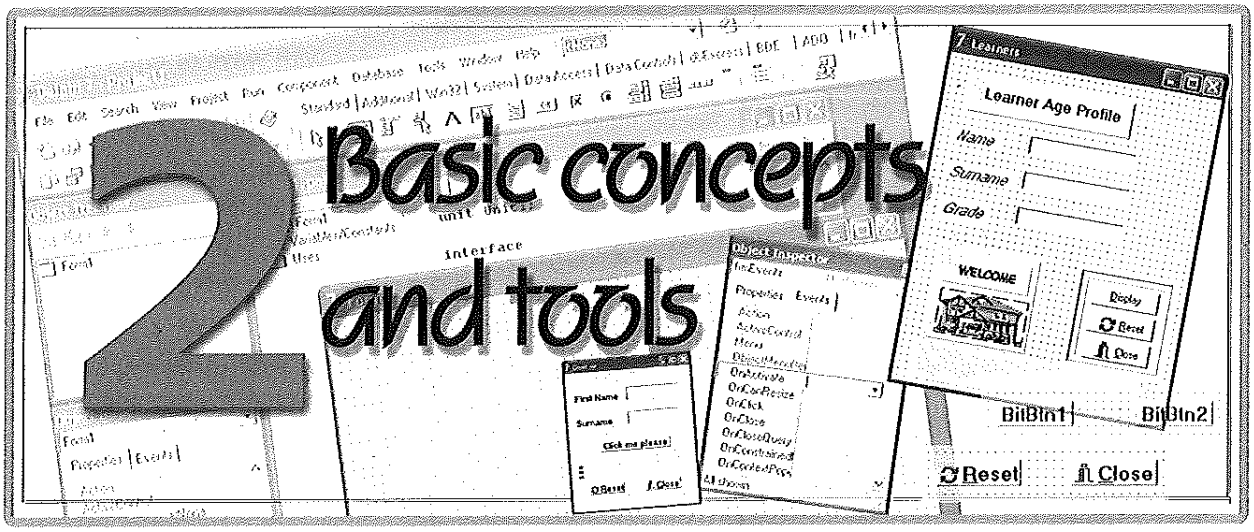
What we have learnt so far

The Delphi program consists of
an ***interface*** (the form and components) as well as
an ***unit*** (code that was written by the programmer).

When a user works with the application,
the *interface* (form with components on it) is seen by the user and
the *unit* allows the interface to re-act/respond to certain commands such as the user clicking on a Button.

We can change the properties of an object in the Object Inspector or in the unit.

- Changing properties during design-time in the OI is called *static* changing of properties.
- Changing properties during run-time (like we have done here) is called *dynamic* changing of properties.



After you have completed this chapter, you should be able to

- describe what input and output is
- make use of basic input and output components such as Labels and Edits in simple Delphi programs
- improve the user-friendliness and appearance of the user interface via the use of components such as BitBtns, the Panel and RichEdit
- use different events
- print your work.

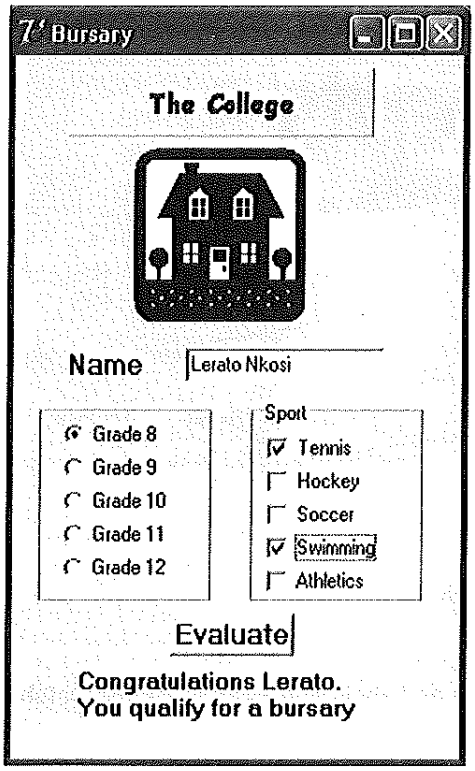
Input and Output

The first thing we have to learn to do efficiently in programming is ***input*** and ***output***. Without input a program has no data to work with. Without output the program serves no purpose – it has to show results in some way and that involves output.

In this chapter we look at the most common aspects of input and output – the Delphi components used to create an interactive interface. This enables the user to input data and receive immediate feedback in terms of visual output.

Let's start with *input*. Programs require input from the user. This input forms the basis for the calculations, decision-making and processing that the program will carry out before it produces output.

Do you remember our very first example of an interface created in Delphi?



The user had to enter certain information. This was the **input** to the program.

When the [Evaluate] Button was clicked, some **processing** was done and the program gave the results or **output.**

This interface provided the users with ways of entering their names and indicating the data that applies to them. They can then click on a Button and immediately see if they qualify for a bursary. This is what is meant by an *interactive interface.*

The first step to take is to learn what the basic input and output components are – and how to use them effectively.

NOTE

This book cannot tell you everything about every component and property that Delphi can offer you. *You* have to take some responsibility, initiative and time to explore and experiment by placing components on the form, changing properties and watching what happens. You also need to think about what is shown to you in the pages below – and how it may apply to and work with components that are not covered in the chapter.

Standard Input / Output components

The most common *input* component is the ***Edit*** (on the Standard page).

The most common *output* component is the ***Label*** (on the Standard page).

We can have as many Edits and Labels on the form as we like – and the users can type as much data into the Edits as they like. Nothing will happen, however, until an *event* is triggered. The most common way to do this is to place a Button on the form and writing some code that will execute (i.e. make something happen) when the user clicks on the Button.

Let's take a closer look at these components:

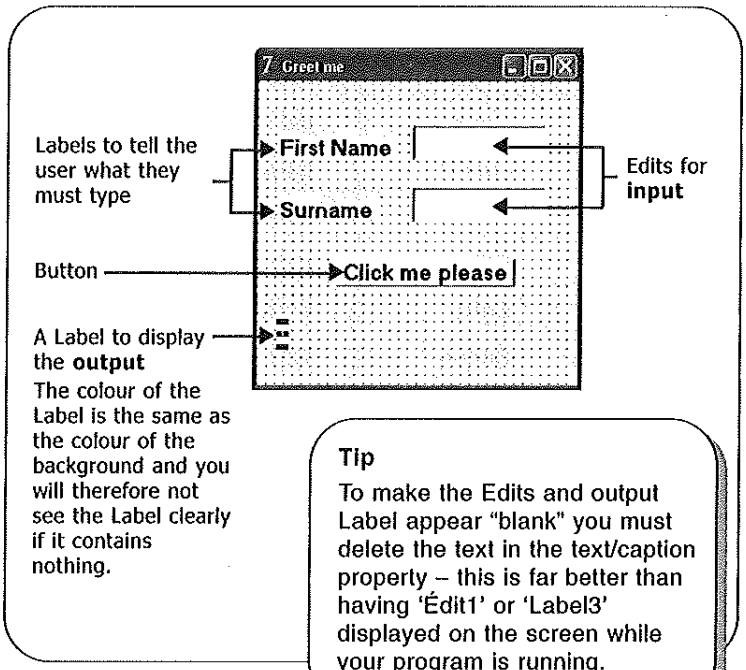
Activity

Exploring Labels, Edits, Buttons

When you are finished with this activity you will be able to

- enter input
- create an output message
- select more than one component and change their properties at the same time
- give components meaningful names
- show that you have a good understanding of events and how to use them.

Start a new application and create a user interface so that you can enter your name and surname. When you click on a Button, the computer must greet you by your name. Use Labels, Edits and a Button as indicated.



Tip

To make the Edits and output Label appear "blank" you must delete the text in the text/caption property – this is far better than having 'Edit1' or 'Label3' displayed on the screen while your program is running.

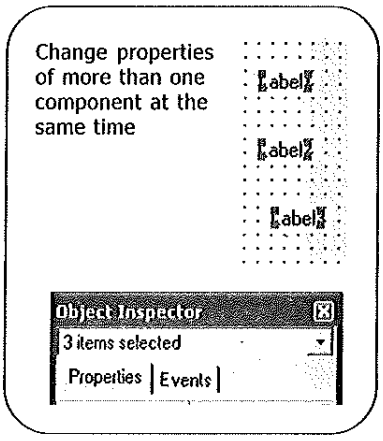
Select and change properties of more than one component at the same time

We often need to set the properties of various components (such as the three Labels) to the same value. For example:

- We want all three *Labels* to be aligned exactly below each other – i.e. the Left property of all three must for example be set to 50.
- The font of all three Labels must be set to for example Arial 12pt, bold.

We can simultaneously select all three by holding down <Shift> when we select the components. If we then change the property in the OI, then that property is changed for all the selected components.

Note – the components you select do not have to be the same, but the OI will reduce its list of properties so that it only shows the properties that all the selected components have in common.



Use meaningful names for components

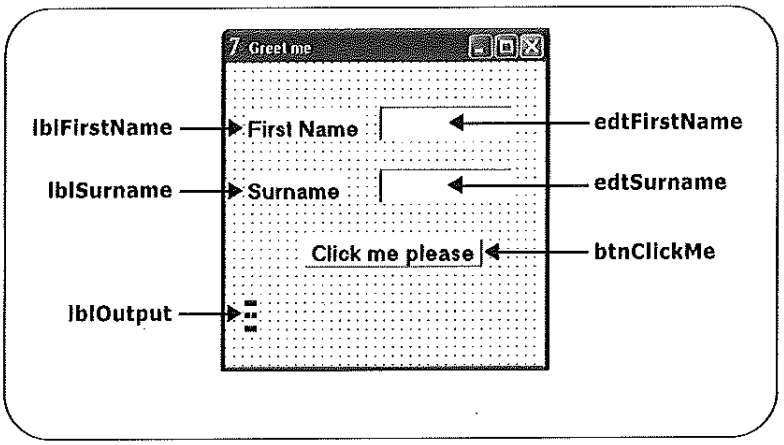
When you place the three Labels on the form, they will automatically be named *Label1*, *Label2* and *Label3* by Delphi. It is difficult to remember what every Label was used for.

By giving the components meaningful names, the programmer not only makes his or her own task easier, it is also easier for any other programmer to understand the program.

Give your components the following names in the OI:

Component	Name
Form	frmGreet
Label1	lblFirstName
Label2	lblSurname
Label3	lblOutput
Edit1	edtFirstName
Edit2	edtSurname
Button	btnClickMe

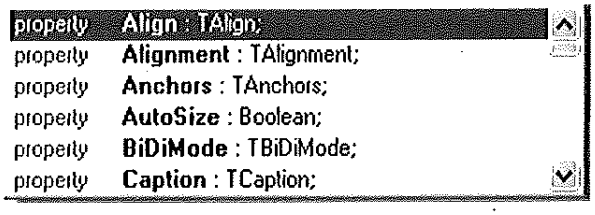
Naming convention:
We are using the Hungarian naming convention.
The name must always have a three-letter prefix to show what kind of component we are using.
A form will get a prefix **frm**, a Label **lbl**, a Button **btn** and an Edit **edt**.



Write the Event handler

- Double-click on the Button and add the following code:

```
procedure TfrmGreet.btnClickMeClick(Sender: TObject);
begin
 lblOutput.Caption := 'Hello ' + edtFirstName.Text + ' ' + edtSurname.Text;
end;
```



Tip
As soon as you type the "." a list of all the properties will appear. You can select the required property and press <Enter>.

Save and run the program

- Save your unit as c2InputOutput_u and the project as c2InputOutput_p.
- Run the program and click the Button [Click me please] to see the result.

Note the following

- A user types input into an Edit. The program obtains the value from the *Text* property of the Edit.
- The assignment statement has a := and follows the following rule: left side *gets*, right side *gives*.
- To use a Label for output we assign a value to the *Caption* property of the Label.
- Text must be placed between single quotes.
- Strings (text) are joined by a "+" sign.

Name property of component	Caption of component
Will be used to *refer to* the component when you write code in the unit	the *words that appear on the component* and can be seen on the form
given in the OI and can not be changed through program code.	can be changed in the OI or in an assignment statement during execution of the program

EXPLORE!

In this chapter you are going to become familiar with Delphi by completing various activities. These will guide you to discover and learn new skills.

The best way to practice and learn is to try to redo each activity on your own without copying the sample code in the book!

In this way, you will be able to apply these skills to other new situations in the future.

Improve the user-friendliness

What is user-friendliness? It is designing your program and its interface so that it is easy to use.

Tips for designing good interfaces:

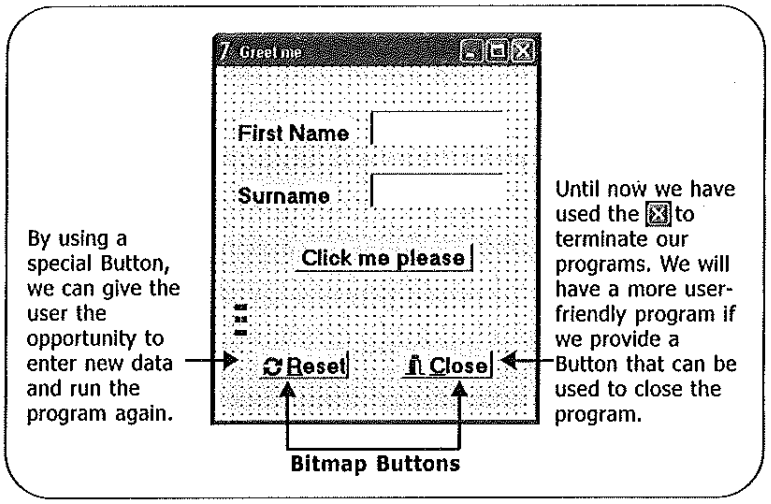
- Use Labels to tell the users what is expected of them.
- Instructions need to be clear.
- The layout of components on the form should be easy to follow.

In this section we will explore a few more methods to improve the user interface.

Bitmap Buttons

Bitmap Buttons (BitBtn) can be used to make our interface more professional.

A BitBtn is a special type of Button that can have a caption as well as a *bitmap* image (picture) on its face. There are a few different types of BitBtns. Some of these Buttons perform certain functions automatically and for others the programmer has to add code.

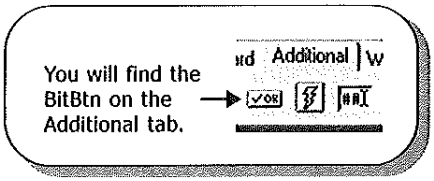


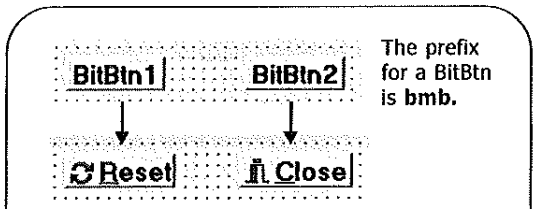
Activity

Place BitBtns on a form

Open the Delphi project c2InputOutput_p and add two BitBtns, one to close the program and one to give the user the opportunity to type in different data.

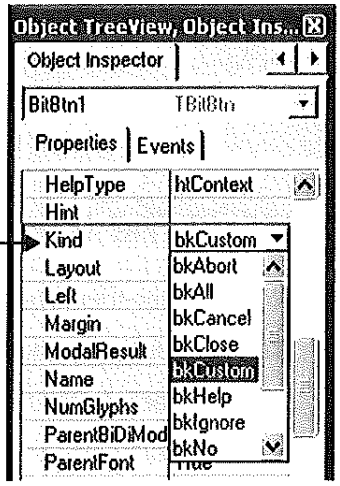
Add the two BitBtns and change the Kind property to **bkRetry** and **bkClose** respectively.





Tip
Use the Retry BitBtn and change the caption to Reset.

Tip
If you click on **Kind** you can see the different types of BitBtns.



Close BitBtn

You do not have to double-click the BitBtn and add code to use it. It has a default behaviour. The program terminates automatically if the user clicks on this BitBtn.

Retry/Reset BitBtn

The Retry BitBtn does not have any default code attached to it.

We want to clear the contents of the Edits and the Label. To enter data again we want the cursor to be in the first Edit box so that the user does not need to use the mouse to move the cursor when he/she enters the next person's details. For this we will program a **Reset** BitBtn.

- Change the caption of the Retry BitBtn to Reset.
- Double-click the Reset BitBtn and add the following code:

```
edtFirstName.Clear;
edtSurname.Clear;
lblOutput.Caption := '';
edtFirstName.SetFocus;
```

This code will clear the Edits and the Label.

This code will move the cursor to the FirstName Edit.

Note the following

Clear and **SetFocus** are examples of ***methods***. A *method* is pre-written code associated with a component that tells the component to perform a specific task.

Unlike the Edit, a Label does not have a **Clear** method. You have to 'clear' the Label yourself by assigning an empty string to the caption of the Label. An empty string is represented by two inverted commas next to one another. (That entails two keystrokes; do not use the double inverted commas.)

Checkpoint: Try the following yourself

Create the following interface as indicated and then write code to achieve the following:

The user must enter data in the Edit. When the user clicks on the [Name] Button, the data entered in the Edit must be displayed in the Label next to the [Name] Button.

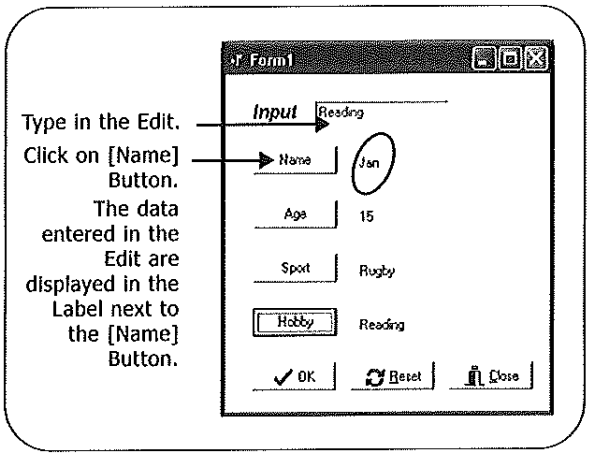
The user then has to click on the OK BitBtn before entering the next data eg. the age. The BitBtn has to be programmed to clear the Edit and place the cursor in the Edit.

When the user enters data and clicks the [Age] Button, the data entered in the Edit must be displayed in the Label next to the [Age] Button, etc.

A click on the Reset BitBtn must clear all the data in the Labels, clear the Edit and return the Focus to the Edit.

The Close BitBtn must close the program.

Save your unit as c2Bmb_u and your project as c2Bmb_p.



The Panel

Panels are used to give your form a more professional look. You can use the BevelInner and BevelOuter properties to change the frame of a Panel. (See the example in the following activity.)

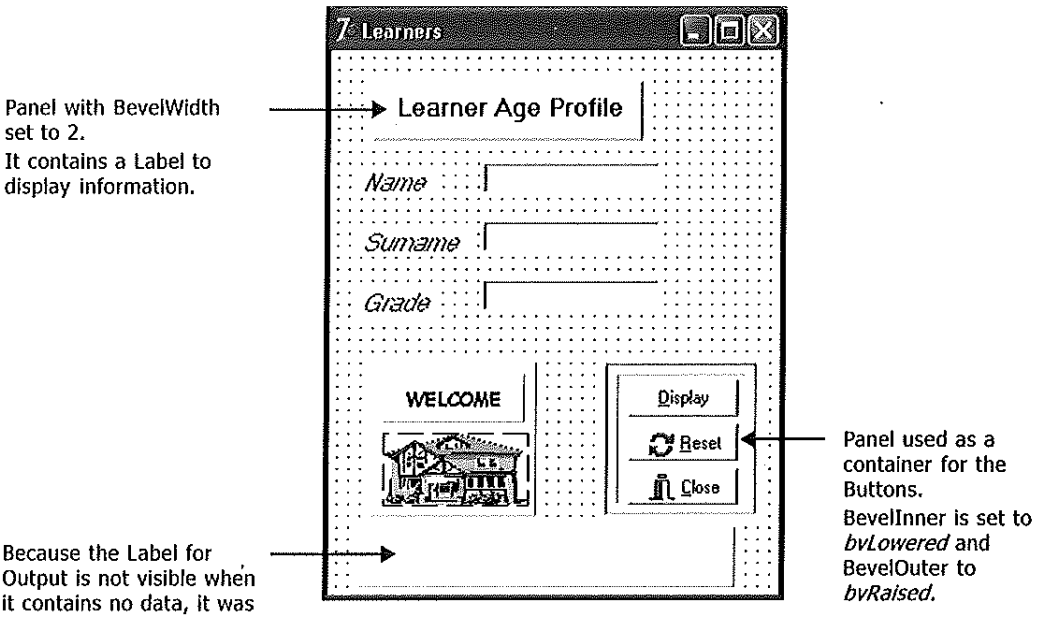
A Panel is often used to visually split up a form. It can be used to act as a container for other components. This means that as the Panel is moved so are the components it contains.

A Panel also has a Caption property that can be used to provide information or for output of text.

Activity

Using Panels

Create the following Interface:



Write an OnClick Event handler for the [Display] Button. The following is an example of the message that needs to be displayed when the user clicks the Button: "Ashley Peters you are registered for Grade 10". Also code the Reset BitBtn appropriately.

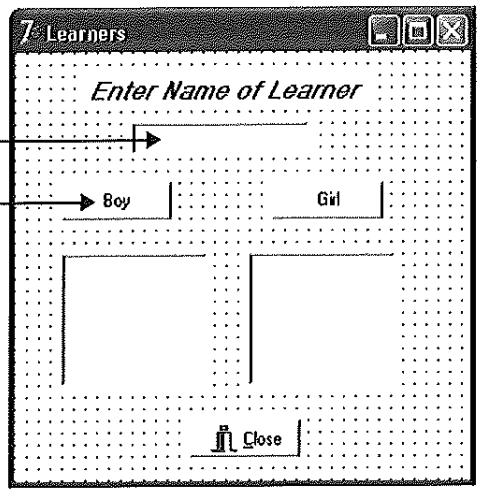
The RichEdit

Sometimes we need to display more information than what a single Label or Panel can contain. The RichEdit component (on the Win 32 tab) is a type of Edit that can handle multiple lines of text. The lines in this control can extend beyond the right boundary of the Edit, or they can wrap onto the next line. (You control whether the lines wrap using the WordWrap property.)

Activity

Exploring the RichEdit

Create the following interface:



When the name of a boy is entered the [Boy] Button must be clicked and the name of the boy added to the RichEdit on the left hand side.
When the name of a girl is entered the [Girl] Button must be clicked and the name of the girl added to the RichEdit on the right hand side.

The abbreviation for a RichEdit is **red.**

Tip

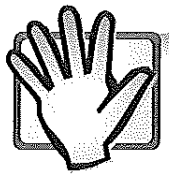
To clear the RichEdit select the Lines property in the OI. When you click on the three dots the Editor will appear. Delete the name of the RichEdit.

The following code must be used to add data to a RichEdit (in this case to redBoy):

```
redBoy.Lines.Add(edtName.Text);
edtName.Clear;
edtName.SetFocus;
```

A useful feature of the RichEdit is the ScrollBars property – change this to ssVertical and you will see a scrollbar appear when the user adds enough names to go off the bottom of the RichEdit.

Also, as it stands the RichEdit is not ideal for displaying the names because the user can simply click anywhere in the RichEdit and change its contents (go ahead and try it). To stop this happening set the RichEdit's ReadOnly property to true. You can still add data with your code but the user will not be able to change it.



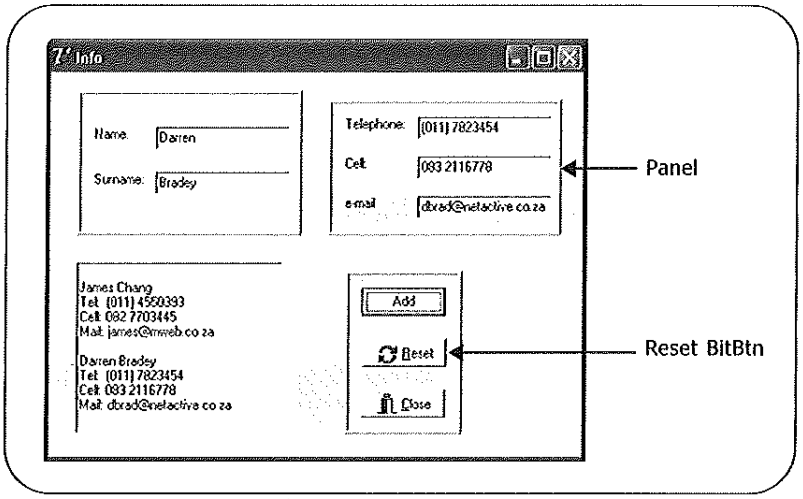
Checkpoint: Try the following yourself

Create the following interface. (This is an example of the form when the program is run.)

Group your components in Panels to give a professional look.

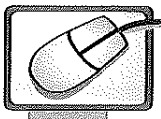
When the user clicks the [Add] Button the information must be displayed in the RichEdit.

The Reset BitBtn must clear the Edits and return the focus to the Name Edit.



More Events

To think about events as just clicking a Button, is limiting. Events can be connected to almost anything on any component. Making intelligent use of the different types of events will make your program richer and more powerful. (Delphi helps us by providing a set of events that each component can respond to. Not all components can respond to the same events.)



Activity

Connecting events to different components

- Open project c2InputOutput_p.
- Take the same code and put it in the *OnChange* event handler of the *edtFirst* component.
- Run the program and you will see that the code happens whenever you type something into the Edit.

Components can also *share* the same event.

- Select the edtSurname component and then go to the **OnChange** event in the OI.
- Instead of double-clicking on the event, click on the arrow and you will see a list of events. Choose the edtFirstChange event.
- If you run the program now will see that the event happens whenever you type in either of the Edits. Now the program works without the user having to click on the Button!

TIP

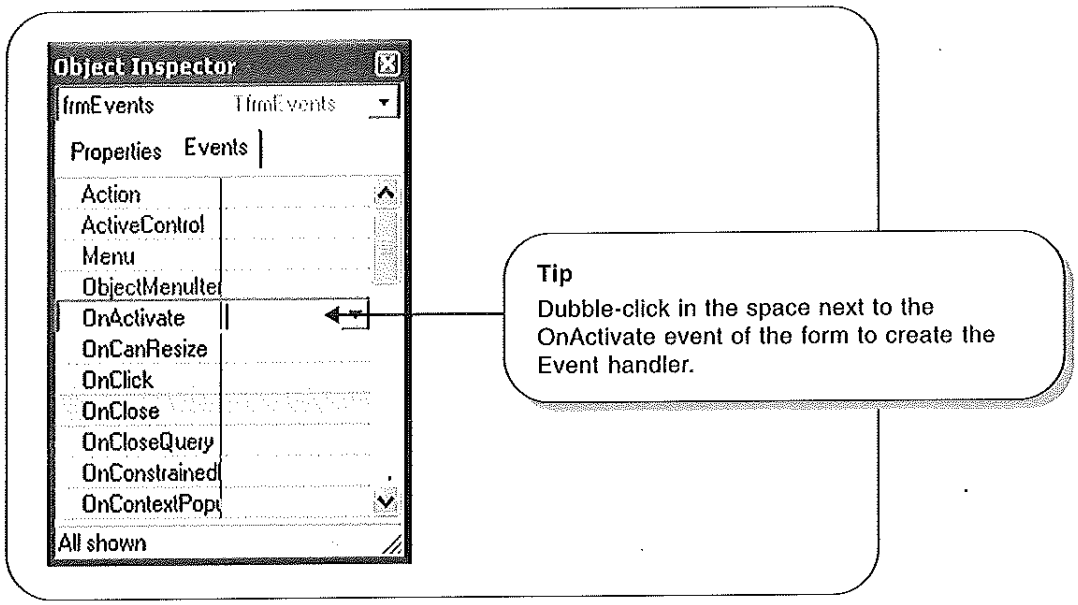
The best way to find out what events a component can respond to is by selecting the component on the form and then looking at the 'Events' tab of the OI. If you do this you will see that each component can respond to far more things than just being clicked on with the mouse.

Activity

Exploring different events

When you want something to happen when the form becomes active, you can use the ***OnActivate*** event of the form.

- Open project c2InputOutput_p if it is not still open.
- Change the Name property of the form to frmEvents.
- Code the Event handler.



Now type the code between the **Begin** and **End:**

```
procedure TfrmEvents.FormActivate(Sender: TObject);
begin
  frmEvents.Color := clSkyBlue;
end;
```

- Run the program.

Also add an ***OnMouseMove*** event. When the user moves the mouse over the form the colour of the form must change to *clLime*.

Add a Button (btnTest) and a Panel to the form and also add an ***OnMouseDown*** Event handler. When the user presses the mouse button the colour of the Panel must change to yellow and the font style of the Button must change to bold.

The statement

```
btnTest.Font.Style := [fsBold];
```

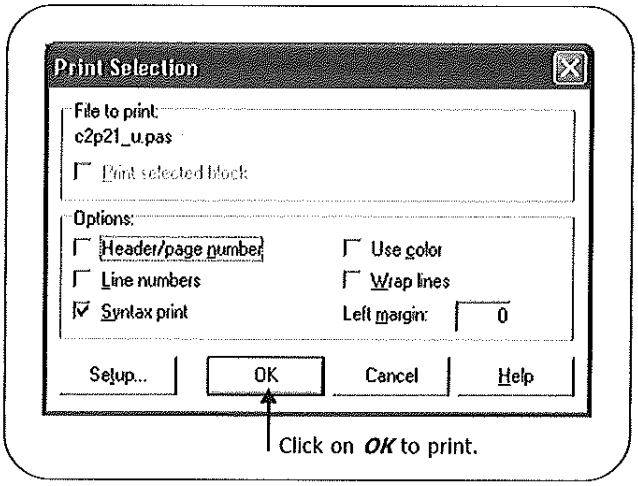
will change the font style.

Add an ***OnDblClick*** Event handler for the form. When the user double-clicks on the form, the caption of the Button must change to "Well Done", the font style must change to Underline and the Left and Top position properties of the Button must change to 25 and 15 respectively.

Printing your work

To print the **unit** make sure the unit you wish to print is the active screen by clicking on it.

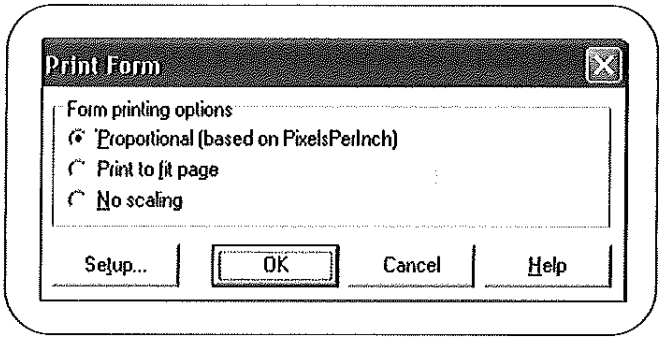
Select ***File, Print*** and a dialog box will appear:



Click ***OK*** to print the unit.

To print the **form** make sure the form is the active screen by clicking on it. Select ***File, Print*** and a dialog box with the following format will appear:

Click ***OK*** to print the form.



You can also add a [Print] Button on your form.

The following code will enable you to print the form:

```
frmAForm.Print
```

where frmAForm is the name of the form.

NOTE

We have barely touched on the components that are supplied with Delphi.

Try looking at

Labelled Edit (if you have Delphi 7 on the additional page)

Progressbar

Statusbar

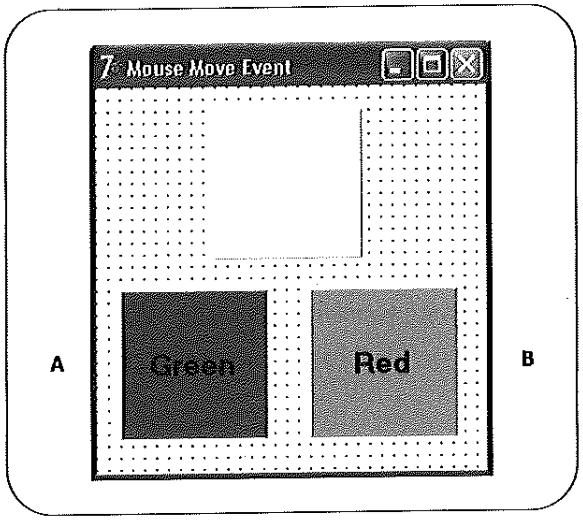
Menu

We will cover some of these later, but experimenting now will do you no harm and you might find it exciting to see some of the things that Delphi components make possible!

Test, improve and apply

Practical Exercises

1. The MouseMove event.



- Start a new application.
- Place the following components on the form named frmMouseMove:
  - pnlSlave
  - pnlGreen
  - pnlRed
- Change the captions in the OI as indicated.
- Change the Color properties of pnlGreen and pnlRed as indicated.
- Add an OnMouseMove event for pnlGreen. When the user moves the mouse over the Green panel the colour of pnlSlave must change to green.
- Do the same for pnlRed.
- Save the unit as c1MouseMove_u and the project as c1MouseMove_p.
- Run the program.
- Now add an OnMouseMove event for the form. When the mouse moves over the form, the colour of pnlSlave must change to the colour of the form.

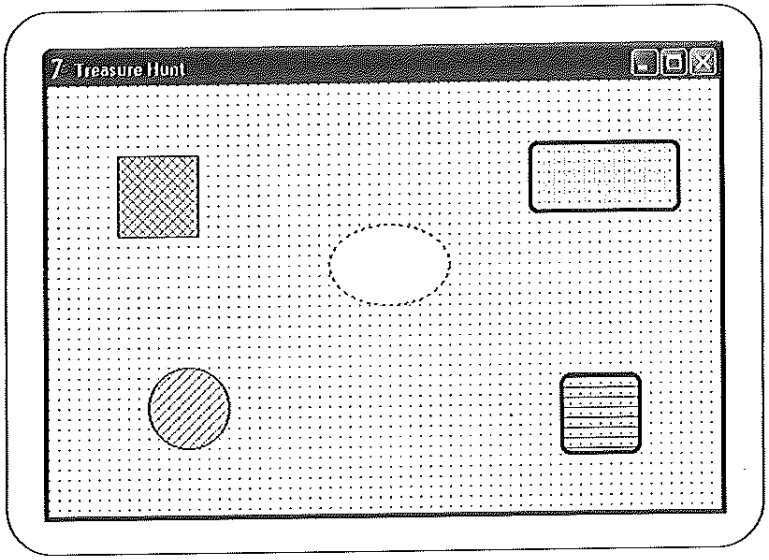
See if you can answer the following questions:

- How many OnMouseMove events take place when you move the mouse from left to right over the form between A and B?
- Can you find the "hotspot" of the mouse arrow?

Tip
Do not give pnlSlave a specific colour. Use the statement:
pnlSlave.color := pnlGreen.color, etc.

2. Treasure Hunt

a) We want to write a Delphi program to simulate a treasure hunt. There will be a number of arbitrary shapes placed on a form that the user cannot see. If they right click on a treasure it is made visible. The user must try and "find" all these treasures.



- Start a new application.
- Place different shapes on the form.

  We can use the Shape component, which is found on the Additional Palette. Each Shape can be customised by changing its Brush, Shape and Pen properties.
  - We need to hide these shapes. This can be done by setting their *brush* and *pen* colours to the same as that of the form.
  - Write an OnActivate Event handler for the form (frmHunt) as follows:

```
Shp1.Brush.Color := frmHunt.Color;
Shp1.Pen.Color := frmHunt.Color;
    etc.
```

  - Now write the 'click' Event handler for the shapes. (It will be an *OnContextPopup Event handler.*) The 'click' event for a Shape is in fact invoked by a *right-click of the mouse.* The 'click' handler for each Shape must set the colour of the shape so that it can be displayed and disabled as a component. The Event handler for one such Shape might look something like the following:

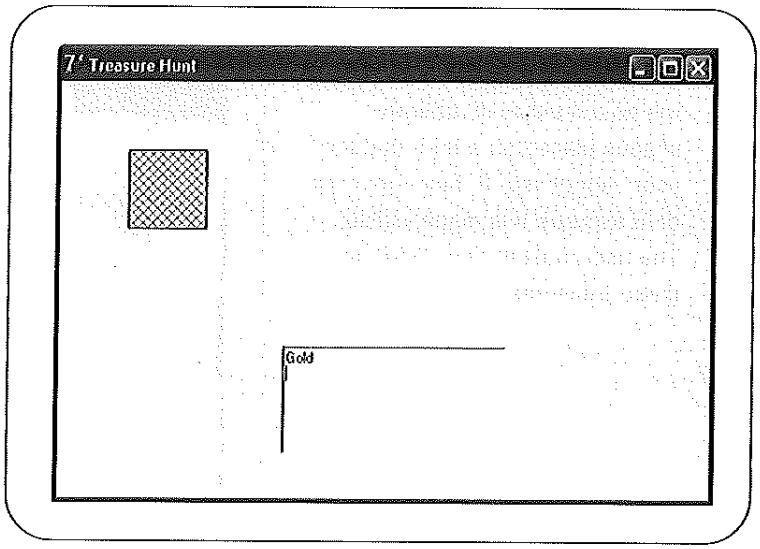
```
procedure TfrmHunt.shp1ContextPopup(MousePos:TPoint;
                                    var Handled:Boolean);
begin
    shape1.Brush.Color := clRed;
    shape1.Pen.Color := clBlue;
    shape1.Enabled := False;
end;
```

You need to use the colours you used for your shapes.

  - Complete the OnContextPopup Event handlers for each of your Shapes.
  - Save your unit as c2Treasure_u and the project as c2Treasure_p.
  - Run the program.

b) It would be nice for the user to see a brief message if and when they pass over a treasure.
This can be accomplished by setting the ShowHint property for all the shapes to be True. Remember that you can set the property for all the shapes at the same time!
Give each treasure (shape) a different value for its Hint property. Set them to different "treasures" such as gold, silver, coins and jewels etc.

c) Add a RichEdit which will "list" all the treasures found so far!



Test yourself

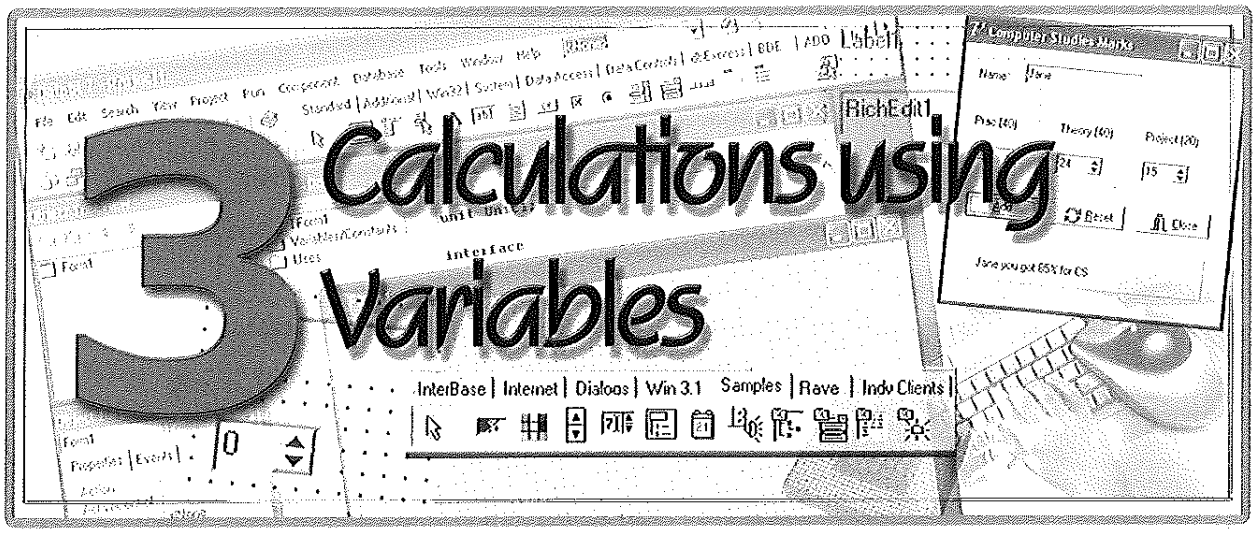
3. Check that you know / can do the following:

		✓
Knowlege	**I can explain in my own words**	
	which components are used for input and which for output in a Delphi program	
	the reason why a Button is used in a Delphi program	
	what the difference is between the *Text* and *Name properties of an* Edit	
	what the difference is between the *Caption* and *Name* properties of a label	
	what a *method* is	
Skills	**I can**	
	clear Edits and Labels during the execution of a Delphi-program	
	write a basic program which reads in data, processes the data and displays output when an *event* occurs	
	simultaneously change the properties of more than one component via the Object Inspector	
	use the Hungarian naming convention to give names to components	
	a program more user-friendly by using the Reset and the Close BitBtn	
	make a program more user-friendly by using the Clear and SetFocus methods of an 'n Edit	
	use the RichEdit	
	make the user interface more professional by making use of Panels	
	determine which events are associated with a particular component	
	write an Event handler for the OnClick event as well as other events associated with a component	
	print a form and a unit	

We have now learnt that

A component

- has properties
- can react to events
- can perform tasks (called Methods).



After you have completed this chapter, you should be able to

- describe the input, processing and output aspects of solving a problem
- work with variables of different types
- write basic programs that use mathematical functions / operations
- show which part of the Delphi program contains the Event handlers that you have written
- write programs to make the computer do basic arithmetic such as adding and averaging data entered by the user
- work with constants
- differentiate between syntax, logic and run-time errors
- display output neatly formatted on the screen.

Introduction

We have already seen that making the computer do things that are cool and interesting is actually quite easy. The real usefulness of computers is in their ability to be more than just 'cool' and interesting – it is in their ability to do calculations and process data. The whole aim of learning to program is learning how to make the computer solve problems and do calculations – so that it can make our lives easier!

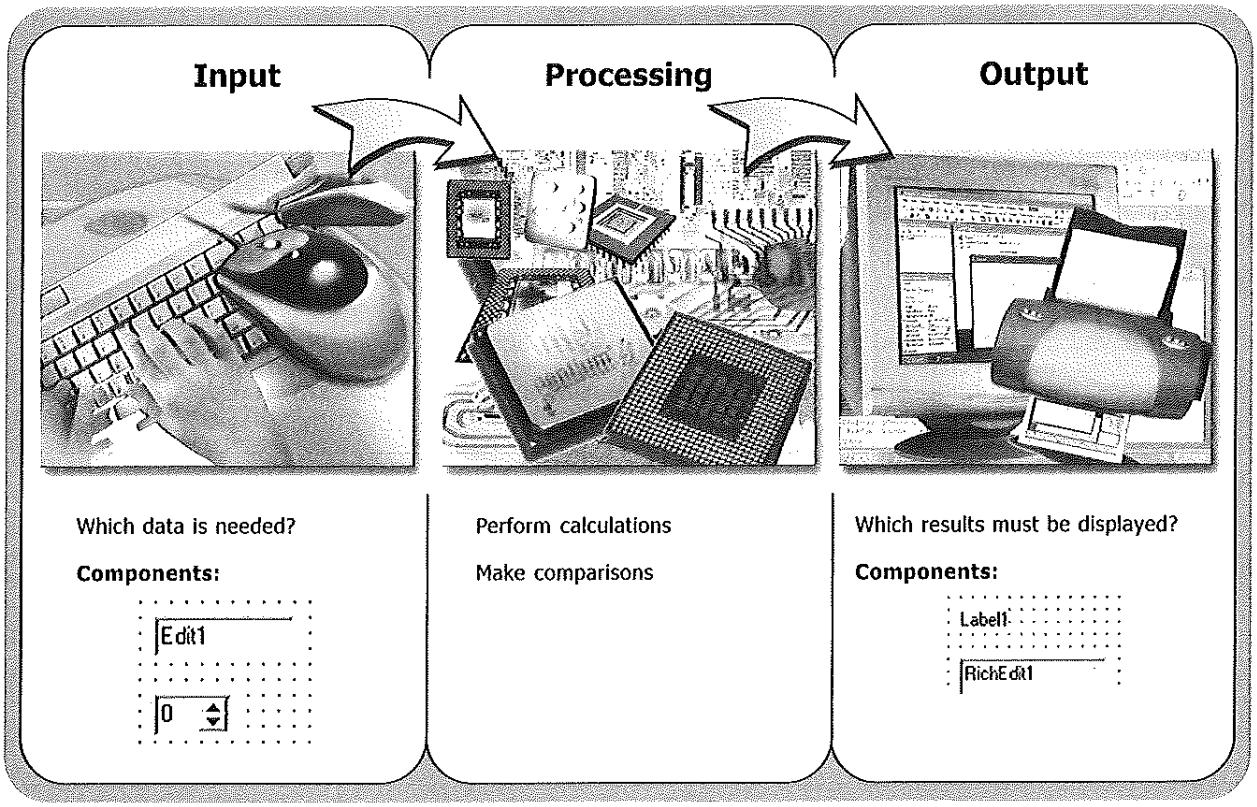
Before we start to try and write code and solve problems for programming, we need to *understand* how computers work and what they actually do whenever we use them to accomplish a task.

And it's going to sound incredibly simple, incredibly basic. And you are probably not going to believe it, but here it is. They

- accept input (from various sources, keyboard, mouse, storage media)
- generate output (to various devices such as screen, storage media, printers)

- store and retrieve data
- perform calculations (usually mathematical in nature)
- make comparisons (and then change what they do as a result of this)
- repeat the steps above in different ways and at different levels to accomplish the task.

And that's it. *Nothing more.* No matter what we *think* they are doing – this is what is actually happening.



The first thing we have to do when learning to program is therefore learning to ***identify input and output*** and ***decide on processing***.

The IPO table

When we begin to examine a problem, we start by using a table to show what **I**nput, **P**rocessing and **O**utput is involved. We call this table the IPO table.

Example 1

The Grade 11 learners are selling raffle tickets to raise money for the Matric dance. The price of a ticket is R5. The name of the learner must be entered and the number of tickets he/she wants to buy. The computer must then work out the amount to be paid and display it with an appropriate message. E.g. Sheldon you need to pay R50.

We can compile the following IPO table:

Input	Processing	Output
Name of learner Number of tickets	Amount payable ← Number of tickets X R5	Name of learner and amount payable

Algorithm

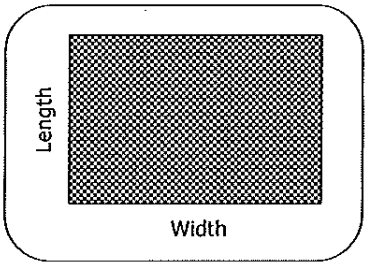
An **algorithm** is a step-by-step description of how to solve a problem. It is not language specific.

The "←" is used to indicate that we work out the right hand side to determine the value of the left hand side.

Example 2

A builder wants to calculate the perimeter and surface area of the floor of a room.

You need to decide here for yourself what input, processing and output will be needed.



Input	Processing	Output
Length, Width	Perimeter ← 2 x Length + 2 x Width Area ← Length x Width	Perimeter Area

Example 3

Enter the name of a learner and his/her marks (as a percentage) for the three Computer Studies tests he /she wrote this term. Determine the average of the three marks. Output his/her name and average.

Input	Processing	Output
Name of learner Three marks	Average ← (Mark1 + Mark2 + Mark3)/3	Name of learner Average

Example 4

You purchase three items at the supermarket. The total price must be calculated. The VAT, at 14%, must then be calculated and added to the price. The price without VAT, the VAT and the final price including VAT must be displayed.

Input	Processing	Output
Prices of the three items	Total price ← Sum of prices of the three items VAT ← 14% of Total price Final price ← Total price + VAT	Total price VAT Final price

Checkpoint: Try the following yourself

Draw an IPO table which indicates the input, processing and output that must take place for the following problems.

1. The PickAJean shop is selling all jeans at a discount price of 10%. When the price of your sales is entered the computer must calculate the discount. VAT of 14% must be paid after the discount is subtracted. Display the price, the discount, VAT and the total amount to be paid.
2. Mrs Botha makes Cremora tarts for Home Industries. She often makes more than one tart and needs to adapt the ingredients for any number of tarts. When she enters the number of tarts the new quantities required must be displayed.

   Ingredients for one tart:

   250 ml Cremora

   125 ml Boiling water

   1 tin (397 g) Condensed Milk

   125 ml Lemon Juice
3. Katlego runs a flea-market stall where he sells packets of sweets for 50c each. He thus needs many 50c pieces for change. If he enters an amount of money he wants to know how many 50c pieces he will receive and what the change will be. E.g. for R2.80 he will receive five 50c pieces and 30c change.

Variables

The problems above all make use of variables (as in Maths). Examples of these are Length, Width, Average, Mark1, etc.

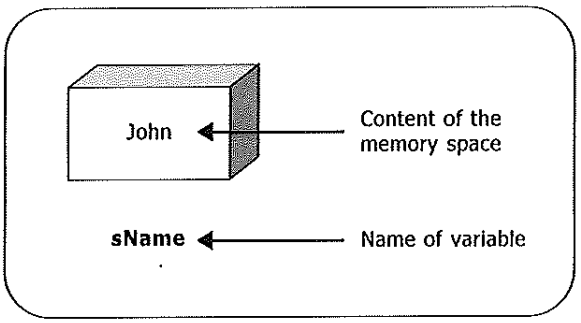
In short we can say that a *variable* is *a value that can change.*

We try not to make use of names like X and Y for variable names. We rather use more meaningful names.

We also make use of variables to temporarily store data values in programming. These variables can then be used when we describe to the computer what needs to be done with them.

A space in the memory of the computer is allocated to each variable where you can save a value under a specific name. This value can be used later.

This space is like a property of a component. It is a box to store data. The difference is that it does not belong to a component so Delphi does not create it for you. You have to create it yourself.



Declaring variables

All variables must be declared before you can use them. Two important things need to be heeded when declaring a variable:

- The *name* of the variable must be indicated.
- You also need to define the *data type*, i.e. tell the computer what type of data can be stored in that variable. It could be a person's name (a number of characters), your Computer Studies' mark (an integer), the price of an item you want to buy (a real value), etc.

The reason for this is because space has to be allocated in memory where this data can be saved. All data types are not of the same size and will therefore not take up the same amount of memory space.

We declare a variable by using the keyword **Var** followed by the variable name, a colon (:) and the type of the data that will be stored in the variable.

```
var
  VariableName : <datatype>;
```

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

```
var
   VariableName1, VariableName2, VariableName3 : <datatype>;
```

Variable names

When we decide on a name for a variable we are going to use meaningful words. This makes the function of the variable more apparent. There are, however, some rules that must be adhered to with variable names in Delphi.

Rules for variable names

- It must start with a letter or an "_" (underscore sign).
- It may not contain spaces.
- Letters, numbers and the underscore sign are allowed after the first character. (The only special character that may be used is the underscore sign.)
- It may not be a key word or a reserved word (a word that already has a meaning in Delphi).
- Only the first 255 characters of the name will be considered as meaningful.

Multi-word variable names are probably the most effective in describing the contents of the variable.

Write them with a capital at the start of each word – e.g. FirstName, TotalCost, etc.

Data types

Let's look at the simple data types which we will use in Delphi.

TIP
We need to think carefully when working with a program, as we will be working with many different types of data. We are less likely to get confused if we use the convention of making the first letter of the variable name that of the data type.

String

A string consists of a combination of characters and can be used, for example, to store the surname, name, address, etc. of a person. In Delphi strings are written in single quotes.

The Caption and Text properties of components are strings.

The smallest possible string contains 0 characters and is called an empty string.

Examples

'Linda', 'The birds sing', 'CTA 473 EC'

Declaration

A string may be declared in one of three ways

```
var
  sName: shortstring;
```
← The string can contain 255 characters.

```
var
  sName: string[10];
```
← This string can contain 10 characters.

```
var
  sName: string;
```
← Unlimited. (Delphi allocates memory to the string as needed.)

Integer

We declare a variable of type Integer if we intend to only do integer calculations with a variable. The number of items that you wish to purchase or your position on a class list are examples of integers.

The Left, Top, Width and Height properties of all components are examples of integers.

A variable declared as an Integer in Delphi can contain whole numbers from -2147483648 to 2147483647.

Examples

510 1000000 -305

Declaration

```
var
    iNumber: integer;
```

Info
No spaces or commas are allowed in an integer value in Delphi.

Real

We make use of the Real data type when working with numbers that contain decimals. The amount of money we pay for an item is usually a decimal value as we are working with Rands and Cents.

Examples

2.5 0.14 -29345.56784

Declaration

```
var
    rValue: real;
```

We make use of the decimal point and not a comma in Delphi.

Char (character)

A character is a letter, number, punctuation mark or any other character from the ASCII table. A character consists of a single character (also written between single quotes) and can be used, for example, to store the class or gender of a pupil, provided that the class or gender consists of only one character.

Examples

'M' '*' '?' 'ë'

Declaration

```
var
  cSymbol    :char;
```

Boolean

A Boolean data type can only be one of two values, namely TRUE or FALSE. One byte is reserved for the Boolean data type. The Boolean data type will be discussed in more detail at a later stage.

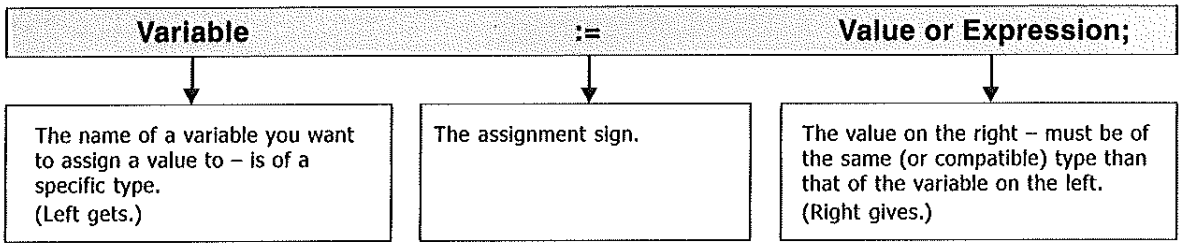
You have used Boolean properties when you disabled a Button (`Button1.Enabled := False;`) and made an Image visible (`Image1.Visible := True;`) in Chapter 1.

Declaration

```
var
    bPass    :boolean;
```

The use of variables in expressions and assignments

We use the assignment symbol to assign a value or an expression to a variable.



The following operators can be used in expressions:

	Operation	Datatype that can use it
Addition	+	Integer, real
Subtraction	-	Integer, real
Multiplication	*	Integer, real
Division	/	Integer, real
Integer division	DIV	Integer
Remainder after division	MOD	Integer
Concatenation	+	String

For **normal division,** the '/' operator is used. The result is always a real value. 10/5 gives us the answer 2.0 and 10/3 gives us the answer 3.333333333333.

There are times when we need only the integer part of a division operation. To do this we use the **DIV** operator.

10 DIV 2 = 5 10 DIV 3 = 3 87 DIV 10 = 8

Note that when division is done with two integers, the answer must be stored in a real variable. E.g.
rAnswer := iNum1 / iNum2;

The **MOD** operation also does an integer division but only gives the ***remainder***. The remainder not the same as the decimal part of the result after normal division. Pay attention to the table below:

/	DIV	MOD
10 / 3 = 3.33333	10 DIV 3 = 3	10 MOD 3 = 1
21 / 7 = 3	21 DIV 7 = 3	21 MOD 7 = 0
94 / 10 = 9.4	94 DIV 10 = 9	94 MOD 10 = 4
3 / 5 = 0.6	3 DIV 5 = 0	3 MOD 5 = 3

The same order of evaluation used in Maths is also applicable to expressions that make use of integer and real values.

Operation	Priority
Brackets	First
* / DIV MOD	Second
+ -	Third

Examples of assignment statements

```
sMyPassWord := 'CTA 473 EC';
sName := edtName.Text + 'SMITH';
iTotal := iMark1 + iMark2;
iNumberPackets := iTotal DIV 5;
rAverage := (iTest1 + iTest2) / 2;
```

Working with Integers and Strings

Activity

Consider example 1 on page 36 again:

The Grade 11 learners are selling raffle tickets to raise money for the Matric dance. The price of a ticket is R 5. The name of the learner must be entered and the number of tickets he/she wants to buy. The computer must then work out the amount to be paid and display it with an appropriate message. E.g. Sheldon you need to pay R50.

We are going to solve a problem by breaking it up in four steps namely:

Step 1: Construct the IPO table.

Step 2: Choose and declare variables.

Step 3: Design the user interface.

Step 4: Code the Event handler.

Step 1: Construct the IPO table

Input	Processing	Output
Name of learner Number of tickets	Amount payable ← Number of tickets X R5	Name of learner and Amount payable

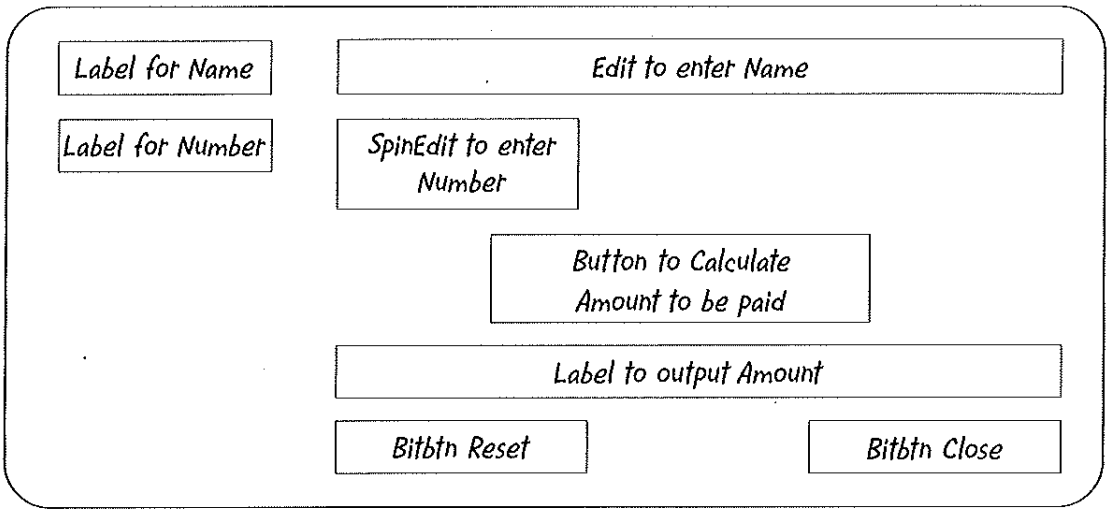
Step 2: Choose and declare variables

Assume we use the following variables:

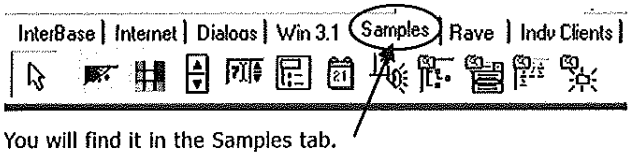
```
var
  sLearner                :string;
  iNumber, iAmount        :integer;
```

Step 3: Design the user interface

Decide on the components you want to use.



The spinEdit is a component used for entering integer data.



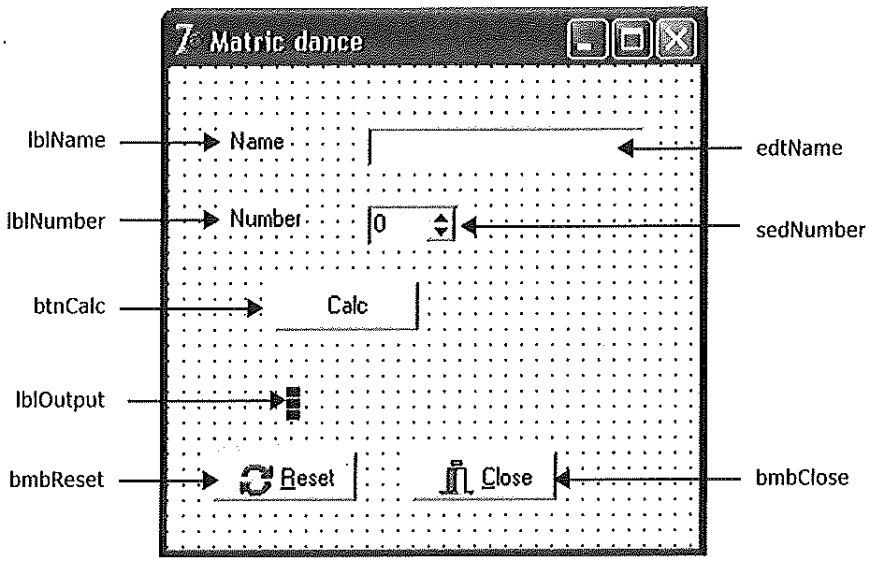
You will find it in the Samples tab.

SpinEdit

- A SpinEdit is a component used when the user only wants to enter Integers.
- A SpinEdit has a Value property.
- The prefix for SpinEdit is **sed**.

Now create the following interface:

As soon as you place components, immediately give them meaningful names and change the properties of the components as needed.



Step 4: Code the Event handler for the Button

Double-click btnCalc and add the following code:

```
procedure TFrmTickets.btnCalcClick(Sender: TObject);
var
  sName             : string;
  iNumber, iAmount  : integer;
begin
  sName := edtName.Text;
  iNumber := sedNumber.Value;
  iAmount :=  iNumber * 5;
 lblOutput.Caption := sName + ' you need to pay R '+ IntToStr(iAmount);
end;
```

Tip

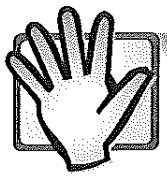
Try autocomplete. Type in the first 2 letters of a variable name and press <Ctrl><Space bar> and then choose the variable name from the list that pops up!

Label for Output
(String)

IntToStr

Integer variable used for calculations
(Integer)

A Label has a Caption property that can only contain text. If we want to display an Integer number we must first convert the number to a string. For this we use the function **IntToStr.**



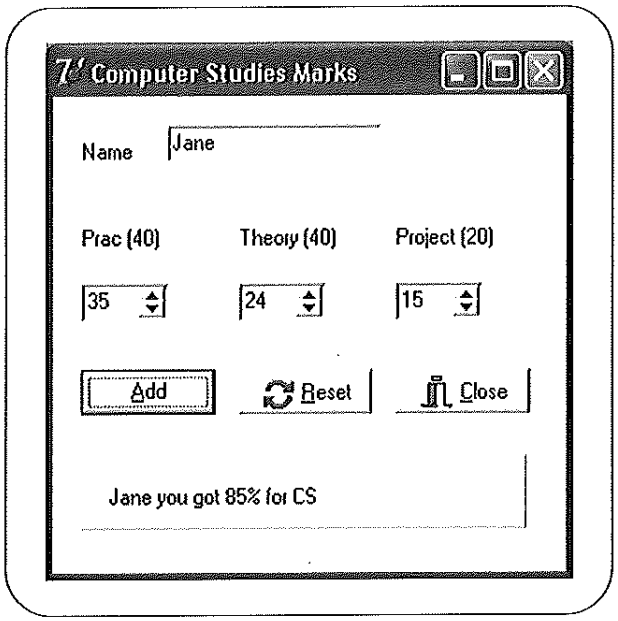
Checkpoint

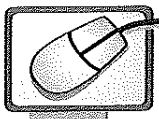
Create the following interface.

A user must enter the name of the learner, the practical mark (out of 40), the theory mark (out of 40) and the Project Mark (out of 20).

When he/she clicks the [Process] Button the name of the learner and his mark (sum of the three marks) must be displayed.

Change the MaxValue property of the SpinEdit components to 40, 40 and 20 respectively.





Working with real data

Activity

Consider example 3 on page 37.

Enter the name of a learner and his/her marks (as a percentage) for the three Computer Studies tests he /she wrote this term. Determine the average of the three marks. Output his/her name and average with an appropriate message. (Determine the average correct to one decimal place.)

Step 1: IPO table

Input	*Processing*	*Output*
Name of learner *Three marks*	*Average ← (Mark1 + Mark2 + Mark3)/3*	*Name of learner* *Average + Message*

Step 2: Choose and declare variables

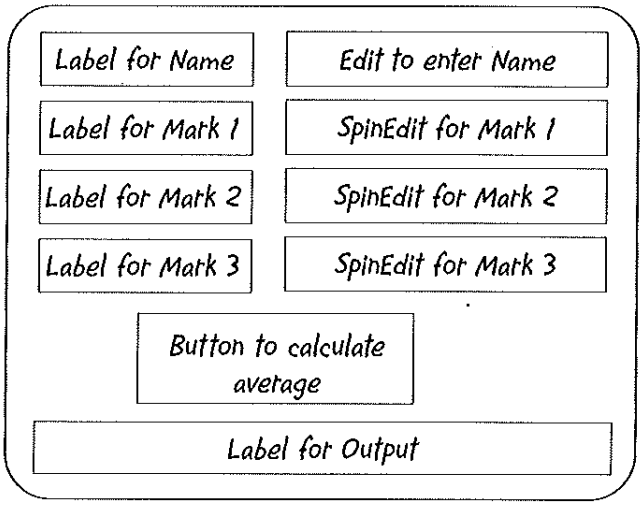
Assume we use the following variables:

```
var
  sLearner                  :string;
  iMark1, iMark2, iMark3    :integer;
  rAverage                  :real;
```

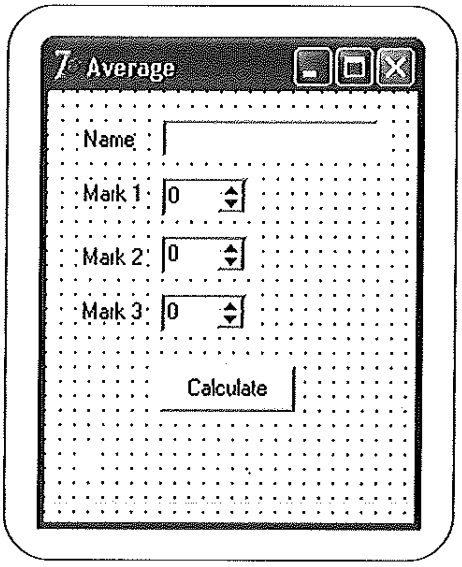
Tip
rAverage will be of type Real because we use the "/" for division that is a real operator.

Step 3: Design the user interface

Decide on the components you want to use.



Now create the interface:



Step 4: Code the Event handler for the Button

Double-click btnCalc and add the code.

Hint:

If we want to display a real number in the caption property of a label, we must first convert the number to a string. For this we can use the function **FloatToStrF.**

```
lblOutput.Caption := FloatToStrF(rAverage, ffFixed, 5, 1) ;
```

Note the following:

- The variable rAverage will be converted from a real number to a string.
- **ffFixed** is a reserved word in Delphi which means *fixed point format.*
- The **5** and the **1** indicate that the value must be printed using **5** positions correct to **1** decimal. If for example, the value is 76.67, then it will be displayed as follows:

Space			Decimal point	Number rounded to 1 decimal
	7	6	.	7

Five positions are used. The FloatToStrF function specifies that only one decimal place must be displayed after the decimal point. The first (leftmost) position is therefore left empty (a space).

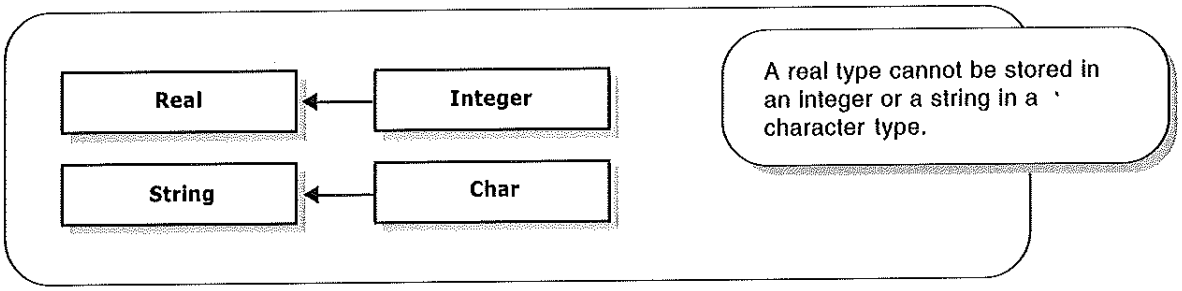
Note

The value in memory is still stored as 76.67. It is only *displayed* as 76.7. Therfore if you later make further use of the variable rAverage, the value of 76.67 is used still.

Compatible data types and conversion of types: summary

The left and right sides must be of the same type or at least compatible types.

Because different variable types do not use the same amount of memory space, a type which needs more bytes for storage cannot be assigned to a type that uses less bytes for its storage.



If the expression on the right hand side contains a mixture of integer and real values, then the variable on the left side must be of type Real.

Please note
If you have real variables on the RHS or an operator that works with real values on the RHS the variable on the LHS must be declared as Real.

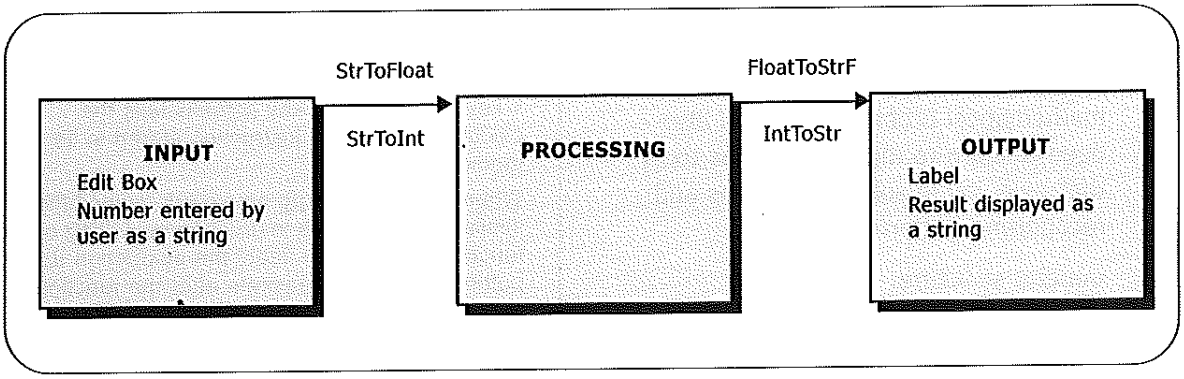
Examples:

```
rVAT := iPrice * 0.14;
rValue := 2 * rLength + 2 * rWidth;
rAnswer := iValue1 / iValue2;
```

We use functions to convert between string and numerical data types

We usually use an Edit to read data in and a Label to show output.

An Edit and a Label can only store *strings*. Conversion to a numerical data type must first occur if we want to do numerical calculations on these values.



Checkpoint

Follow the four steps of problem solving for each of the following problems:

1. Now do example 4 on page 37.
2. A waiter is paid R 12.50 per hour. The user must enter the number of hours that the waiter has worked. The program must calculate and display the wages of the waiter.
3. A learner's mark for a test is input, as well the total that the test was out of. The program must calculate and display the percentage obtained by the learner
4. The program must direct the user to enter a temperature value in degrees Fahrenheit. This temperature must be converted to degrees Celsius and the answer displayed.

   Formula: C = (Fahrenheit – 32) * 5/9
5. Write a Delphi program that will convert a monetary value. Depending on which of two buttons the user clicks, they must be able to either convert the value from Pounds to Rands or vice versa. Assume that the exchange rate is such that 1 Pound = R 7.56
6. Jing-jing has a stall at the flee market. He sells packets of sherbet for R3 a packet. Children often gives him a amount of money and say: "I want sherbet for all this money". He wants a Delphi program on his laptop that will enable him to enter the amount and the computer must then indicate how many packets of sherbet he needs to give the customer as well as the change that must be given. Write the necessary Delphi program. (Tip: Use DIV and MOD.)

Useful Mathematical functions in Delphi

Besides basic arithmetic, Delphi (like most other programming languages) offers some more advanced maths functions that you can use in your programs. You can use these to solve mathematical problems, but you also need to learn to think of them as tools that help you in solving problems and recognising patterns.

General structure:

Answer := <FunctionName>(Value);

Using maths functions in Delphi is like using all the maths operators you learnt in primary school.

Remember using your calculator to calculate the square root of a number? You would type the number into the calculator and then press the √ key. Pressing this key actually tells the calculator to use an instruction stored in its memory.

When you write a program you do something similar – you use the instruction by writing out the functions name. You 'feed' the instruction data by putting the data in brackets after the function name (this is called a parameter). You are responsible for displaying the result on the screen yourself.

Trunc, Round

Often we need to *change a real number into an integer.* To be able to do this we can use one of two functions, namely Trunc or Round.

Trunc	Round
Trunc just chops off the decimal point of a real number – you are always only left with the integer part.	Round takes a real number and rounds it to the nearest integer.
Trunc(7.5) = 7	Round(7.5) = 8
Trunc(7.4) = 7	Round(7.4) = 7

Note that Trunc and Round always give an answer that is an integer.

Frac

Besides obtaining the integer part of a real number, sometimes we need to find out exactly *what the decimal part of the real number is.* The function we use to do this is the Frac function.

Frac (74.89) = 0.89
Frac (0.3728) = 0.3728
Frac (994.321) = 0.321

It always gives an answer that is a real number.

If the number is halfway between two integers, it will always be ***rounded to the nearest even number***. E.g.
Round(7.5) = 8 and also
Round(8.5) = 8

Sqr, Sqrt

Sqr takes a number and gives you the square of the number. The data type of the result must be the same as that of the number that was squared.

Sqr(5) = 25 Sqr(-10) = 100 Sqr(2.5) = 6.25

Sqrt takes a number and gives you the square root of the number. The result will always be a real number.

Sqrt(25) = 5 Sqrt(81) = 9 Sqrt(46.24) = 6.8

Activity

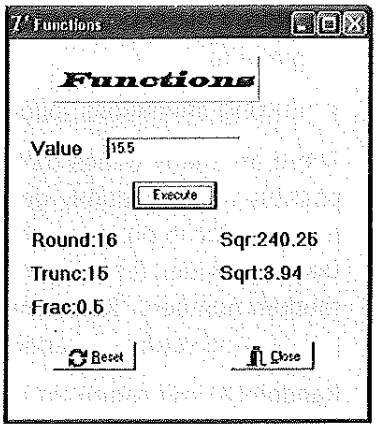
Experimenting with functions

Create the following interface:

When the user enters a number and clicks the [Execute] Button, the results of the functions must be determined.

Display the answers for Sqr and Sqrt to two decimal places.

Make use of **FloatToStr** (instead of FloatToStrF) when you display the value for Frac. (In other words, show the actual calculated value, as opposed to being rounded to a certain number of decimal places).



Pi

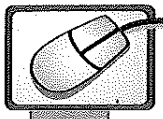
Pi simply gives you a value for the mathematical constant PI. (You know the formula to calculate the circumference or area of a circle.
Circumference = $2\Pi r$ and Area = Πr^2.)

In Delphi the function could, for example be used as follows:

```
rArea := Pi * Sqr(radius);
rCircumference:= 2 * Pi * radius;
```

Pi is a function that returns a real number.

There are many more functions that Delphi can offer you that work with maths; trig functions, financial functions and even some statistical functions. Look them up using the online help.



Using Pi

Activity

Write a Delphi application that will do the following:

Calculate the circumference and the area of a circle when the user enters the **diameter** of the circle (use the Pi function).

Random

Sometimes parts of our programs need to be random. This can range from

- making text appear in random colours to
- creating a whole bunch of random numbers to
- making 'enemies' appear at unpredictable times in a game to
- making animations follow unpredictable paths over our screen at unpredictable speeds.

Tip
Random (100) has 100 possible answers (from 0 to 99). To generate a number between 1 and 100 do the following:
iNo := Random (100) + 1;

There are many things that we can possibly need randomly generated numbers for in our programs. Luckily Delphi provides function to generate random numbers. **Random** takes an integer and returns a random integer.

Random(X) will return an integer in the interval [0 .. X-1].

NB: To make the numbers generated by the random function less predictable use the **Randomize** procedure. Randomize simply scrambles the Random function. It is only necessary to use this function once in your program, preferably when the program starts (use the OnActivate event of the form).

Activity

Fun with Random

We are going to write a very basic game. Although it is not a game that one would want to play again and again, you can at least trick your friends once!

The concept:

- We have a Button on a form that offers the user **R 1 000 000** if the Button is clicked.
- This must work – it must be possible for the user to win! Except, when one tries to click the Button it ***runs away*** from the mouse!
- The result: the Button can never be clicked.

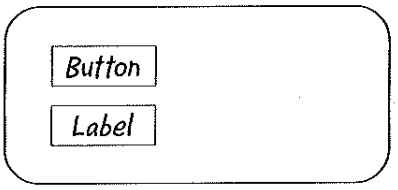
Step 1: Construct the IPO table

Input	*Processing*	*Output*
The movement of the mouse	*Move the button so that the person can't click it*	*Message to tell the person he won*

Step 2: Choose and declare variables

We won't need variables for this program

Step 3: Design and plan the user interface



Planning a user interface is more than just components. In this case we need to think about things like:

- What happens if the user resizes the form?
- Should we limit the minimum size of the form? (How do we do this? – READ the list of properties of the form in the Object Inspector and you will see the properties to use!)
- What happens if the Button tries to move off the form?

Step 4: Code the Event handlers

Events are already partially defined in the task of the program. We formalise this after planning the interface because the type of events that we use largely depends on the interface we create!

Anticipating the user's actions and choosing the correct events to write code for are essential skills to develop.

The OnClick event of the Button

You need to write code for the OnClick event that will change the caption of the label to tell the user they have won R1 000 000.

The OnMouseMove event of the Button

We must determine if the mouse is moving on the Button and move it away before the user can click!

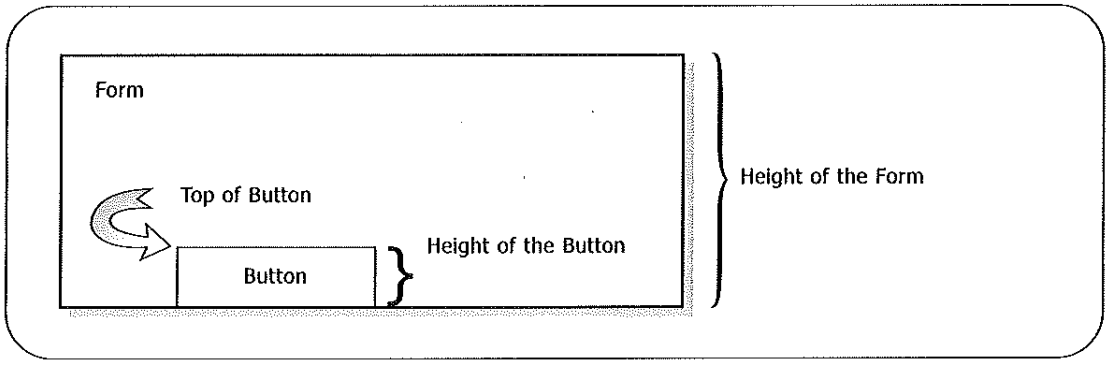
We can move the Button by changing the Left and Top properties of the Button.

This is the working heart of this particular program. If the mouse moves onto the Button we have to move the Button away from the mouse. We need to move a random distance but bearing in mind the questions asked in the user interface, we have to remember:

that the form can change its size

that the button must not move off the form.

Consider the form and the Button:



On the diagram you can see that the top of the Button needs to be at least the height of the Button above the bottom of the form to ensure that it does not move off the form.

We will add another 5 pixels for good measure.

The same applies to the width of the form.

We therefore need to enter the following code :

```
button1.Top := Random(form1.ClientHeight - 5 - button1.Height);
button1.Left := Random(form1.ClientWidth - 5 - button1.Width);
```

Why ClientWidth instead of Width?

The ClientWidth of the form is closer to the space that your program actually gets to use – look and you will see that it is slightly less than the actual width of the form.

The same applies to the ClientHeight of the form.

Using these values makes your program a little more accurate.

The OnActivate event of the Form

If we want to use Random, we need to add Randomize. This only needs to be done once and we therefore put it in the OnActivate Event handler of the form.

Between the **begin** and the **end** write the following code:

```
Randomize;
```

Now save your program and test it.

The Structure of the Delphi Unit

When you start a new application (before you add any components or code) Delphi provides the following skeleton code for Unit1.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
end.
```

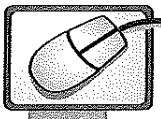
The name of the Unit

Other units used by this unit

Objects this unit consists of: at this stage only the Form

All the procedures in the Unit can use these variables. They are called *global variables*.

The Event handlers and code you enter are added in this section.



Activity

Exploring the unit and the Uses section

- Open a new application.
- Press <F12> and maximize the unit. You will see the skeleton code.
- Ensure that you can distinguish between the different sections of the unit.
- Examine the Uses section where other units can be added.

Why do we need additional units?

Anything that a computer does is made possible by computer code that has to be written. This means that the components that you place on your form – Buttons, Labels, etc, all have to have some code written somewhere that tells the computer what they are and how they work.

Delphi does not put all of this code directly into the place where you write your own program code. Why not? Well, the code for the components adds up to thousands and thousands of lines. Putting this all into the place where you write your own code would confuse you and make it difficult for you to see what is going on in your program.

Instead Delphi uses the system of Units - pieces of code can be written and stored in different files. In fact, each form that you create and use in your program is stored in a seperate unit of its own. For a unit to access the code stored in another unit, you simply add the units name to the **Uses** clause of your program.

The code that makes the components possible is stored in many different 'unit' files. When you add a component to the form, Delphi automatically adds the correct unit name to the Uses clause of your program.

- Only the Forms unit currently occurs in the Uses section because our program does not contain any components at the moment.

The RoundTo and Power functions

RoundTo

The ***RoundTo*** function rounds a number off to the nearest power of 10.

Example: RoundTo (1678, **3**) will round 1678 to 10^3 (thousand). The result will be 2000.

General format:

Result := RoundTo (Number, Decimals);

- The data type of **Number** must be Double. (Equivalent to the Real data type; See Appendix C.)
- **Decimals** can be any integer in the range -37 .. 37.
- **Result** must be a Real value.

	Power of 10	Function	Number	Result
Round Number off to the nearest 1000	3	RoundTo(Number,3)	678	1000
			67	0
Round Number off to the nearest 100	2	RoundTo(Number,2)	67	100
Round Number off to the nearest hundreth ($1/100 = 1/10^2 = 10^{-2}$)	-2	RoundTo(Number,-2)	61.5053	61.51

Note: The RoundTo function actually *changes the value* being rounded and not just the way it is displayed as with FloatToStr of FloatToStrF.

Power

Power takes two numbers and raises the first number to the power of the second number.

General format:

Result := Power(Base, Exponent);

- The data type of **Base** and **Exponent** must be an Extended data type (Equivalent to the Real data type; See Appendix C.)
- Result must be also be an Extended (Real) type.

Examples

Function	Meaning	Result
Power (5, 3)	5^3	125.0
Power (10, 4)	10^4	10000.0
Power (-2,3)	-2^3	-8.0
Power (2,-1)	$2^{-1} = ½$	0.5
Power (9,0.5)	$9^{0.5} = 9^{½}$	3

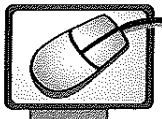
Note:

The RoundTo and Power functions appear in the Math unit.

To be able to use them in our program, we need to add **Math** to the uses section of the unit.

```
uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs, Math;
```

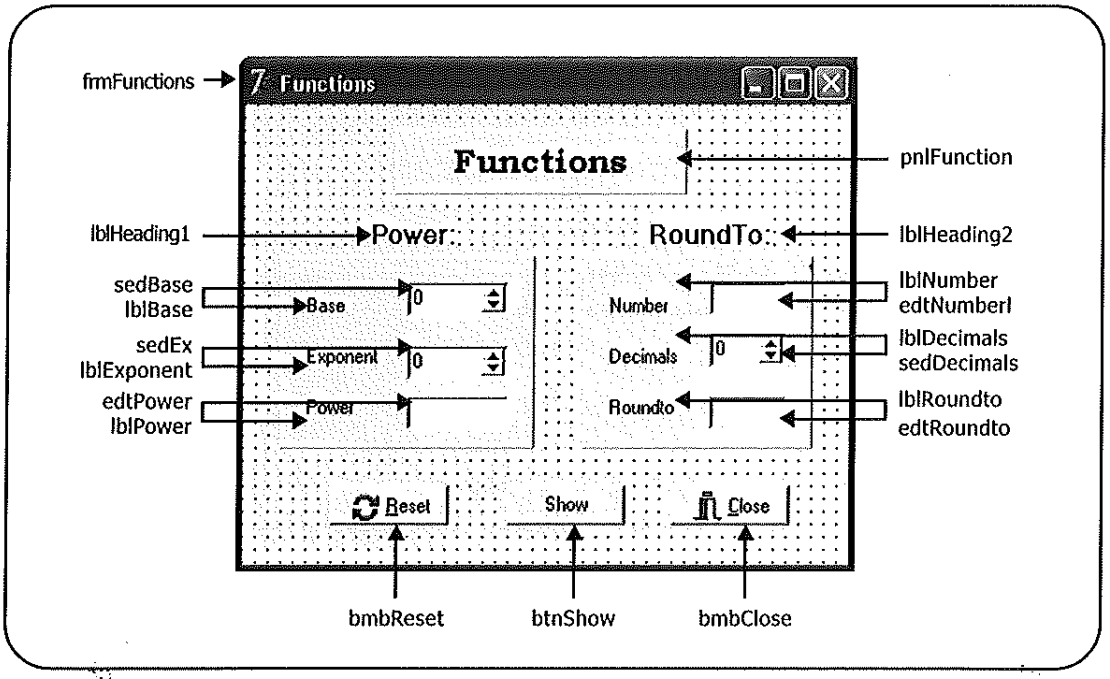
Other units used by this unit.



Activity

Using RoundTo and Power

- Create the following Delphi Interface:
- Name the components as indicated:



- Now look at the Unit again and you will notice the following change:

```
type
  TfrmFunctions = class(TForm)
    pnlFunctions: TPanel;
    pnlPower: TPanel;
    pnlRoundto: TPanel;
    lblBasis: TLabel;
    lblExponent: TLabel;
    lblPower: TLabel;
    sedBase: TSpinEdit;
    sedExp: TSpinEdit;
    edtNumber: TEdit;
    sedDecimals: TSpinEdit;
    lblNumber: TLabel;
    lblDecimals: TLabel;
    lblRoundto: TLabel;
    edtRoundto: TEdit;
    edtPower: TEdit;
    bmbReset: TBitBtn;
    bmbClose: TBitBtn;
    btnShow: TButton;
    lblHeading1: TLabel;
    lblHeading2: TLabel;
```

The name of the form also changed from the default value to the given name (frmFunctions).

All the components you placed on the form can now be seen under the Type declaration.

- Double-Click btnShow and add the code in bold:

```
procedure TfrmFunctions.btnShowClick(Sender: TObject);
var
   iBase, iExp, iDecimals      : integer;
   rNumber, rPower, rRoundto   : real;
begin
   //  Calculate the Power
   iBase := sedBase.Value;
   iExp := sedExp.Value;
   rPower := Power(iBase, iExp);
   edtPower.Text := FloatToStr(rPower);

   //  Calculate the rounded value
   rNumber := StrToFloat(edtNumber.Text);
   iDecimals := sedDecimals.Value;
   rRoundto := RoundTo(rNumber, iDecimals);
   edtRoundto.Text := FloatToStr(rRoundto);
end;
```

You can make use of comments within your program code to indicate what each section of the code does.

Simply place // in front of the text and the computer will regard this as a comment and it will be ignored by the Delphi compiler at compile-time.

Remember the Power and RoundTo functions will only work if you add **Math** to the Uses part of the program.

Double-click bmbReset and add the necessary code to clear all the components and reset the focus.

Now save and run your program.

Programming the computer to do simple arithmetic

The first computers were nothing but calculating machines. Nowadays we expect computers to be far more useful and multi-purpose. Even so, no matter what task they are being used to accomplish, most programs tend to perform some form of calculation or manipulation of numbers – either directly or indirectly.

What we need to do now is learn how to tell the computer how to do simple arithmetic – and then learn to think about how to use this to solve common problems!

Teaching the computer to count

Many of the problems that computers solve involve some form of counting. This is one of the most important and very basic skills that you need to know if you want learn how to program.

The basics of how to count:

- You need an integer variable to count with.
  iCounter is an example of a good variable name to use here.
- You need to *initialise* the counter (give it an initial value) before you use it.
  Most of the time we want to start counting at 0 – so before we ever use the counter variable we set its value to 0 with a simple assignment statement:
  iCount := 0;
- The counting bit is as simple as **iCount := iCount + 1;**
  This instruction increases the value stored in the variable iCount by 1. This is the same as counting!

Please Note:

- We can count backwards as well:
  ```
  iCount := iCount - 1;
  ```
- We can count in 5's (or any other number):
  ```
  iCount := iCount + 5;
  ```

Inc and Dec

There are two other commands that we can use with counting. They are ***Inc*** and ***Dec***.

Inc increases the value of a variable.

Inc (iCount); is the same as ***iCount := iCount + 1;***

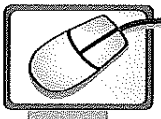
Inc (iCount, 5); is the same as ***iCount := iCount + 5;***

Dec decreases the value of a variable.

Dec (iCount); is the same as ***iCount := iCount - 1;***

Dec (iCount, 5); is the same as ***iCount := iCount - 5;***.

It is shorter to type in ***Inc*** and ***Dec*** and using these commands is a good indication that we are counting rather than doing some sort of mathematical calculation.



Activity

Let the computer count

Extend the Random Game program (on P51) by placing a Label on the form which will display a count of how many times the person has attempted to click the Button.

Tips:

- **Ensure that your counter variable has been initialised.**
  Initialise the counter variable in the *form's* OnActivate event. (If you initialise it in the Button's OnClick event, you will reset the amount to 0 every time the Button is clicked!)
- **Make your counter variable a global variable.**
  If it is a variable that is only used in the OnClick procedure then it only exists when that procedure is run. It is destroyed and loses its contents every time the procedure is completed!

Checkpoint : Try the following yourself

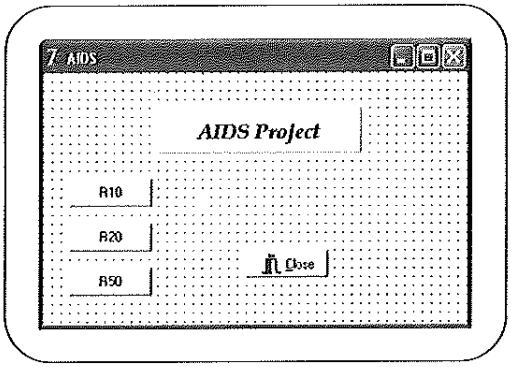
The Outreach Society of the school is collecting money to buy clothes and toys for the children in the Aids Home. They asked every learner in the school to make a financial contribution – either R10, R20 or R50.

Create the Delphi interface shown on the following page.

- Write OnClick Event handlers for the three Buttons. When the user clicks the appropriate Button, the amount raised so far, increases by the value of the Button, which was clicked on. The new total amount must then be displayed.

  E.g. If the user first clicks the [R10] a message 'The total amount raised is now R 10' must be displayed.

  If the [R50] Button is then clicked, the message must change to: 'The total amount raised is now: R 60'. etc.

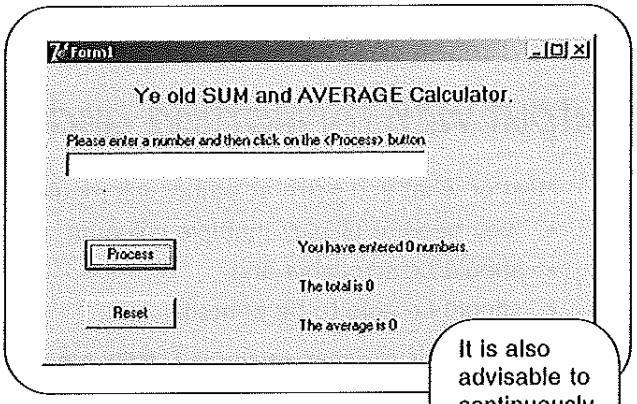


Writing a really useful program that works is one of the greatest feelings you'll ever have!

Calculate the sum and average

We are going to write a program that allows the user to enter numbers. The program will give the sum and average of the numbers entered.

Your program ought to be written in such a fashion that the user can control what happens. As per usual though, you need to give the user clear instructions.



It is also advisable to continuously provide feedback.

We will need the following **variables**:

rNumber
rSum, rAverage (real numbers)
iCounter (an integer to hold the amount of numbers the user has entered).

Algorithm

Initialise iCounter and rSum to 0 on the ***form's*** OnActivate event.

The following must occur every time the user clicks on the [Process] Button:

The counter must be incremented.

The value input must be added to the sum:

```
rSum := rSum + rNumber;
```

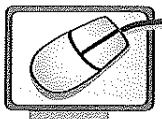
The current average must be calculated:

```
rAverage := rSum / iCounter;
```

(Also display all current values.)

If the user clicks on Reset, then iCounter, rSum and rAverage must be reset to 0.
Also change the appropriate messages on the screen.

The program above could work with any amount of numbers – the user could carry on entering numbers and calculating for as long as they wanted to if you didn't try to control the program.



Activity

Sum and Average

Now write the above-mentioned sum and average program.

Checkpoint: Try the following yourself

Write a program to assist the teacher with the calculation of her marks.
She must be able to

- specify the maximum marks a learner could have achieved (this will be done once only)
- enter each learners' mark.

The program must

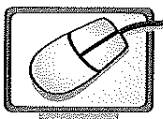
- display the percentage of each mark as it is entered
- continuously display the number of marks entered
- continuously display the class average.

Output with formatting and spacing of results

Most of what you learn at school will require you to create output lists that

- may need to display multiple columns
- should be scrollable and
- should be able to be printed.

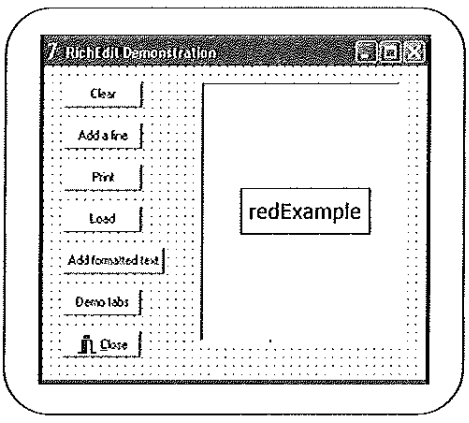
The RichEdit component (on the ***Win 32***) tab of the component palette is the ideal component to use for this – as it, amongst other things, allows you to format your output by including bold, italic, different font sizes, etc.



Activity

Exploring the RichEdit

- Create the following interface.



We are going to write Event handlers for each of the Buttons by ***typing code for the OnClick event of each Button:***

btnClear

- Code: `RichEdit1.Lines.Clear;`
- This code simply clears the contents of the Lines property.

btnAdd_a_Line

- Code: `RichEdit1.Lines.Add ('Adding a line of text');`
- This code simply adds a line of text to the Lines property.

btnPrint

- Code: `Richedit1.Print ('My Print Job');`
- This code causes the RichEdit to print its contents to the default printer.

btnLoad

Do the following:

- Start up ***Word*** (or whatever word processor you use).
- Type a document using different fonts, bold, italics, underline, paragraph alignment, bullets, etc.
- Save the document in the same folder as your Delphi program.
- *Be sure* to save the document as a ***Rich Text Format*** document called 'Test' (with the .rtf extension).

Type in the OnClick event of btnLoad:

- Code: `RichEdit1.Lines.Loadfromfile('test.rtf');`
- This code will load the document you created in ***Word*** and you should see that all your formatting has stayed the same!

btnAdd_formatted_text

```
RichEdit1.SelAttributes.Size := 12;
RichEdit1.SelAttributes.Color := Random(650000);
RichEdit1.SelAttributes.Style := [fsBold];
RichEdit1.SelAttributes.Name := 'Comic Sans MS';
RichEdit1.Paragraph.Alignment := taCenter;
RichEdit1.Lines.Add('Add a line');
```

NOTE
The SelAttributes property of the RichEdit is available at run time only (i.e. it is not visible in the OI).

Remember when we use Random, we need to add Randomize in the OnActivate event of the Form.

btnDemoTabs

- `RichEdit1.Lines.Add (IntToStr(Random(99)) + #9 + IntToStr(Random(99)) + #9 + IntToStr(Random(99)));`
- Three random numbers are added seperated by tabs.
- When you click the button repeatedly, you will see three columns with numbers.

Please note: To add a *tab* to your text, just add #9 as indicated above.

When we use a RichEdit we can change the following:

- the size of the font
- the colour of the font
- the font itself (any font on your computer)
- the style of the font

```
[]; [fsBold]; [fsItalic]; [fsUnderline]; [fsBold, fsItalic]; [fsBold,
fsUnderline]; [fsItalic, fsUnderline];
```

- the alignment of the paragraph

```
taLeftJustify; taRightJustify; taCenter;
```

- the bulleting of the paragraph
  - To set Bullets: `RichEdit1.Paragraph.Numbering := nsBullet`
  - To remove Bullets: `RichEdit1.Paragraph.Numbering := nsNone;`

Refer to Appendix A for more information on the use of tabs in the RichEdit.

Constants

Besides variables we have another way of storing data in memory.

Sometimes you already know the value of a something when you create a program. This value can then be used in statements in the program. It is however a much better programming practice to store the value in a ***Constant*** and use the *constant* instead of the value.

A constant is declared as follows:

Const ConstantName = <value>;

Examples

```
Const VatRate = 0.14;
      MinimumBalance = 100;
      ProgramLanguage = 'Delphi';
```

Note that a = is used when declaring a constant.

The value of a true constant *remains the same* during the execution of the program. This means that a constant can be used in statements but never have values assigned to them. They can be seen as *read-only variables.*

Advantages of using constants

- A constant is declared with a descriptive name. Such names are easier to read and understand than numbers.

  Creating a constant called VatRate with a value of 0.14 is much better than simply using the value 0.14 in your code:

  ```
  TotalPrice := Price + (Price * VatRate);
  ```

  is easier to read and understand than

  ```
  TotalPrice := Price + (Price * 0.14);
  ```

- Constants make it easier to change values in your program.

  To change the VatRate in your entire program all you need to do is change the value of VatRate in the "Const" section and not everywhere that you have used the constant in calculations in the program.

Errors

Remember this section about errors.
You will need to refer back to this as your programming skills develop.

Runtime errors

A ***runtime*** error is what happens when your code has the correct syntax and so it compiles properly and runs. Then the user does something and your program suddenly crashes and stops working. When this happens a ***runtime error*** has occurred. Delphi handles these errors better than many other languages, so it does not always 'crash' your program. In fact, Delphi tries to help you by

- displaying an error message
- stopping the code that is running – for example the instructions that follow the error just won't happen
- keeping the program running so that it does not 'crash' and confuse your user. The user can then contact you, tell you what they were doing and what message appeared – which makes it easier for you to debug your program.

There are many different types of runtime errors, but some of the most common include:

- errors in type conversion (for example when you use ***StrToInt*** and the user has typed text into the Edit.)
- doing a calculation and dividing by 0
- doing a calculation that has an answer bigger than the variable type it is being stored in can handle (overflow error)
- errors in handling files (you will learn how to handle files in grade 11).

Tip
If the green Run Button is grey use ***Run, Program reset*** to enable you to run the program again - or <Ctrl><F2>.

Logical errors

The compiler cannot determine logical errors. A logical error occurs when the programmer makes a mistake in the way they think to solve the problem. This includes the whole process of the code – from input through processing to output. With this kind of error the compiler will not find any errors, the program will compile and run but it will not react as expected.

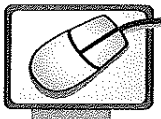
Common logical errors include:

- getting the ***sequence*** of your program instructions wrong (for example writing the calculation line BEFORE you write the input line)
- doing calculations but forgetting to assign the results to variables
- forgetting to display the results of your processing in a Label (for example your program works but does not appear to because you forgot to display the answers)
- writing code that does the correct calculations but forgetting to get the input from the Edits into the variables
- making complex decisions but using the AND, OR and NOT operators incorrectly (refer to Chapter 4)

- doing calculations that give the wrong answers because you don't take the order of precedence into consideration
- making loops infinite so that your program seems to hang (it is not hanging, it is just stuck in the loop and can't do anything else). (Refer to Chapter 5.)

Logical errors happen most often when

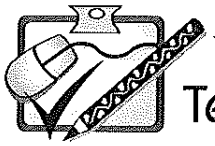
- you have not planned your program properly
- you make changes to existing code
- you do not properly understand the job that your program is supposed to be doing.



Activity

Learn more about errors

Look at the program called ***errors*** on the CD to see some examples of these types of error.



Test, Improve, Apply

Written exercises

1. Use the order of precedence of operators and determine the value of **rAns** for each of the following:

   a := 14.0; b := 2.5; c := 2;

   a) rAns:= a – c / a * c;

   b) rAns:= (a – c / a * c) * (a – c) / b;

   c) rAns:= a DIV c * b + Trunc(a * b);

2. Write the following mathematical calculations in Delphi code. Give the declaration of the variables as well as the assignment statements.

   a) Answer $= \frac{a+b}{c+d}$

   b) Result $= \frac{1}{2}(5x - 3y)$

3. Study the following Delphi declarations:

```
const  Word     = 'HELLO';
var    iNumber  : integer;
       rAns     : real ;
       cChar    : char;
       sLine    : string[20];
```

Determine if the following Delphi statements are valid.
If a statement is invalid, explain why and correct the statement if possible.

a) `rAns := iNumber + rAns;`
b) `Dec(rAns);`
c) `Word := sLine;`
d) `Random(iNumber);`
e) `iNumber := 100/iNumber;`
f) `sLine := IntToStr(rAns);`
g) `iNumber := Sqr(rAns);`
h) `rAns := Round (rAns) * 7;`
i) `rAns MOD 30 := iNumber;`
j) `iNumber := Inc(iNumber ,5);`

4. The following Delphi statements were written in an attempt to swop the values of two numbers:

```
iVal1 := sedVal1.Value;
iVal2 := sedVal2.Value;
iVal1 := iVal2;
iVal2 := iVal1;
```

When the values of **iVal1** and **iVal2** are displayed, they both have the same value. Why did this happen? How can the code be changed so the outcome is correct?

5. Study the following Delphi program segment:

```
var sName : string[20];
   cLett : char;
begin
   sName := cLett;    {1}
   cLett := sName;    {2}
end;
```

Explain why the following error message is displayed when compiling statement 2 and not at statement 1 as well.

`Incompatible types: 'Char' and 'ShortString'`

6. Give Delphi assignment statements for each of the following:
   a) Store the average of three integers in **rAvg.**
   b) Store the number of times an integer can be divided by 6 in **lAns.**
   c) Store the first and the last character of a name in **sAlpha.**

   Give the necessary declarations for the variables used.

7. A Delphi program has been written to do certain mathematical calculations. Study the statements that form part of the program and answer questions a, b and c for each statement. (Assume the variables have been declared with types according to the naming convention used e.g. **iValue** has been declared as **Integer.**)

Purpose of statement	Delphi code
1. Calculate one quarter of the sum of two values.	`iValue := sedA.Value + sedB.Value / 4;`

2. Calculate the average of 10 values.	`iCount := 0 ;`
	`iTotal := sedTotal.Value ;`
	`rAvg := iTotal / iCount ;`

3. Calculate the sum of two values	`iSum := sedValue1 + sedValue2 ;`
4. Calculate the square root of a value.	`rRoot := Sqrt(-sedValue.Value);`

a) Will the statement(s) compile without a syntax error?
b) Will the program perform the calculation without displaying an error message? (No runtime error occurs.)
c) Will the correct result be displayed? (No logical error has been made.)

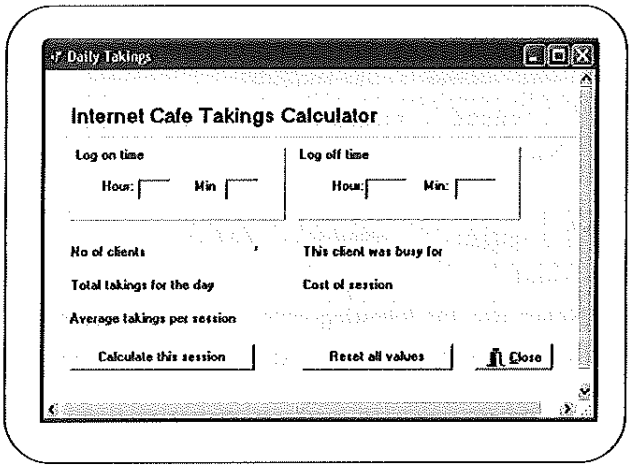
8. Complete the following program code so the variable **`iTens`** will contain the digit indicating the tens in a value, and the variable **`iHundreds`** will contain the digit indicating the hundreds in a value. E.g. if the variable **`iValue`** contains the value 584, **`iTens`** must contain the digit 8 and **`iHundreds`** the digit 5 once the code has been executed. You may only use the operators DIV and MOD.

```
iValue := sedValue.Value ;
iOnes := iValue MOD 10 ;
//..... Complete the code
```

Practical exercises

9. The hardware store sells wallpaper with a width of 500mm in rolls of 5m for R58.50. Write a program that will calculate how many rolls of wallpaper a person will need. Also calculate what the cost of buying the rolls would be. (You may decide what the input should be.)

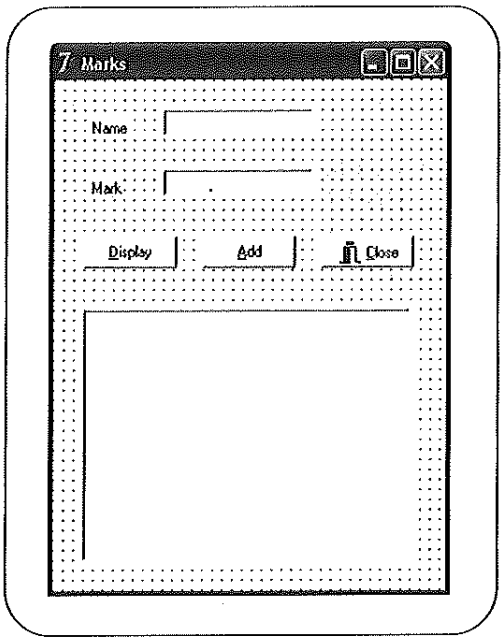
10. You need pocket money and decide to run an Internet Café from your home because you have 3 computers and an ISDN Internet connection. You charge R25 per hour - or part of an hour. (Please Note: If they have been on the computer for 2hrs 1min they pay for 3 hours!)
You decide you need a program to help you manage your daily activities in your business. It will help you calculate how much a client owes you, what your total income for the day was, what the average income per client was and it will record how many sessions of usage were made.
Create a form with the following layout and complete the program to do the necessary calculations for one computer in your Internet Café.



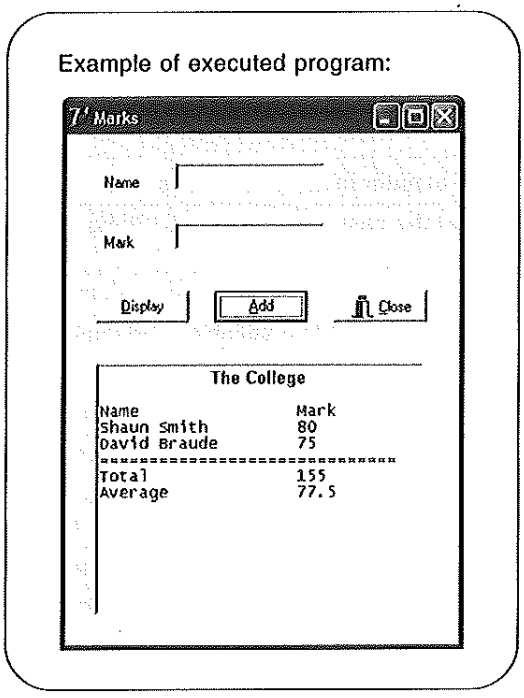
11. Currency Conversion Program
People who tour overseas often have a problem of working out how many Rand they need to pay for travelling and other expenses indicated in another currency such as dollars. They often know they need 1500 Dollars or Pounds or Euros spending money but they don't know how much money that is in terms of Rands. Let's write a general purpose program to help them. We need to ask them
- what the conversion rate is for their target currency (i.e. how many Rand do you need to buy 1 unit of that currency?)
- what their total travelling and hotel fees are (in the foreign currency)
- what their estimated spending money requirement is (in the foreign currency)

Then we show them in *Rand*
- their travelling and hotel fees
- their spending money
- the total amount they need.

12. Design the following user interface:



Example of executed program:



When the form is activated, the name of the school and the headings must be displayed in a RichEdit.

The following is an example of the code you need to enter in the FormActivate Event handler:

```
procedure TfrmMarks.FormActivate(Sender: TObject);
Const
   SchoolName = 'The College';
begin
   iTotal := 0;
   iCount := 0;
   redOutput.SelAttributes.Size := 10;
   redOutput.SelAttributes.Style := [fsBold];
   redOutput.SelAttributes.Name := 'Arial';
   redOutput.Paragraph.Alignment := taCenter;
   redOutput.Lines.Add(SchoolName);
   redOutput.Lines.Add('');
   redOutput.Paragraph.Alignment := taLeftJustify;
   redOutput.Lines.Add('Name' + #9 + 'Mark');
   redOutput.Lines.Add('');
   edtName.SetFocus;
end;
```

Remember since **iTotal** and **iCount** are used in more than one procedure, they need to be declared as global variables.

When the name and the mark of a learner is entered, the [Display] Button must be clicked to display the name and the mark of the learner in the RichEdit.

When the all the names and marks are entered, the [Calculate] Button must be clicked. The Total and the Average must then be displayed.

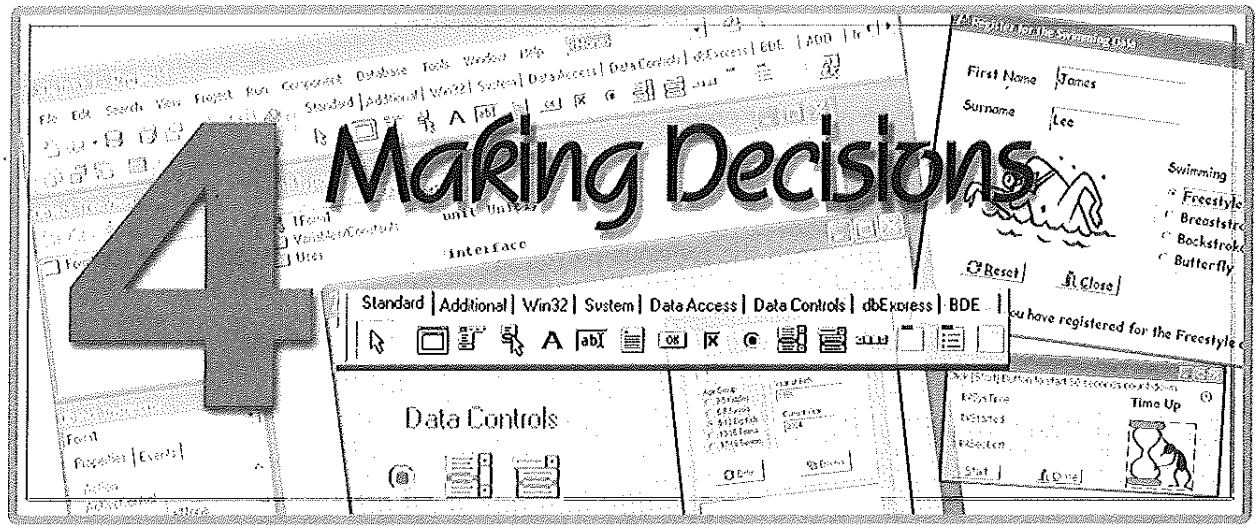
Tip

If you have trouble with spacing the names and marks neatly, refer to Appedix A to find out how to use tabs in the RichEdit component.

Test yourself

13. Check that you know / can do the following:

		✓
Knowledge	**I can explain in my own words**	
	what the basic data types are (String, Integer, Real, Char and Boolean) and when each type would be used in Delphi	
	what range of values can be stored in the various data types	
	what a variable is and how it can be assigned values and used in program	
	how the assignment statement works	
	explain how and why integer and real values are converted to a string and vice versa	
	what an operator is and which operators are available in Delphi that work on numeric values	
	what the difference is between integer and real division	
	what the difference is between the mathematical functions Trunc and Round	
	how and where the Random function and the Randomize procedure is used	
	what the difference is between a variable and constant	
	what the Uses and Implementation sections of a unit are used for	
	the difference between logical and run-time errors by referring to typical examples.	
Skills	**I can**	
	create an IPO table as part of the planning of a program by identifying what input, processing and output is required	
	apply the rules for the declaration of variables and their naming rules, as well as using meaningful variable names	
	apply the order of precedence in the evaluation of a mathematical expression	
	convert between numeric and string values, using the IntToStr, FloatToStr, StrToInt, StrToFloat and FloatToStrF functions	
	display the various data types on the screen	
	use the basic mathematical operators (+ - * / DIV MOD) and functions (Trunc, Round, Frac , Sqr, Sqrt , Pi, Random, RoundTo and Power) in a Delphi program	
	investigate which other mathematical functions are available in Delphi by consulting the on-line help functions	
	generate random numbers in a specified range for use in a Delphi program	
	declare and use constants in a program	
	use the appropriate properties and methods of a RichEdit to produce neat, formatted output	
	use the appropriate properties of a SpinEdit to read in an integer value in a program	
	write programs that can count the number of times data is input by the user and then perform simple calculations such as finding the sum and average of a series of numbers	
	detect common logical and run-time errors in programs.	



After you have completed this chapter you should be able to

- use the IF-statement for decision making in programs
- use mathematical functions in conjunction with the IF-statement in the solution of problems
- make use of the RadioGroup and CheckBox to allow users to indicate their choices
- specify multiple conditions in an If-statement using AND, OR and NOT
- make use of nested IF-statements
- make use of sets
- use the CASE statement to specify a set of conditions and corresponding statements
- make use of the Timer and the MaskEdit

Introduction

So far all the problems we have tackled have involved writing Event handlers with code where each instruction executes one after the other in the order that it was written. This is known as sequential processing.

Not all problems can be solved by simply executing all the instructions from top to bottom each time in the same order. Sometimes we require certain steps only to be done under certain circumstances.

Here are a few examples:

- A learner can be promoted to the next grade if his marks are good enough.
- The symbol assigned to a learner on his report will be based on the mark obtained by a learner.
- If the user types in the correct password, the program will continue.
- If you have a higher score than the previous player, your name will be placed at the top of the list in a game.
- If I buy a cellphone before the end of the month, I will get a 10% discount.

The If statement

The most common instructions for making decisions in programming are almost the same as the words we use to describe making a decision in English. This instruction is called the ***If*** statement.

If .. Then statement

In English we say:

If my mark is 80% or more *then* I will get a distinction.

In Delphi the statement will be:

```
if iMark >= 80
 then lblMessage.Caption := 'Distinction';
```

The condition

The instruction that is executed if the condition is true.

The general structure of the If..Then statement is

```
if <condition(s)>
   then
      begin
         <statement(s)>
      end;
```

The statements(s) between the Begin...End are only executed if the condition is **True.**
If the condition is **False** then the statements between the Begin...End are *skipped over* and not executed.

Let's look at two examples.

```
if iMark >= 40
 then lblMessage.Caption := 'Pass';
```

Note

If only one statement needs to be executed if the condition is true then it does not need to be placed between a **Begin** and **End.**

```
if iAge < 15
 then
   begin
     inc(iNumberTeenagers);
     lblMessage.Caption := 'Report to Gate 6';
   end;
```

Note

When a number of statements need to be executed as a result of a condition, it is known as a *compound statement*. These statements must be bound together as a unit with a **Begin...End** pairing.

If .. Then .. Else statement

Sometimes it is necessary for one statement to be executed when the condition is *true* and *another* statement to be executed if the condition is *false*.

Example:

When a mark is 40% or above, the word 'Pass' must be displayed or if the mark is less than 40% the word 'Fail' must be displayed.

This can be achieved with the following two If statements:

```
if iMark >= 40
 then lblMessage.Caption := 'Pass';
if iMark < 40
 then lblMessage.Caption := 'Fail';
```

It is better to combine these two statements with the use of an **If..Then..Else** statement:

```
if iMark >= 40
 then lblMessage.Caption := 'Pass'
 else lblMessage.Caption := 'Fail';
```

If the condition is True then the statement after the *Then* will be executed.

If, however, the condition is False, then the statement after the *Else* will be executed.

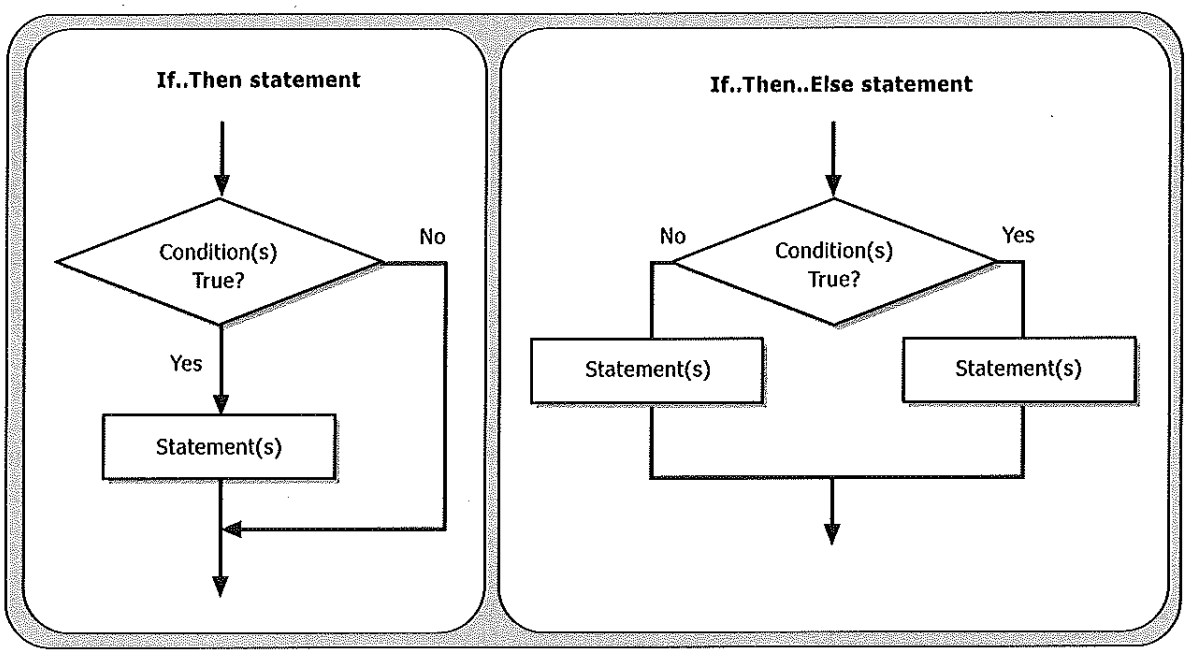
The general structure of the If...Then...Else statement:

```
if <condition(s)>
  then
    begin
      <statements(s)>
    end
  else
    begin
      <statements(s)>
    end;
```

Note

A semi-colon is used to indicate the end of a statement – therefore *no semi-colon occurs before the Else*. The End of the If statement occurs only *after the Else part*.

Flowcharts:



The Boolean Condition

All the conditions that you encounter in programming can be transformed to true/false situations. We need to make comparisons between values in order to construct these conditions. Consider the following table, which shows the types of relational operators that we can use.

Example of condition	**Relational Operator**	Meaning	Implications	
			True	False
Value < 40	**<**	Less than	Value is less than 40	Value is greater than or equal to 40
Total >= 1680	**>=**	greater than or equal to	Total is greater than or equal to 1680	Total is less than 1680
A > B	**>**	greater than	A is bigger than B	A is either equal to or less than B
Score <= High_Score	**<=**	less than or equal to	Score is less than or equal to the High Score	Score is bigger than the High Score
Password = '007'	**=**	equal to	Password is '007'	Password is not '007'
A <> B	**<>**	not equal to	A is not equal to B	A is equal to B

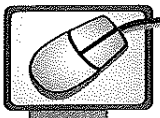
These *True/False* situations are termed *Boolean conditions* - named after George Boole who used them originally.

Tip: it is advisable to always see a Boolean condition as a question.
(Mark >= 80) - True or False?
(Password = '007') - True or False?
(A <> B) – True or False?

Note
It is very important to note that we use the "=" sign in a Boolean expression and not the assignment operator ":="

Note the following:
The condition can also contain a mathematical expression, for example

```
if iValue MOD 2 = 0
  then lblMessage.Caption := 'Value is divisible by 2';
```



Using the If..Then statement

Activity

Design a program, which determines if a learner qualifies for the Einstein bursary for the following year. To qualify for this bursary, the learner's average (rounded) for Maths and Science must be 90% or higher on his or her final report.

Step 1: Set up the IPO table

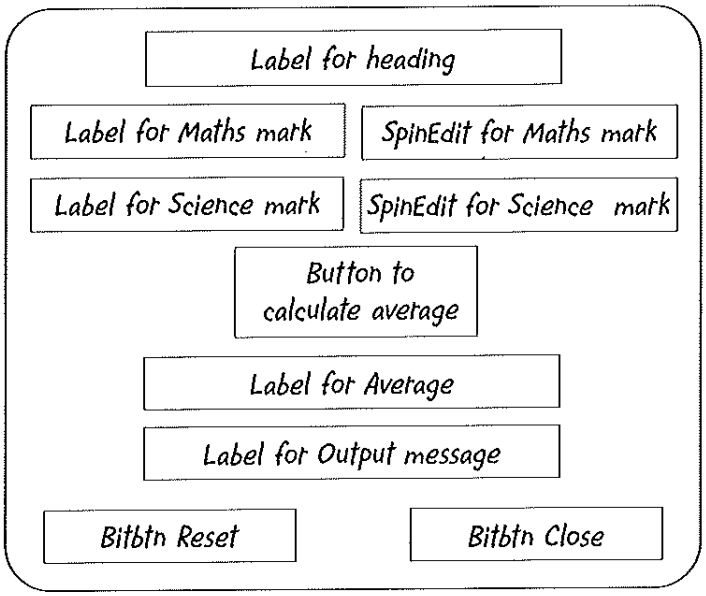
Input	*Processing*	*Output*
Maths and Science marks	*Average = round((Maths+Science)/2)* *If Average >= 90* *then the user qualifies for the bursary*	*Average* *Appropriate message*

Step 2: Choose and declare variables

```
var
  iMaths, iScience, iAverage    : integer;
```

Step 3: Design the user interface

Decide which components are needed:



Step 4: Code of the Event handler btnAverage:

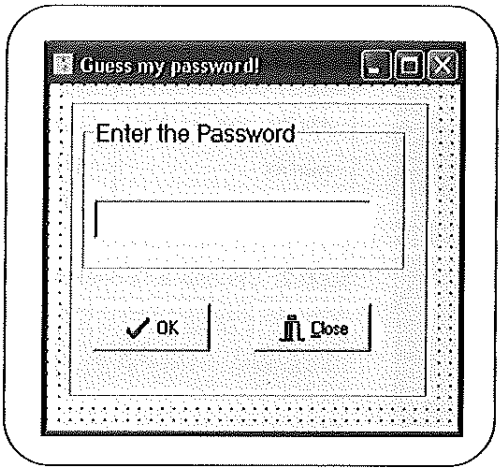
```
begin
  iMaths      := sedMaths.value;
  iScience    := sedScience.value;
  iAverage    := Round((iMaths + iScience)/2);
  lblAve.Caption := IntToStr(iAverage);
  if iAverage >= 90
     then lblMessage.Caption := 'Qualify for the Einstein bursary';
end;
```

Also code the Reset BitBtn and save and test your program.

Activity

Using the If..Then..Else statement

You may have already used programs where you need to type in a password. Let's create one with an interface that looks something like the one shown here.



We want the user to be able to type in a password in the edit control. When they click the OK BitBtn, a suitable message must be displayed if their password was correct. If, however the password is incorrect, we then want to clear the Edit, display an error message and give the user another chance to type in the password.

The IPO table

Input	*Processing*	*Output*
Password	*If Password is correct* *Then display a suitable message* *Else clear the edit component* *also display an error message*	*Message* *Error message*

You may use a password of your choice.

Make sure that you add a Begin and End if you make use of compound statements.

Checkpoint

Design and write a program for each of the following.

1. Type in a person's name and their age in years. Display the message '15% discount' for senior citizens if the person is older than 65. Display the message 'Full price' if the person is not a senior citizen.
2. A golf day is organised by your school. The girls tee off at the 8th hole and must therefore meet there. The boys tee off at the 4th hole. The program must read in the name and gender of the participant. A welcoming message and indication as to where they must report must be displayed.
3. Determine the average of the marks of the failures and the average of the marks of the passes as the user types in a list of marks for a test. The pass mark is 50%.
4. A company is organising a fun day for their clientele. People need to register as they arrive. The company wants to determine how many children (younger than 13 years) and how many adults (13 years and older) are at the fun day. Read in the name of the person, as well as their age. A message needs to be displayed on the screen of how many children and how many adults have arrived. Every time a new person types in their name and age, a welcoming message must be displayed and the numbers must be adjusted accordingly.

Problem Solving

In problem solving, we need to describe step-for-step what the computer must do. Bear the following in mind:

- Solve the problem yourself and take note of the steps you follow.
- Analyse the steps you have followed and break them down into computer language instructions.

Try to think like a computer:

- Remember that a computer needs variables to hold and "remember" data to do any work at all.
- Work out what variables you need in order to solve the problem.

Let's look at a few basic problems, which can be solved with the aid of a computer program:

Determine the largest and smallest

We often need to find the largest or smallest value in a series of numbers.

If we think carefully what needs to be done, then the **IPO table** would look something like:

Input	Processing	Output
Values	The first value is initially both the Largest and the Smallest. (For each value thereafter) If the value > Largest then this value is now the Largest. Similarly If value < Smallest then this new value is the Smallest.	Largest Smallest

Let's look at the **variables** we need:

- a variable to hold the current number
- two variables to hold the largest and smallest values respectively
- a counter variable.

We can describe the **algorithm** as follows:

Set the counter to 0.

This must only be done once when the user activates the program.

Every time the user enters a value and clicks the [Process] Button, the following must happen:

- Get the number from the Edit and assign it to the number variable.
- If the counter is 0 then

This if is to see whether it is the first value being entered

  - Assign the number to ***largest*** and ***smallest*** (it is the first number the user is entering, so it has to be the largest *and* the smallest number!)
  - Increment counter
  - Clear the Edit and put the cursor in the Edit.

These statements will only be executed if count=0.

- If the counter > 0 then
  - if ***number*** > ***largest*** then Largest must get the value of the number
  - if ***number*** < ***smallest*** then Smallest must get the value of number
  - Clear the Edit and put the cursor in the Edit.

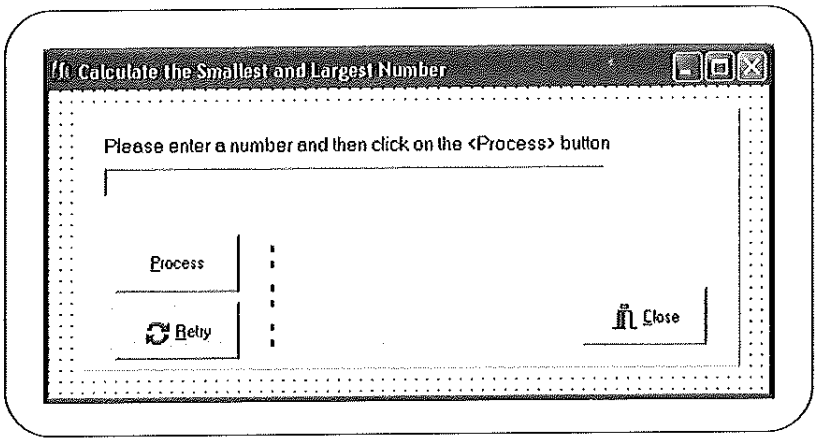
These statements will be done every time a new number is entered after the first number.

Hint
Longest and smallest must be declared as global variables.

Activity

Find the largest and smallest number

Write a Delphi program to determine the largest and smallest of a number of values. Consider the following example of the user interface:



Using Mathematical functions in combination

Mathematical functions are used in conjunction with one another in the following situations to solve problems. This highlights an important concept in problem solving: Always think in terms of what the computer already can do so that you can teach it something new!

Checking if a number is a perfect square

A perfect square is a number where the square root of the number is a whole number (an integer). How do we use the computer to determine if a number is a perfect square?

We already know the Trunc function. If we combine decision-making, Trunc and Sqrt then we can solve this problem.

The code to solve the problem is:

```
iNo := StrToInt(edtNo.Text);
if Sqrt (iNo) = Trunc (Sqrt (iNo))
  then lblAns.Caption := 'Yes, the number IS a perfect Square'
  else lblAns.Caption := 'NO, the number is NOT a perfect Square';
```

If the square root of a number is the same the square root of the number with the fraction part truncated, then we now that the number is a perfect square!

- Let's test this:
  (We can make use of a table where we indicate all the variables, all the conditions, as well as the output. This is known as a ***trace table.***)

iNo	Sqrt(iNo)	Trunc(Sqrt(iNo))	Sqrt(iNo) = Trunc(Sqrt(iNo)) ?	Output
12	3.464	3	No	No, the number is NOT a perfect square
16	4.0	4	Yes	Yes, the number IS a perfect square

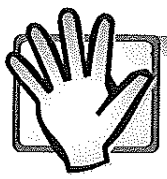
Testing to see if a number is a factor of another number

To solve this problem we need to ask ourselves what a factor is. When we realise that it is a number that divides into the number we have been given without leaving any remainder, we have our first clue as to how to make the computer solve this problem.

Do we know any commands in Delphi that can be used to find the remainder when division is done?

Yes – we can use MOD. Remember, DIV and MOD – like all functions - can be used in all sorts of creative ways! There is an example below that uses MOD to check if 3 is a factor of a number that the user enters:

```
iNo := StrToInt(edtUserNo.Text);
 if iNo Mod 3 = 0
   then lblModResult.Caption := 'Yes, 3 is a factor of ' +
                                edtUserNo.Text
   else lblModResult.Caption := 'NO, 3 is NOT a factor of ' +
                                edtUserNo.Text;
```



Checkpoint

Design and write a program for each of the following:

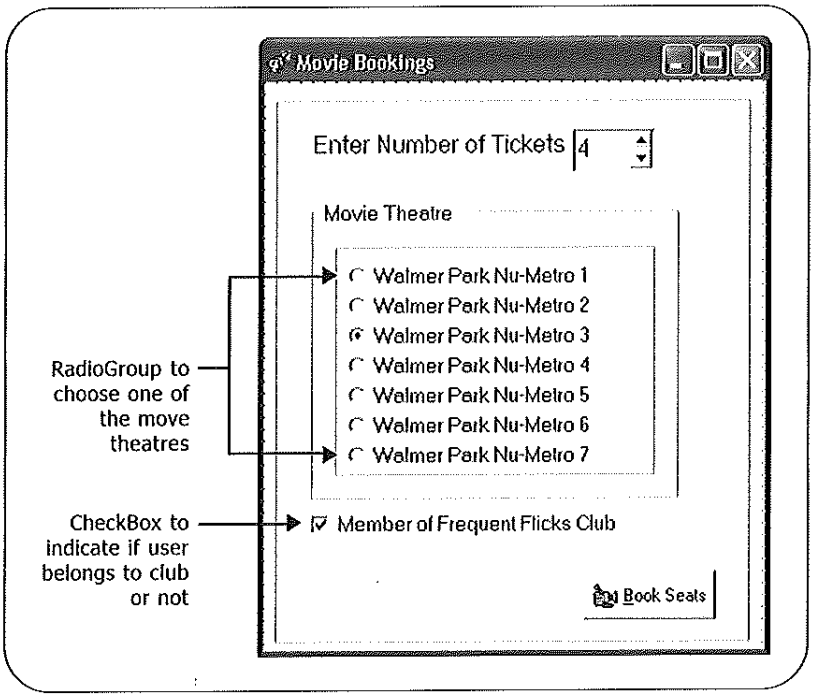
1. Read in a value and determine if it is a real (decimal) value or not.
2. Test if a number that is entered by the user is even or odd (Hint: There is an Odd function but not an even function in Delphi! Remember: An even number is divisible by 2 without a remainder)
3. Read in a number of integer values and determine the difference between the largest and smallest of these values.
4. Read in two integer values and check if either is a factor of the other.
5. Read in a quantity of T-Shirts purchased. They are priced at R 60 each. The store is having a two-for-one special sale. For every two purchased, you get one free. This is subject to at least 4 T-shirts being purchased; otherwise the normal cost of R 60 per T-shirt applies. The total number of T-shirts, free T-shirts and total cost must be displayed.

Getting the user to indicate choices visually

Programs start to become more powerful when they can make decisions based on the user's input. Windows is a graphical environment, so it makes sense to offer the user choices in a more graphical way than just asking them to type something into an Edit.

There are two types of choice that a user can make, namely

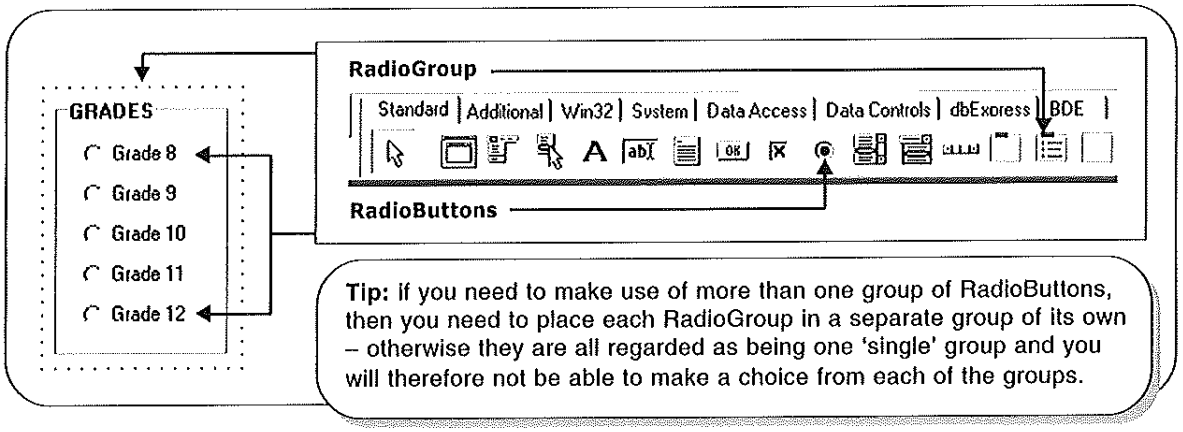
- Select one item from a whole range of options. This is like a radio station – you can only select one to listen to at any time. That is why the component is called a ***RadioGroup.***
- Select as many options as you like from a range of options. This is like a shopping list. You can select one, many or no options, simply by putting a tick next to it. The component is called a ***CheckBox.***



RadioGroup

It is easier for a user to just click on a Button to make a choice, instead of having to enter data. **RadioButtons** are used to allow the user to make *only one* choice from a number of available options. RadioButtons are frequently grouped in a RadioGroup (TRadioGroup).

The abbreviation for a RadioGroup is **rgp** and the abbreviation for a RadioButton is **rad.**

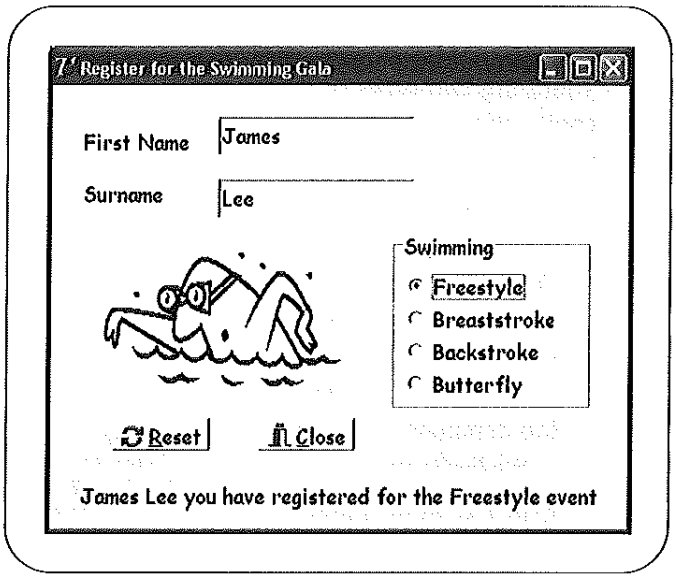


Tip: If you need to make use of more than one group of RadioButtons, then you need to place each RadioGroup in a separate group of its own – otherwise they are all regarded as being one 'single' group and you will therefore not be able to make a choice from each of the groups.

Activity

Exploring RadioButtons and RadioGroups

Write a Delphi application that prompts the user to enter his or her name and surname. The user can then choose which event he or she wants to register for. Depending on the event an appropriate message will be displayed.



The IPO table would look something like the following:

Input	*Processing*	*Output*
Choice of Activity	*Depending on the input, form an appropriate response*	*Suitable message*

Decide which variables you need to use and complete the user interface.

The RadioGroup and RadioButtons can be added as follows:

1. Place a RadioGroup on the form from the Standard Palette.
2. Find the Items property for the RadioGroup in the Object Inspector and double-click the ... shown to the right of the [TStrings] in the Value column and the String List editor will be displayed.
3. Type the *Captions* for each of the four RadioButtons as shown and click **OK** to close the String List editor.
4. Change the Caption property as needed.

To complete the rest of the code:

In this program, an event occurs if the user clicks on one of the RadioButtons. The RadioGroup has an OnClick event and it is for this event that we will write the Event handler.

The RadioGroup has an ItemIndex property that can be used by you to determine which item in the group was selected (in other words, which RadioButton has been clicked). The RadioButtons are, in order, numbered from zero onwards, so if the value if the ItemIndex property is a 0, then the first RadioButton has been selected. Similarly, if the value of the ItemIndex property is a 2, then the third RadioButton has been selected.

The value of the ItemIndex property is set to –1 by default to indicate that no options are initially selected. The current value of the ItemIndex property is read to determine which of the options was selected.

```
if rgpActivity.ItemIndex = 0
  then lblOutput.Caption := edtName.Text + ' ' + edtSurname.Text +
          ' you have registered for ' + rgpActivity.Items[0];
if rgpActivity.ItemIndex = 1
  then lblOutput.Caption := edtName.Text + ' ' + edtSurname.Text +
          ' you have registered for ' + rgpActivity.Items[1];
........
```

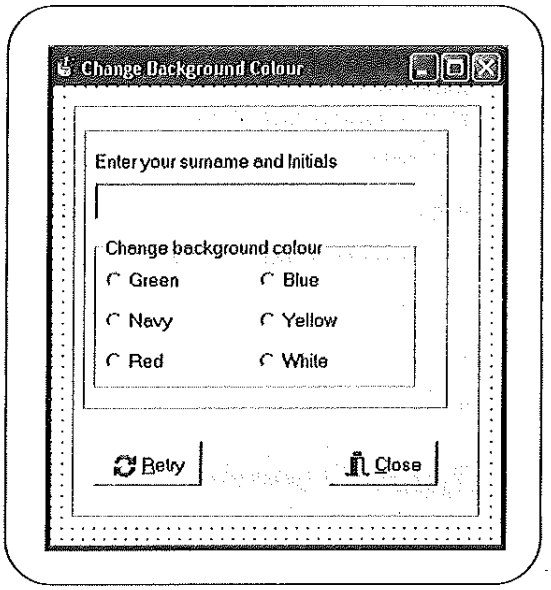
Tip: This code could be greatly simplified as the ItemIndex property can be used in conjunction with the Items property of the RadioGroup as follows:

```
iSelection := rgpActivity.ItemIndex;
lblOutput.Caption := edtName.Text + ' ' + edtSurname.Text +
        ' you have registered for ' + rgpActivity.Items[iSelection];
```

If you want to, you can add an extra OK BitBtn to the form, which will allows the user to change their choice, before processing takes place.

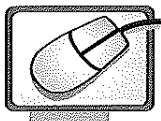
Checkpoint: Try the following yourself

- Create the following interface.
- Now write OnClick Event handlers for the RadioButtons. When the user enters his/her surname and initials, they can click one of the RadioButtons. The background colour of the Edit must then change accordingly.
- Code the Reset BitBtn to clear the contents of the Edit and "uncheck" the RadioButtons.



The CheckBox

A CheckBox allows the user to easily select or de-select an option. It is controlled by the ***Checked*** property, which can either be *true* or *false*. If Checked is *true* then a small 'tick' is displayed in the box part of the component to show the user that the option has been selected. Each time the user clicks on the component the Checked property toggles between *true* and *false*. We use the naming convention of *cbx* to prefix a CheckBox name. Typical code involving a CheckBox would look something like: *If cbxMember.Checked Then...*

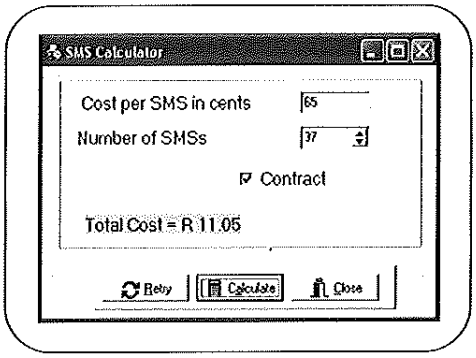


Activity

Using a CheckBox

Write a Delphi application that prompts the users to enter how many SMS messages they sent in a month and the cost per SMS message. The program simply needs to calculate the cost of SMS messages for the month.

We now want to add a CheckBox so the users can indicate whether they are on a cell phone contract or not. If they are on a contract, they get their first 20 messages free.



Specifying more than one condition

We often need to use *more than one condition* in our choices.

Examples
If *it is raining* ***and*** *I am going outside* then I must wear a raincoat.
If *I have a cough* ***or*** *I have a runny nose* then I should see the doctor.

We use these kinds of complex conditions every day without thinking about them.

The *operators* that we use in programming to construct these complex decisions are:

- AND
- OR
- NOT

The AND operator

The AND operator implies that *all the conditions* must be True for the result to be True and the statement(s) to be executed.

Example: To vote you must be over 18 AND be a South African citizen AND you must be registered.

General structure in Delphi:

if (<condition 1>) AND (<condition 2>) AND (<condition 3>)
then ..

Examples:

```
if (iAge >= 18) AND (bSACitizen = True) AND (bRegistered = True)
 then lblMessage.Caption := 'You may vote';
if (iAge >= 12) AND (iAge <= 18)
 then sCategory := 'Teenager';
```

NOTE

- Each condition must be enclosed in round brackets.
- If any one of the conditions is False, then the whole condition is False and the Then part will not be executed.

The OR operator

On the other hand OR requires that *only one of the conditions* be True for the overall condition to be True and the statement(s) to be executed.

Example: You can be disqualified from voting if you are under 18 OR you are not registered OR you are not a South African citizen. (Any one of the conditions would disqualify you from being eligible to vote.)

General structure in Delphi

if (<condition 1>) OR (<condition 2>) OR (<condition 3>)
then ..

Examples:

```
if (iAge < 18) OR (bSAcitizen = False) OR (bRegistered = False)
 then lblMessage.Caption := 'You may not vote';
if (iAverage < 40) OR (iFailed_Subjs > 1)
 then iFailed := iFailed + 1;
```

NOTE

One of the problems that we have to watch out for is that we often use AND and OR interchangeably and often in the wrong places in every day conversations, as *we* know what we mean, but remember it has to be spelt out to the computer.

We may incorrectly say: Will the learners who are in grade 10 *and* grade 11 meet at break time.

```
if (iGrade = 10) AND (iGrade = 11)
 then lblMessage.Caption := 'Attend Meeting'
 else lblMessage.Caption := 'No need to attend meeting';
```

The caption will *always* say **No need to attend meeting** because a learner cannot be in grade 10 *and* 11 at the same time!

What we mean to say is: Will the learners who are in grade 10 ***or*** grade 11 meet at break time.

```
if (iGrade = 10) OR (iGrade = 11)
  then lblMessage.Caption := 'Attend Meeting'
  else lblMessage.Caption := 'No need to
    attend meeting';
```

The NOT operator

The NOT operator *returns the opposite* of the Boolean expression it acts on, i.e. True for False and False for a True condition.

General structure in Delphi

if NOT (<condition(s)>)
** then .....**

Examples:

```
if NOT (iAverage >= 40)
  then lblMessage.Caption := 'FAIL';
if NOT (cGender = 'M')
  then lblMessage.Caption := 'Welcome Ladies';
```

You could also have said:

```
if iAverage < 40
  then lblMessage.Caption := 'FAIL';
if cGender <> 'M'
  then lblMessage.Caption := 'Welcome Ladies';
```

TIP

We can use the operators in any combination to allow our program to make decisions.In general try to avoid mixing AND and OR operators in an If statement if you can. Very often this leads to "faulty" logic as the AND part of the condition will be tested before the OR part, much like the * and / before the + and – is in Delphi arithmetic.

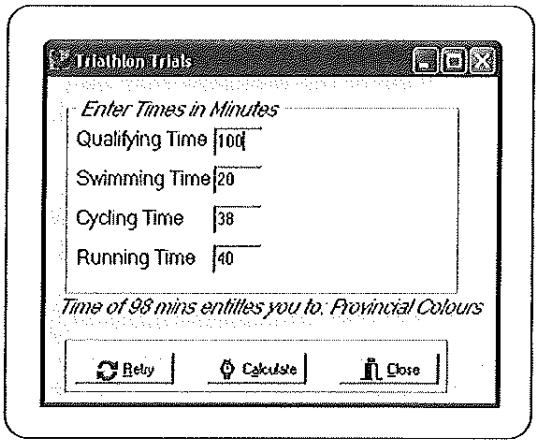
Activity

Work with more than one condition

We want to design a program that reads in the times (in minutes) for the three disciplines of the triathlon, namely swimming, cycling and running, as well as the qualifying time (QT).

(Assume that the user will input valid values for the times and will also input all four values for the times before they click on the [Calculate] Button.)

The program must calculate and display the total time taken to complete the triathlon. Also display the award the athlete receives according to the table on the following page.



Total Time	*Award*
Under QT	Provincial colours
Within 5 minutes of QT	Provincial Half-Colours
Within 10 minutes of QT	Provincial Scroll
10 minutes or more than QT	Participation Certificate

The IPO table

Input	*Processing*	*Output*
QT and three times in minutes	*Determine Total Time Taken (TT)* *Determine award based on TT:* *If TT < QT then Award = 'Provincial colours'* *If TT >= QT and TT <= QT + 5* *then Award = 'Half colours'* *If TT > QT + 5 and TT <= QT + 10* *then Award = 'Scroll'* *If TT > QT + 10* *then Award = 'Certificate'*	*Message displaying Total time and appropriate award*

(It should be easy to display the total in hours and minutes instead of just minutes by making using of the DIV and MOD operators.)

Checkpoint

Your school needs to choose its student leaders for next year. The grade 11 and grade 12 learners have cast their votes. The number of learners in grade 11 and 12, the number of votes the learner receives from grade 11 learners, as well the number of votes received by the grade 12 learners are read in. The votes for the learners from the grade 11 and 12 groups are combined and the learner's chances of becoming a student leader are then classified in terms of *definitely*, *probable*, *possible* or *unlikely*.

The criteria used to classify their chances are as follows:

Classification	*Criteria*
Definitely	60 % or more of the vote
Probable	Less than 60% but more than 47% of the vote
Possible	From 40%- 47% of the votes
Unlikely	Less than 40% of the vote

Create a program to assist your school where they can type in the name of the learner, the numbers of grade 11 and 12 learners, the number of grade 11 votes and the number of grade 12 votes. The percentage of the total vote obtained by the learner must then be utilised to output a message displaying the learner's chances of becoming a student leader using the criteria above.

Nested If Statements

A nested If statement occurs where the Then or Else parts of an If statement contains *another* If statement.

Example 1

Let's say we want to display a message indicating the size of two numbers A and B relative to each other. Essentially there are three possibilities:

They are either equal *or* A is bigger than B *or* B is bigger than A.

We could program this as follows using three separate If statements:

```
if A > B
 then lblOutput.Caption := 'A is bigger than B';
if  B > A
 then lblOutput.Caption := 'B is bigger than A';
if A = B
 then lblOutput.Caption := 'A is equal to B';
```

or we can use nested If statements:

```
if A > B
 then lblOutput.Caption := 'A is bigger than B'
   else if B > A
    then lblOutput.Caption := 'B is bigger than A'
    else lblOutput.Caption := 'A is equal to B';
```

TIP

You can use the nested If to make your life a lot easier by using the fact that the last set of statements is only executed if none of the previous If conditions are true. In this way you do not have to specify what the conditions for the last set of instructions have to be.

Example 2

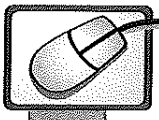
Let's say a program has to assign a symbol based on a mark, where an "A" represents 80% - 100% , a "B" represents 70% - 79% etc. We can use the following code.

```
if (iMark >= 80)
 then cSymbol := 'A';
if (iMark >= 70) AND (iMark <= 79)
 then cSymbol := 'B';
if (iMark >= 60) AND (iMark <= 69)
 then cSymbol := 'C';
 ..........
```

This code could be greatly simplified with nested If statements.

```
if iMark >= 80
    then cSymbol := 'A'
    else if iMark >= 70
           then cSymbol := 'B'
           else if iMark >= 60
                  then cSymbol := 'C'
                  else .....
```

Note: We will see later in this chapter that we can use a Case statement to express this code in a far easier and more elegant fashion.



Using a nested If

Activity

We want to create a program to read in the lengths of three sides of a triangle and then decide if the triangle is *equilateral* (all sides are equal) or *isosceles* (two sides equal) or *scalene* (no two sides are equal).

There are a number of different ways to implement the test. Consider the following approach:

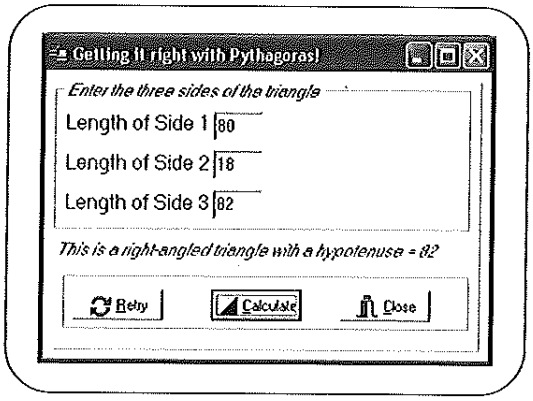
Input	Processing	Output
Sides A, B and C of the triangle	If all sides are equal then triangle is equilateral else If any two sides are equal the triangle is isosceles else the triangle is scalene	Appropriate Message

Checkpoint: Try the following yourself

Design a program to read in the lengths of the three sides of a triangle (in any order). The program must decide if these sides could represent the sides of a right-angles triangle or not. If it is a right-angled, triangle, the length of the hypotenuse must also be displayed.

Remember according to Pythagoras for a triangle to be a right-angled triangle, the square of the longest side must equal the sum of the squares of the other two sides. In other words: $C^2 = A^2 + B^2$, where C is the longest of the three sides.

If we think about it, there are basically only four possibilities namely the first side entered is the longest side and the triangle is right-angled, the second side entered is the longest side and the triangle is right-angled, the third side entered is the longest side and the triangle is right-angled or the triangle is not right-angled.



Simplifying decision making with sets

An ***ordinal data type*** is a data type in which the values are arranged according to a natural order.

- The Boolean, Char and Integer types are considered to be ordinal types.
- Real and String types are not considered to be ordinal types as they have no natural order.

A ***subrange*** is a sub-set of an ordinal type and is indicated with the following syntax:
<lower bound> .. <upper bound>

Example: 1..10 is a sub-range of the Integer type.

A ***set*** is a collection of values of the same ordinal type. It can contain either no values or one or more values. The values contained in the set are indicated in square brackets.

Examples of sets:

- [1..6] meaning 1 , 2 , 3 , 4 , 5 and 6.
- ['a', 'e', 'i', 'o', 'u']

We can use sets together with the IN operator to simplify our If statements quite dramatically.

```
if (iCount = 1) OR (iCount = 2) OR (iCount = 3) OR (iCount = 4)
 then Inc(iTimes);
```

can be simplified to:

```
if iCount IN [1..4]
 then Inc(iTimes);
```

Note that in a subrange of integers, the values to be tested have to be in the range [0..255]. Anything outside of this range will cause an error to occur.

The IN operator tests if the data is contained in the set or not:

```
if (iMonth IN [4, 6, 9, 11]) AND (iDays > 30)
 then lblOut.Caption := 'Date is wrong';
```

We can specify multiple subranges in a set, provided they are of course of the same data type:

```
if cLetter IN ['A'..'Z', 'a'..'z' ]
 then ShowMessage ('Character is a letter of the
                    alphabet');
if NOT (iOption IN [10..20, 1..5] )
 then ShowMessage ('Option must be from 10 to 20 or
                    1 to 5');
```

A Message box is a Delphi form that appears on top of a running application to provide information. The easiest way to create a Message box is to use the *ShowMessage* command.

The Case Statement

We very often wish to test a variable's value and then according to some criteria or conditions, do various things depending on the value of the variable. The more conditions we have, the more complicated the If statement will become.

A Case statement is a better option if we have a lot of conditions of which only one can be true.

Let's revisit the example where a symbol had to be determined based on a mark. The Delphi code using If statements looked something like:

```
if (iMark >= 80)
  then cSymbol := 'A'
  else if (iMark >= 70)
         then cSymbol := 'B'
         else if (iMark >= 60)
                then cSymbol := 'C';
                ..............
```

A Case statement can be used instead of using the nested If statements. The code would look something like:

```
case iMark of
    80..100 : cSymbol := 'A';
    70..79  : cSymbol := 'B';
    60..69  : cSymbol := 'C';
    50..59  : cSymbol := 'D';
    40..49  : cSymbol := 'E'
  else cSymbol := 'F'
end;
```

The variable or expression which is compared to all the possible conditions.

A list of all the conditions to be tested, each with a corresponding 'result' or set of instructions to be executed if a 'match' is found.

The general structure of a Case statement is:

```
case  <Selector Expression> of
   <case 1>  :   <statement(s)>;
   <case 2>  :   <statement(s)>;
   <case 3>  :   <statement(s)>;
   ..................
   [else <statement(s)> ]
end;
```

The following points should be noted:

- The Selector Expression being evaluated must be *ordinal* in type
- Each group of values represented must be unique in the Case statement
- A Case statement can have a optional else part, but it is useful to "trap" the exceptions that are not catered for by the preceding Cases.
- *There is an End but no matching Begin.*
- When a Case statement is executed, at most one of its statements is executed.

Examples of applications of the Case statement:

Example 1

```
case iAge of
     0..12      : Inc(iChildren);
     13..18     : Inc(iTeenAgers);
     19..59     : Inc(iAdults);
     60..120    : begin
                    Inc(iAdults);
                    Inc(iSeniors);
                  end
   else ShowMessage('Please check age ')
end;   // end of case
```

Example 2

The Case statement is often used in conjunction with RadioGroups:

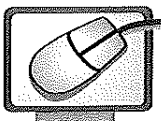
```
procedure TfrmColours.rgpColoursClick
begin
 case rgpColours.ItemIndex of
      0 : frmColours.Color := clRed;
      1 : frmColours.Color := clGreen;
      2 : frmColours.Color := clBlue;
      3 : frmColours.Color := clYellow
   else lblOutput.Caption := 'No option selected yet'
 end; // end of case statement
end; // end of procedure
```

Example 3

The Case statement can be used in helping to validate data:

```
case iMonth of
     4,6,9,11          : iDays := 30;
     1,3,5,7,8,10,12   : iDays := 31;
     2                 : begin
                           iDays := 28;
                           if isLeapYear(iYear)then iDays := 29;
                         end
 else Beep; //makes a beep sound on the speaker of the PC!
end;
```

As you have seen, the Case statement simply makes the testing process easier. The Case statement can always be replaced by nested If statements, but it is far more time-consuming and not as easy to read.



Activity

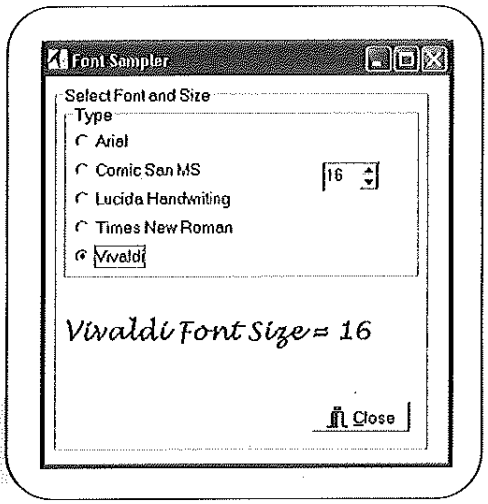
Using Case

We want to design a program that allows the user to view what different combinations of font types and sizes would look like. (Use the fonts of your own choice!)

A RadioGroup will allow the user to select the type of font to be used and a SpinEdit can be used to set the size of the font. Use a Label to show your 'sample' text.

The required font properties of the Label can be changed with

```
lblSample.Font.Size := 24;
lblSample.Font.Name := 'Times New Roman';
```



Tip: You can use the same Event handler for the OnClick event of the RadioGroup and for the OnChange event of the SpinEdit.

TIP

The Code Templates

The "outlines" of the various versions of the Case statement can be accessed by pressing <Ctrl><J> while in the code editor.

Checkpoint

1. Rewrite the following nested If statement in the form of a Case statement:

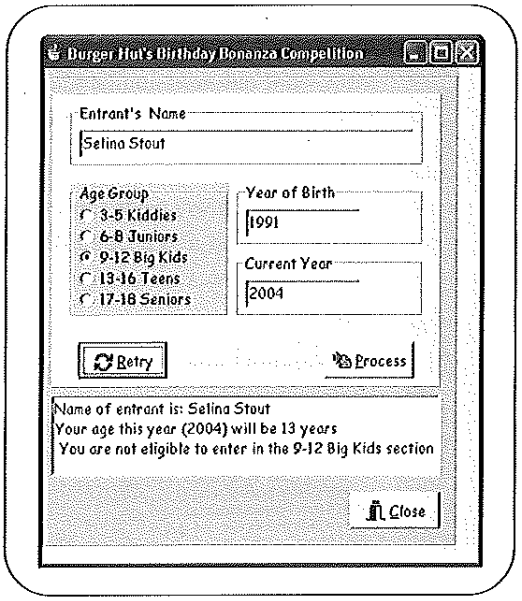
```
if iChoice IN [1..5]
  then lblMessage.Caption := 'Low'
  else if iChoice IN [6..10]
        then lblMessage.Caption := 'High'
        else if (iChoice = 0) OR (iChoice IN [11..99])
              then lblMessage.Caption := 'Out of range'
              else lblMessage.Caption := '';
```

2. The Burger Hut is running a drawing competition as part of their birthday celebrations. Entrants are invited to take part in any age grouping *as long as their age does not exceed the upper age limit for the category they choose*. Entrants must also be between the ages of 3 and 18. The various age groups are listed below:

Group	Age Range
Kiddies	3 – 5
Juniors	6 – 8
Big Kids	9 - 12
Teens	13 – 16
Seniors	17 – 18

Design a program that allows the user to enter the name of the entrant, their choice of category to enter, their year of birth and the current year. Create an interface similar to the one shown.

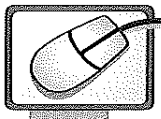
When the user clicks on the [Process] Button, the program must calculate and display the age of the entrant (at the end of the current year). This must be checked to see if they are eligible to enter the selected age group category. The program must indicate whether the entrant is allowed to participate according to his/her age.



The Timer

A Timer is useful when you need to have events being triggered at fixed time intervals or when you need to time a user's response for some reason. The Timer is a non-visual component, which means it can only be 'seen' at design-time and does not appear on the screen at run-time.

The Timer is a system event. That means that this event will take place without any action of the user. It will respond to something happening in the computer, i.e. the internal clock of the computer can trigger the Timer event.



Activity

Investigating the Timer

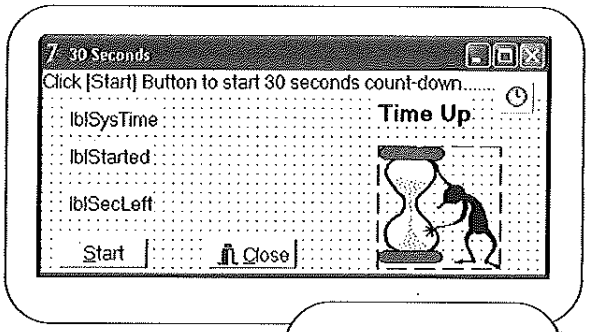
'30 Seconds' is a very popular board game. Teams play against each other and a member of a team gets thirty seconds to describe to his team mates the names on the card he drew. The hour glass provided to time the 30 seconds has, however, broken. We are now going to use the computer to keep time.

Design the following interface (as on the next page).

Place a Timer on the form and call it tmrSysTimeShow. Leave the interval of this timer as 1000 i.e. it will make something happen every second.

The Timer is going to make something happen every second. How do we know when 30 seconds have elapsed? Well, we let the computer count the seconds for us. To do this we

- create a global timer variable
- set the counter to 0 when we start the countdown
- increase the counter by 1 every time the Timer event happens
- put an If statement inside the Timer event to check if the counter has reached 30 yet!



The Timer can be found on the System tab.

Take a look at the code in the Event handlers below to see an example of how this can be done.

```
var
  frm30Seconds: Tfrm30Seconds;
  iCounter : integer;    (global counter variable)
implementation
{$R *.dfm}

procedure Tfrm30Seconds.btnStartClick(Sender: TObject);
begin
  tmrSysTime.Enabled := True;
  lblSysTime.Visible := True;
  imgTimeUp.Visible := False;
  lblTimeUp.Visible := False;
  lblStarted.Caption := 'Countdown started at: ' + TimeToStr(Time);
  lblStarted.Visible := True;
  lblSecLeft.Visible := True;
  iCounter := 0;
end;

procedure Tfrm30Seconds.tmrSysTimeTimer(Sender: TObject);
begin
  lblSysTime.Caption := 'Current Time: ' + TimeToStr(Time);
  lblSecLeft.Caption := 'Seconds left: ' + IntToStr(30 - iCounter);
  Inc(iCounter);
  if iCounter = 30
    then begin
           imgTimeUp.Visible := True;
           lblTimeUp.Visible := True;
           tmrSysTime.Enabled := False;
           lblSecLeft.Visible := False;
         end;
end;
```

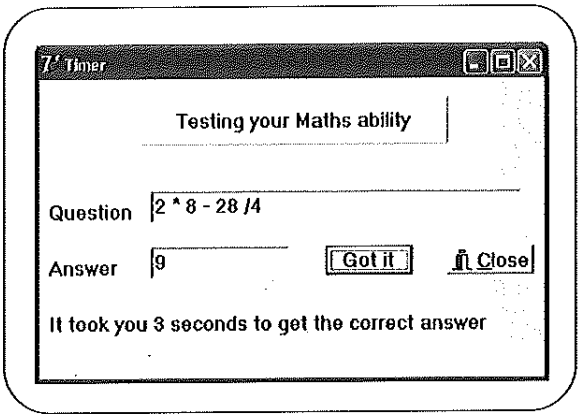
Now complete the program.

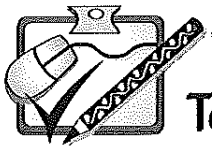
Checkpoint: Try the following yourself

1. Write a program that will flash the message "SMOKING IS DANGEROUS" and an appropriate picture every two seconds on the screen.
2. Write a progam that will ask the user to supply the answer to a mathematical question. When the form activates the question must be displayed in the Edit and the counter must be set to zero. A Timer must be added to count the time from the moment the form was activated until the user clicks the [Got it] Button to indicate that he/she has entered an answer.

   If the user answers correctly, the time it took him/her to get the correct answer must be displayed. If the answer is wrong an appropriate message must be displayed.

   The following is an example of the form after execution. (Remember that the Timer will not be visible.)





Test, Improve, Apply

Written exercises

1. Correct the following If statements if they are not correct.
   a) if iNum2 >< iNum1 then Inc(K);
   b) if a > b OR a < c then Inc(K);
   c) if a > b > c then Inc(K);
2. Write down the condition(s) that need to be specified as part of the If statement(s) to accomplish the following tests.

   Example:

   Question: Is G a positive number? Answer: if G > 0

   a) Is X a multiple of Y?
   b) Is A an even factor of B?
   c) Is M a perfect square?
   d) Is R a whole number and also not a negative number?
   e) Is K the smallest of three numbers K, M and P?
   f) Is J an even number?
   g) Is the average of three numbers A, B and C is less than -15?
   h) Does the value entered via SpinEdit sedGrade represent a valid grade for a learner at high school?

3. Explain for each program segment why the variable *iCount1* will ***never*** be incremented , irrespective of the values of the other variables used in each case:

a)
```
if (iNum1 Mod 2 = 0) OR ( Odd(iNum1) )
    then Inc(iCount2)
    else Inc(iCount1);
```

b)
```
if (iNum1 <> iNum2) AND (Min (iNum1,iNum2) - Max (iNum1,iNum2) = 0)
   then Inc(iCount1)
   else Inc(iCount2);
```

c)
```
if Int (Frac (iNum1 / iNum2 ) ) = 0
    then Inc(iCount2)
    else Inc(iCount1);
```

d)
```
iNum1 := -iNum2;
if (iNum1 > iNum2) AND (iNum1 * iNum2 > 0)
   then Inc(iCount1)
   else Inc(iCount2);
```

4. Consider the following Delphi code segment, which displays a grade symbol or a message based on a student's mark. For a mark from 80 to 100, an A symbol will be displayed, for 70 to 79, a B symbol, for 60 to 69 a C symbol. For any mark below 60 the message 'Grade is below C' will be displayed.

   The code is not correct because when marks such as 75 and 85 are entered, the message 'Grade is below C' is displayed instead of the correct symbol.

```
if iMark > 80 then edtGrade.Text := 'A' ;
if iMark > 70 then edtGrade.Text := 'B' ;
if (iMark >= 60) AND (iMark < 70)
    then edtGrade.Text := 'C'
    else edtGrade.Text := 'Grade is below C' ;
```

   a) Briefly explain why the code produces the wrong output for marks such as 75 and 85.

   b) Rewrite the code so that the program works correctly for all possible values for iMark from 1 to 100 (inclusive).

5. The code segment below must indicate that someone is eligible to vote if they are 18 years or older and a citizen of 'RSA'.

```
if iAge >= 18
    then if sCountry = 'RSA'
           then lblMessage.Caption := 'Eligible to vote'
           else lblMessage.Caption := 'Too young to vote';
```

   a) Show what the output will be in each of the following cases, using the values of the variables given:

   (i)
```
iAge := 16 ; sCountry := 'ZIM';
```

   (ii)
```
iAge := 16 ; sCountry := 'RSA';
```

   (iii)
```
iAge := 36 ; sCountry := 'ZIM';
```

   b) Rewrite the code so that it works for all possible combinations of values.

6. Rewrite the following nested If statement as a Case statement:

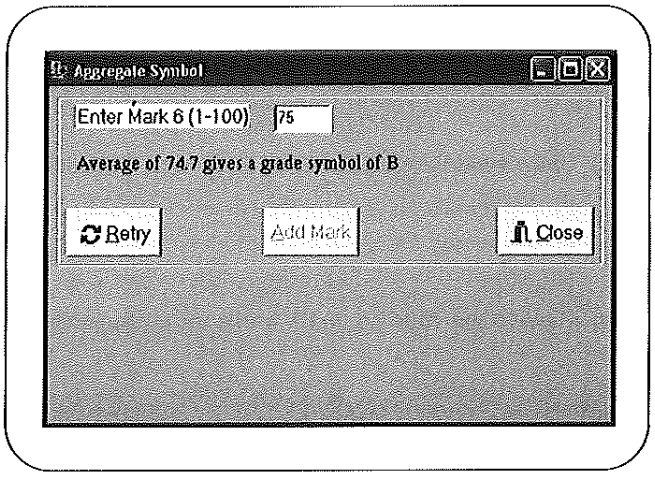
```
if (iChoice > 5) AND (iChoice <= 10)
   then begin
          iNum1 := iNum2;
          iNum2 := iNum3;
        end
   else if iChoice IN [1..5]
          then begin
                 iNum1 := iNum2 + iNum3;
                 iNum3 := Sqr(iNum3);
               end
          else if iChoice < 15
                 then iNum1 := iNum3
                 else blOutput.Caption := 'Choice not valid';
```

7. Simply the following code by making use of sets:

```
if (iChoice >= 5) AND (iChoice < 11)
   then
     begin
       if((iM <= 3) AND (iM >= 1))OR((iM <= 12)AND(iM >= 10))
         then Inc(iCounter1);
       if ((iDay <= 15) AND (iDay >= 1))then Inc(iCounter2);
     end
   else if (iChoice >= 1) AND (iChoice <= 4)
          then
            begin
              if (cX = 'A') OR (cX = 'B') OR (cX = 'C') OR (cX = 'D')
                then Dec(iCounter1);
              if (cX <> 'X') AND (cX <> 'Y') AND (cX <> 'Z')
                then Dec(iCounter2);
            end;
```

Practical exercises

8. We want to design a program that allows a user to enter six marks (out of 100). The program must calculate the average and allocate an overall grade based on the average mark.



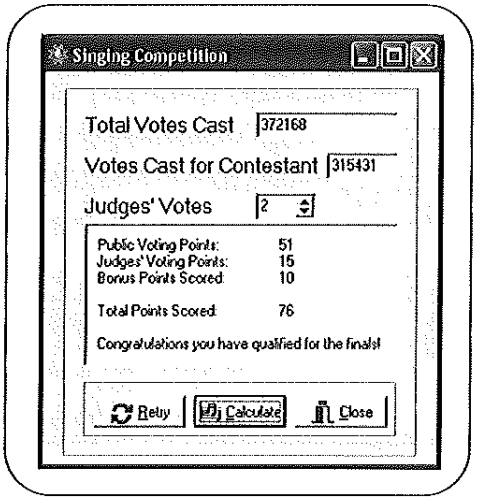
The overall grade is classified according to the average of the six marks as follows:

A	B	C	D	E	F	G	H
80-100	70-79	60-69	50-59	40-49	30-39	20-29	Below 20

The average and the corresponding aggregate symbol must be displayed as soon as the user has entered the sixth mark.

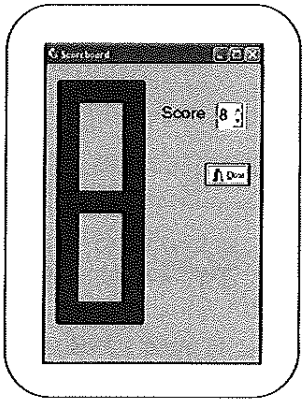
9. The organisers of a singing competition use a grading system to see if a contestant makes it through to the final:
   - They can score a maximum of 100 points.
   - The percentage of votes they get of the total public vote is rounded to a number out of 60.
   - The number of judges' votes that they obtain (out of the 4) is rounded to a number out of 30.
   - They score 10 additional bonus points if they score more than 75% of the public ***or*** of the judges' votes.
   - To qualify they must have a point's score of at least 75 ***and*** have at least two judges' votes.

   Display the points and whether the contestant has qualified or not.



10. A single digit scoreboard is made up of a circuit of seven LEDs *(Light Emitting Diodes).* Each of the digits 0 to 9 can be displayed by highlighting or switching on the correct LEDs. (For example, the digit 8, would be displayed by switching on *all* the LEDs.)

    Create an interface where seven Shapes on a form represent the LEDs. Add a SpinEdit which when altered, must light up the appropriate LEDs by making them visible or invisible as the case might be.

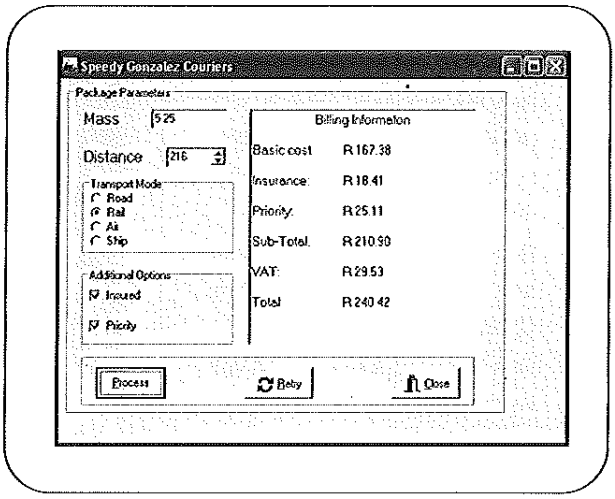


11. Use Shapes (in the form of circles) with a Timer to simulate a traffic light on your form, changing from green to amber to red in turn. Allow the user to specify how many times that they want the cycle to be repeated.

12. You need to design a program for a courier company to calculate the cost of sending a parcel. The cost is determined by the weight of the parcel, the mode of transport used and the distance travelled.

    The basic cost is calculated as R 1.23 per kilogram times the transport cost per kilometre, which is fixed as follows:

Road	R 0.15
Rail	R 0.12
Air	R 0.36
Ship	R 0.25



If the customer wishes to insure the parcel, a charge of 11% of the basic cost is added. If the customer wishes to send the parcel as priority, a charge of 15% of the basic cost is added. VAT must be added to determine the total cost .The billing details need to be calculated and an appropriate breakdown of the cost displayed when the user clicks on the [Process] Button.

13. ***Teach yourself the MaskEdit***

- Start a new application and add a [Display] Button, a Close BitBtn and a MaskEdit (found on Additional palette). Rename the MaskEdit *medInput.*
- Add the following code in the Event handler of the [Display] Button:

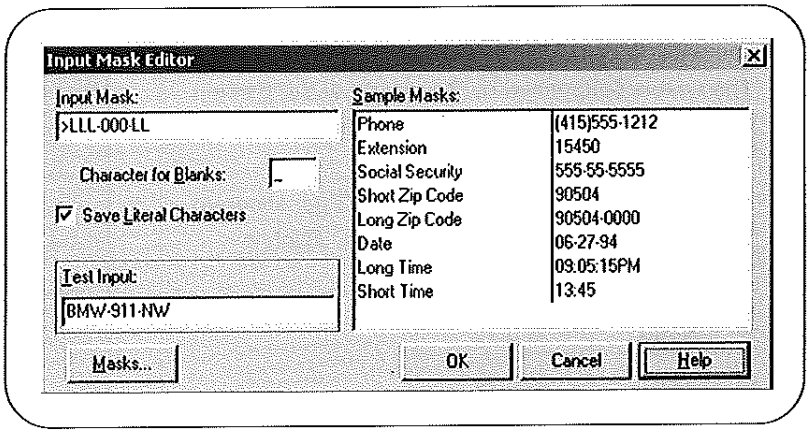
```
ShowMessage (medInput.Text);
```

- Now, expand the EditMask property of the MaskEdit by clicking on the ellipsis (...) Button.
- The Input Mask Editor will be displayed.

The following is a summary of the more commonly used characters used in the mask:

Character	Significance in Mask
>	Characters that follow are in uppercase.
<	Characters that follow are in lowercase.
\	Character that follows is literal (as it appears in the mask).
L	Character is compulsory and must be alphabetic.
l	Character is optional but must be alphabetic if used.
A	Character is compulsory and must be alphanumeric ('A'..'Z', 'a'..'z', '0'..'9').
a	Character is optional, but must be alphanumeric if used.
0	Character is compulsory and must be numeric (0..9).
9	Character is optional but must be numeric if used.

- Let's experiment with some masks. Let's say we want the user to enter the registration number of a motorcar in the format BCD345NP. The input mask to achieve this is: >LLL-000-LL.
  In other words all letters will be uppercase with three compulsory letters, followed by three compulsory digits and then two more compulsory letters.
- Type in the mask and test the mask in the Test Input Edit provided.
- Run the program, enter a registration number, and take notice of what is displayed by the ShowMessage.
- Return to the Input Mask Editor and 'uncheck' the 'Save Literal Characters' option.
- Run the program again. What is the difference?



- Try the following masks and make sure you understand why they are specified like they are:

  For a Identity Number: 000000-0000-000

  For a date in the format: 2004-12-30 \2000-90-90

  For a cell phone number: \000-000-0000

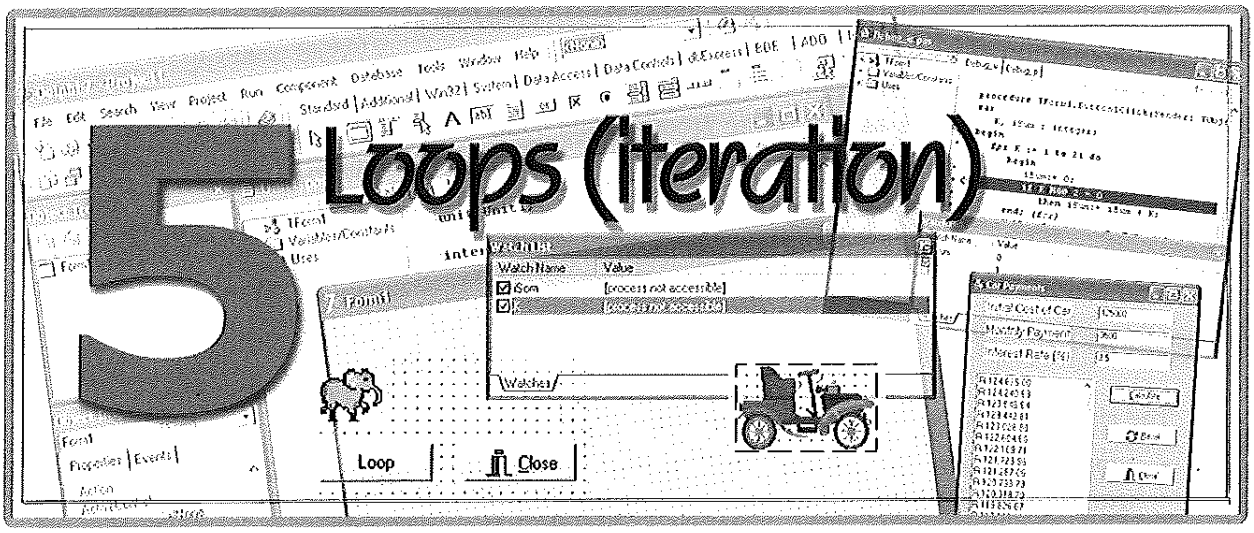
The MaskEdit not only forces the user to enter data in the correct format, but to some extent provides type and range validation of the data being input.

Remember also that the MaskEdit, like the Edit, has a MaxLength property to limit the number of characters that can be entered into the control.

Test yourself

14. Check that you know / can do the following:

		✓
Knowledge	**I can explain in my own words**	
	what an IF statement is used for	
	what a relational operator and a Boolean condition is	
	the difference between the AND, OR and NOT operators	
	what a compound statement is	
	what a nested If statements is	
	when we would make use of a Case statement	
	when to use CheckBoxes, RadioButtons and RadioGroups	
Skills	**I can**	
	compile and specify Boolean conditions using relational operators and the AND, OR and NOT operators	
	formulate and use If statements in a problem-solving context	
	use a RadioGroup and CheckBox to allow users to indicate choices visually in programs	
	implement Case statements	
	use combinations of mathematical functions to test for various conditions including finding maximum/minimum values and to test for factors, perfect squares and whole numbers	
	specify sets and utilise sub-ranges	
	use the Timer to trigger Event handlers at specific time intervals	
	use the MaskEdit to help the user to enter data in the correct format	



When you have completed this chapter you should be able to

- explain under circumstances you must use loops to execute repetitive tasks
- make use of appropriate loop structures in your programs
- set up trace tables to detect errors in programs
- detect errors in programs through the use of Delphi's debugger
- use the Canvas-property of a form to draw diagrams and figures

Why Loops?

Computers are good at doing boring jobs that are repetitive. In fact the more simple and repetitive a job is the better the computer is at doing it!

As you develop your programming skills you will find that almost every programming problem you will ever encounter can be solved by using some form of loop (repetition). This is why one of the most important skills you can learn as a programmer is to be able to identify repetitive patterns when solving problems.

Creating a program to do the job is then simply a matter of writing code to do the job once and then applying an appropriate loop to repeat that job as many times as necessary!

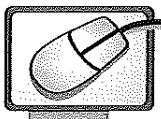
In some applications the user will control the repetition by clicking on a particular Button every time an action must take place. In other applications, however, we would prefer to let the program do the repetition without allowing the user to take control. For these applications we need control structures as part of the programming code. In Delphi, as in most of the other programming languages, there are two basic control structures, namely:

- Unconditional loops (For loop)
- Conditional loops (While loop, Repeat loop)

The For Loop

The simplest type of loop is a ***For*** loop. This loop

- repeats an instruction (or a few instructions) a **set number** of times (this makes it an **unconditional** loop)
- is predictable.



Activity

The working of the For loop

- Select a new application.
- Place a RichEdit called redNumbers on the form.
- Place a Button with the caption Show Numbers on the form and a Close BitBtn.
- Double-click on the [Show Numbers] Button.
- Type in the following code and execute the program.

```
var
   K : integer;
begin
   for K := 1 to 10 do
      redNumbers.Lines.Add(IntToStr(K));
end;
```

Note the following:

The For loop works on the principle of making the computer repeat a statement or a few statements a set number of times.

A loop variable (K) is used to do the counting from 1 to 10 in this example.

We would have to repeat this statement 10 times if we did not make use of a for-loop:

```
redNumbers.Lines.Add(IntToStr(1));
redNumbers.Lines.Add(IntToStr(2));
redNumbers.Lines.Add(IntToStr(3));
.....
```

Change the For loops as indicated and run the program each time:

- `for K := 10 to 15 do`
- `for K := -5 to 3 do`
- `for K := 15 downto 10 do`
-
```
var
  K : char;
for K := 'A' to 'G' do
```

Therefore we see that:

- the loop variable can begin with any ordinal value
- the loop variable's value can be *decreased* by one repeatedly by using *downto*
- characters can also be used to count.

General syntax

Note: There is no ';' after the *Do*.

for <LoopVariable> := <StartValue> to <EndValue> do

begin

.....

.....

.....

end;

When there are more than one statement that should be repeated, these statements should be placed between a Begin and an End.

LoopVariable: Acts as the counter for the For loop. The LoopVariable must be an ordinal data type such as Integer or Char. (It may not, for example, be a real value or a character string.)

StartValue: It is the first value assigend to the LoopVariable.

EndValue: The LoopVariable will be increased by one automatically each time the loop is executed until the value of the LoopVariable is greater than the EndValue, by which time the loop will stop.

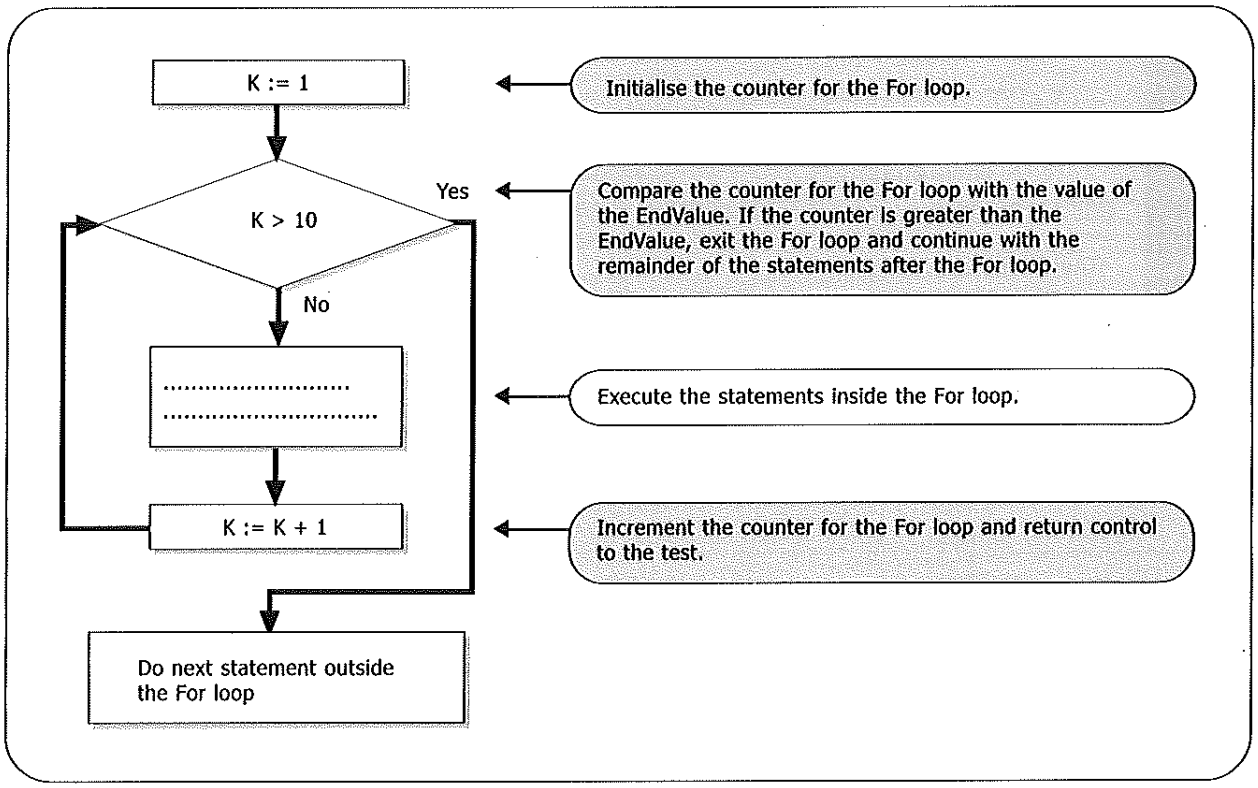
Flowchart of the For loop:

The loop structure

```
for K := 1 to 10 do
begin
   ..................
   ..................
end;
```

can be depicted as a flow chart as follows:

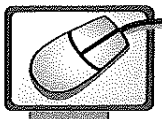
The shaded parts are executed ***automatically***.



Use the For loop to solve problems

For loops are good for working with anything that involves sequences of numbers. This can cover a larger area of topics than you might think possible. The key to knowing whether to use a For loop is to ask yourself if the task you want the computer to do involves a fixed number of repetitions.

You have to learn to spot repetitive patterns in a problem. That is when you will need a For loop.



Activity

Using the For loop

1. Write a Delphi program to do the following: Display the first 10 square numbers.
   First think of how you would solve the problem yourself before you create the IPO table:
   A square number is a value that is the product of two identical numbers:
   We have to print the following numbers 1, 4, 9, ...
   How do we find the values? 1 * 1 = 1
   2 * 2 = 4 etc.
   As soon as you find the *repetitive pattern* as identified above, the For loop can be used.

Step 1: Create the IPO table

Input	Processing	Output
None	Loop from 1 to 10 iSquare ← iCount ^{2}	iSquare

Step 2: Choose and declare variables

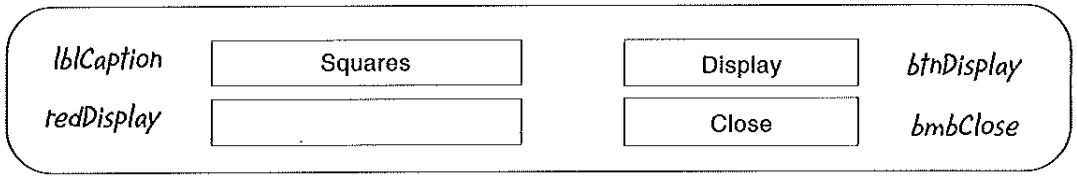
You always need a LoopVariable if you use a For loop.

We also need a variable to store the answers.

```
var
   iCount, iSquare : integer;
```

Step 3: Design the user interface

Decide on the necessary components:



Step 4: Code of the Event handler for btnDisplay

```
for iCount := 1 to 10 do
   begin
      iSquare := sqr(iCount);
      redDisplay.Lines.Add(IntToStr(iSquare));
   end;
```

2. Write a Delphi program to do the following: Calculate the sum of a number of integers, i.e. 1 + 2 + 3 + 4 + 5 ... + N, where N is determined by the user.

Step 1: Create the IPO table

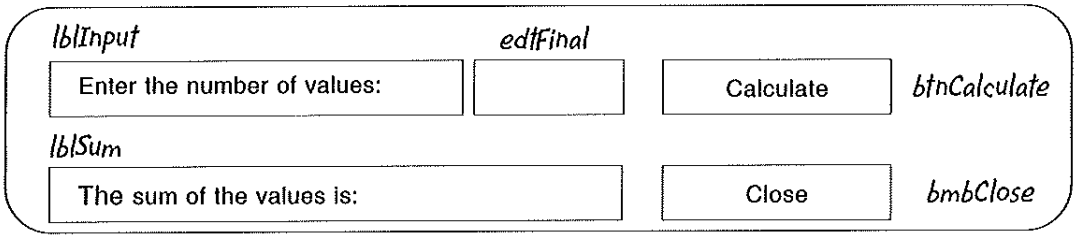
Input	Processing	Output
Final	Loop from 1 to Final Sum = Sum + Number	Sum with a message

Step 2: Choose and declare variables

```
var
   iFinal, iSum, iCount : integer;
```

Step 3: Design the user interface

Decide on the necessary components:



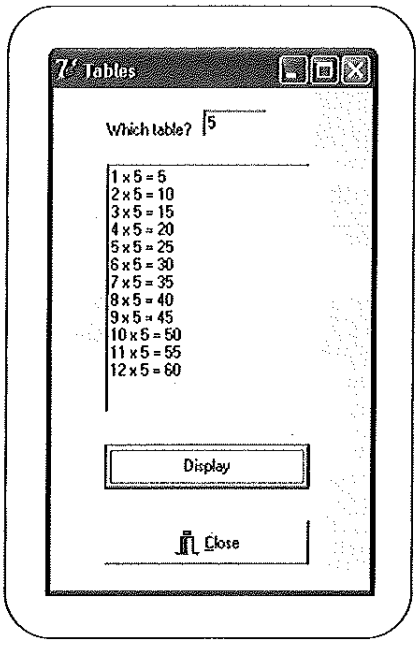
Step 4: Code for the Event handler of btnCalculate:

```
iFinal := StrToInt(edtFinal.Text);
iSum := 0;
for iCount := 1 to iFinal do
   iSum := iSum + iCount;
lblSum.Caption := 'The sum of the values is ' + IntToStr(iSum);
```

NB
iSum has to be reset to zero before the For loop starts.

Checkpoint: Try the following yourself

1. Write a Delphi program that asks the user which multiplication table has to be displayed. The program must use a For loop to display the tables from 1 to 12.



2. Write a Delphi program that will calculate and display the sum of all the even numbers between two values. The program must also display the number of even numbers found. The user has to enter the two values of the boundaries. If the lower boundary is 10 and the upper boundary 20, then 12 + 14 + 16 + 18 has to be calculated.
3. The value of your computer decreases by 5% per year. Write a program that will ask the user what the original value of your computer was. Use a For loop to calculate the value of your computer in 5 years time. Display each year's decreased value in a RichEdit.

   Display the difference between the price you purchased the computer for and the decreased value after 5 years as well.
4. Let the user enter the start value and the incremental value of a series of numbers. Use a For loop to calculate and display the first 5 terms of the series. The program must calculate and display the sum of the first 5 values as well.

   Example:

   Input: Start value : 5
   Incremental value : 3

   Output: 5 + 8 + 11 + 14 + 17 = 55

Trace tables

When a program is executed but incorrect results are displayed, it is usually because a logical error was made in the program. Sometimes it is difficult to spot a logical error. A trace tabel is used to trace the way a program executes. This table is used to represent the *contents of variables* when the program is executed, the results of all *tests* as well as the *output*.

The creation of a trace table is a valuable technique

- to detect logical errors
- to determine the purpose of a segment of programming code.

Example

Determine the output of the following program code:

```
var
  iCount, iNumber   : integer;
begin
    for iCount := 1 to 5 do                          {1}
      begin
        iNumber := iCount * iCount;                  {2}
        redNumber.Lines.Add(IntToStr(iNumber));      {3}
      end; {for}
end;
```

The statements have been numbered to assist in the creation of the trace table.

Trace table:

Heading of trace table:

- Each **variable** in the code is a heading in the trace table.
- Since iCount is the LoopVariable, it is compared to the end value of the For loop. This **test** is a heading in the table.
- The **output** component is a heading in the table.

Line	iCount	iCount > 5?	iNumber	Output redNumber
1	1	No		
2			1	
3				1
1	2	No		
2			4	
3				4
1	3	No		
2			9	
3				9
1	4	No		
2			16	
3				16
1	5	No		
2			25	
3				25
1	6	Yes		

Content:

- Each line in the table represents the execution of a statement in the program.

  {1}: iCount receives the value 1 and is compared to the end value

  {2}: iNumber receives a value (iCount * iCount))

  {3}: iNumber is transferred to the output component

  {1}: Jumps back to the For loop, add 1 to iCount and compare

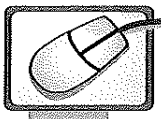
  and so forth..

Conclusion:

From the values in the output column we can see that the squares of the first 5 whole numbers will be displayed.

The Debugger

Delphi includes a debugger that can help you to find errors in your program. Let's see how Delphi's debugger works.



Activity

Working with the debugger

- Start a new application.
- Place Buttons named Calculate Total and a Close on the form.
- Place a Label on the form to display the result of the calculation. Name this Label lblSum.
- Double-click on the Button [Calculate Total].
- Enter the following code and execute the program:

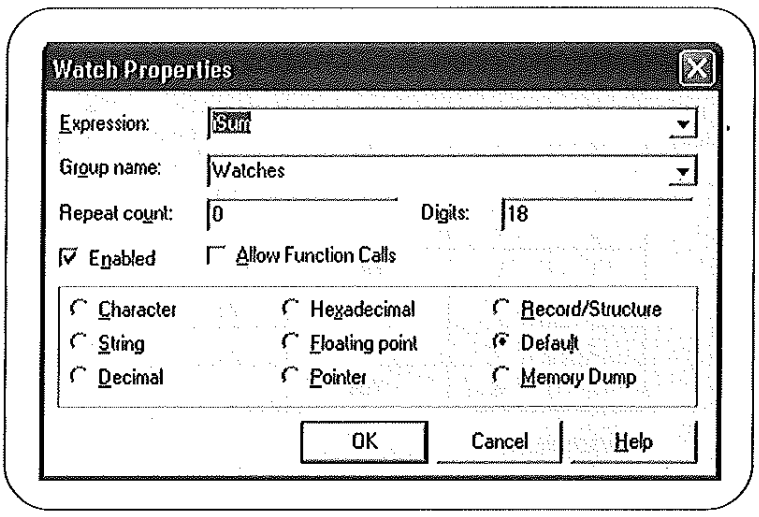
```
var
  K, iSum : integer;
  begin
     for K := 1 to 10 do                    (1)
     begin
        iSum := 0;                          (2)
        if K MOD 2 = 0                      (3)
           then iSum := iSum + K;           (4)
     end;
     lblSum.Caption := IntToStr(iSum);      (5)
end;
```

The purpose of the program is to calculate the sum of all the even numbers from 1 to 10. When you execute the program an answer of **10** will be displayed in the Label, but the value of iSum should be **30**.

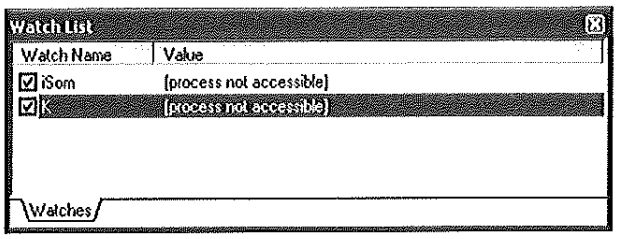
You want to know what happens to the variable iSum inside the loop so that you can identify and correct the error.

Use the debugger as follows:

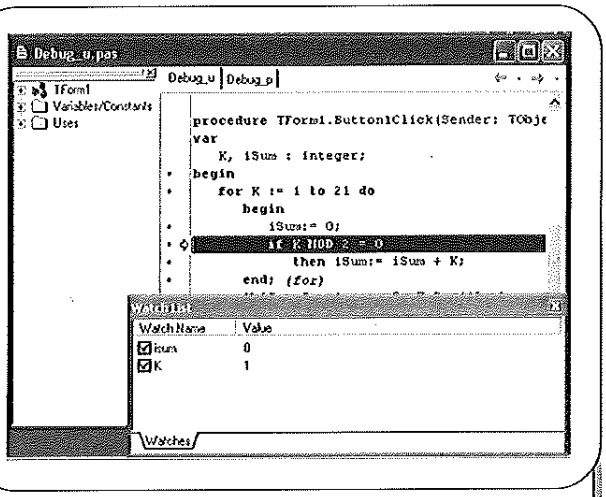
- Use the mouse to click on the variable you want to investigate – iSum in line {4}.
- Click on ***Run, Add Watch.***



- A Watch dialogue will appear indicating that iSum will be watched. (You can also enter the name of the variable you want to trace in the Expression window instead of clicking on the variable before the time.)
- A Watch List appears once you click on OK.
- Suppose you want to add the variable K as well. Right-click on the Watch List and enter the name of the variable you want to add (K) in the Expression window.
- Right-click on the Watch List and choose the *Stay on Top* option to display the values of the variables in the Watch List.



- Repeatedly press <F7> (or click on ***Run, Trace Into***) to execute the program statements one by one.
- Click on the [Calculate Total] Button and press <F7> repeatedly again so that the statements are executed one by one.
- Can you see that the value of iSum is reset to 0 each time?
- Switch off the debugging session by clicking on ***Run, Reset Program***.
- Move Statement {2} to before the For loop so that iSum only receives the value 0 before the loop starts.
- Step through the program again using <F7> and look at the value of iSum to ensure that the program is correct.
- Deactivate the Watch List by right-clicking and then clicking on the *Stay on Top* option again.



Tip

When your programs start to get longer, stepping through them until you reach the line with the error can be a long and irritating process. Luckily, Delphi is designed to make things easier for us. If you click in the left hand margin (the grey edge) of the code window next to the line which contains the code you want to check you will create a ***breakpoint***. This simply means that you can run your program as normal. When it gets to that line of code it will stop and return you to the code windows where you can see all your watches and use <F7> to step through the program. When you have checked the errrors you want to check you can press <F9> and the program will continue running.

You will know if you have set a breakpoint correctly because that line of code will be highlighted in red. If you click in the marging again when the breakpoint is already set then you will clear the breakpoint (i.e. turn it off).

Note:

You can also move the mouse over the variable iSum in the program at any time during debugging. The contents of iSum will appear in a hint box on the screen.

The Watch window can be docked underneath the unit and closed when all errors have been corrected. Watches are no longer necessary. By right-clicking on the window watches can be added or removed.

Change the program and use the debugger again to try and find the error.

Change the activity above as follows:

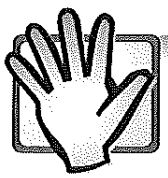
The program above has to display all the even numbers before the sum is displayed. Create a RichEdit on the form with the name redEven. Add the statements {1} and {2} as indicated:

If the debugger does not want to work, make sure the necessary compiler options are set:

- Click on ***Project, Options, Compiler.***
- Make sure the following debugging options are active:
  Debug Information, Local Symbols, Assertions.

```
var
   K, iSum : integer;
   begin
      iSum := 0;
      for K := 1 to 10 do
      begin
        if K MOD 2 = 0
           then iSum := iSum + K;
                iEven := K;                              {1}
                redEven.Lines.Add(IntToStr(iEven));      {2}
      end; {for}
      lblSum.Caption := IntToStr(iSum);
   end;
```

Let the program execute. The output in the RichEdit is not correct. Use the debugger and try to find the error.



Checkpoint: Try the following yourself

Load each program and see if you can find the error. You may use trace tables or the debugger in Delphi to find the errors. Correct the errors so that the program produces the correct output.

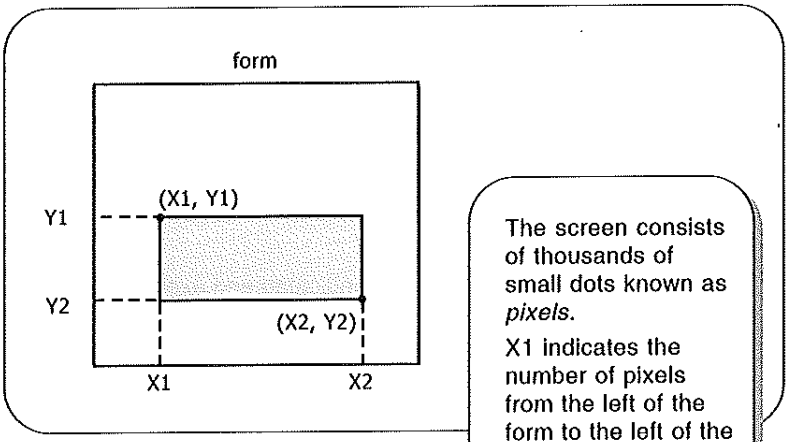
1. Load the program ForOne on the CD. This program is meant to display 10 * symbols in a Label. See if you can find and correct the programming error.
2. Load the program ForTwo on the CD. This program is meant to display 10 * and 10 # symbols in a Label. See if you can find and correct the programming error.
3. Load the program ForThree on the CD. This program is meant to display the 5 times table. See if you can find and correct the programming error.

Fun with the Canvas

The form's Canvas property represents the drawing surface of the form and allows us to draw bitmaps, lines, shapes or even text on the form at runtime. Instead of being restricted to the different shapes of the Shape component, we can use the Canvas to draw our own pictures. Many of these options require us to use loops, so let's have some fun while using For loops!

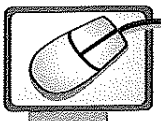
The *Canvas* can be used to draw various objects such as circles and rectangles. To draw a rectangle:

Rectangle (X1, Y1, X2, Y2). (X1,Y1) are the top left coordinates of the rectangle and (X2,Y2) are the coordinates of the bottom-right point.



The screen consists of thousands of small dots known as *pixels*.

X1 indicates the number of pixels from the left of the form to the left of the rectangle.



Activity

Exploring the Canvas

Playing with Rectangles

To draw a single rectangle, do the following:

- Create a new application.
- Place a Button with the caption Forms on the form, as well as a Close BitBtn.
- Double click on the [Forms] Button and enter the following code:

```
begin
   Canvas.Brush.Color := clYellow;
   Canvas.Rectangle(100, 100, 200, 200);
end;
```

Set the *Brush* property to specify the colour of the rectangle.

- Run the program.
- Change the first two values of the rectangle statement and run the program.
- Change the last two values of the rectangle statement and run the program.
- Replace the word Rectangle with the word Ellipse and run the program.

Creating the illusion of movement

- We will now use a For loop to create the effect of movement on the screen.
- Change the code as follows:

```
var
   K : integer;
begin
   Canvas.Brush.Color := clYellow;
   for K := 1 to 200 do
      begin
         Canvas.Rectangle(50, 50, 100 + K, 100 + K);
         Sleep(50);
      end;
end;
```

Note the following:
The rectangle is drawn 200 times each time with a new X2 and Y2 position.
The first rectangle's coordinates will be (50, 50, 101, 101).
The next rectangle's coordinates will be (50, 50, 102, 102).
The next rectangle's coordinates will be (50, 50, 103, 103).
And so forth.
The Sleep statement will pause the program for 50 milliseconds before the next rectangle is drawn.

Run the program.

Change the For loops as indicated and execute the program:

- `for K := 100 to 200 do`
- `for K := 200 downto 100 do`

Add the following code below the existing code:

```
Canvas.Brush.Color := clBlue;
for K := 1 to 200 do
   begin
      Canvas.Rectangle(200,200,100 + K, 100 + K);
      Sleep(20);
   end;
```

Try the following:

```
var
   K : integer;
begin
   Canvas.Brush.Color := clGreen;
   for K := 1 to 5 do
      begin
         Canvas.Rectangle(100,100,200,200);
         Sleep(300);
         frmRect.Refresh;
         Sleep(300);
      end;
end;
```

Use the name you gave your form.

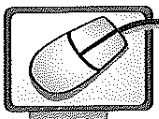
See what types of interesting patterns you can create by using For loops to repeatedly draw circles and rectangles.

The While Loop

The While loop is used when the programmer does not know how many times the loop will be executed. This loop

- repeats an instruction (or a few instructions) an unknown number of times
- is repeated while a certain condition is met (this makes it a **conditional** loop)

Let's have a look at an example of a ***While*** loop to see how it works.



Activity

Investigating While loops

- Select a new application.
- Create a RichEdit called redAdd.
- Place a Button with the caption Show Total on the form and a Close BitBtn.
- Double-click on the [Show Total] Button
- Type in the following code and execute the program.

```
var
   iNumber : integer;
begin
   iNumber := 13;
   while iNumber < 100 do
   begin
      redAdd.Lines.Add(IntToStr(iNumber));
      iNumber := iNumber + 13;
   end; (while)
end;
```

This program displays all the multiples of 13 less than 100.

General Structure

<Initialise variable(s) needed to test condition(s) before the beginning of the loop>
While <conditions(s)> do

Begin

.....

.....

<change(s) made to variables affecting loop's test condition(s)>

End;

The 'ITC-principle' holds true for the While loop:

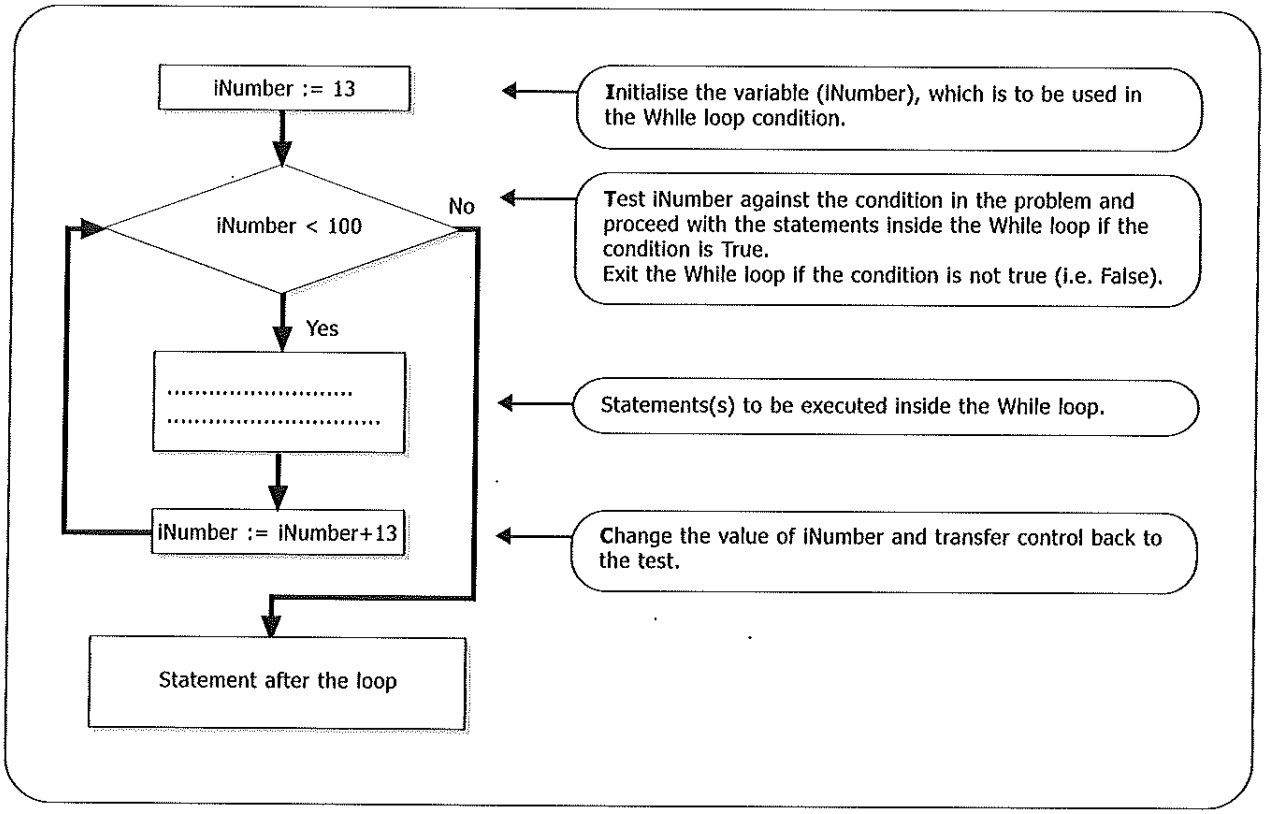
Initialise:	The variable(s) tested at the beginning of the loop must be assigned value(s) **before** the test is done by the While loop.
Test	The variable(s) is tested at the beginning of the loop. If the condition(s) is **True**, then the statement(s) within the loop is executed.
Change:	The variable(s) used in the test at the beginning of the loop needs to be changed otherwise the condition(s) will always be True, causing the loop to run infinitely (an infinite loop).

Flowchart of the While loop:

The loop structure

```
iNumber := 13;
while iNumber < 100 do
begin
  .....
  iNumber := iNumber + 13;
end;
```

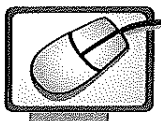
can be represented in flowchart format as follows:



Using the While loop in problem solving

To decide whether we need to use a While loop to solve a problem, we need to analyse the problem and determine the following:

- Is there *repetition* of some code (activities)?
- Is the number of repetitions *unknown*?
- When must the repetition stop and what is the condition required for the loop to continue executing?



Activity

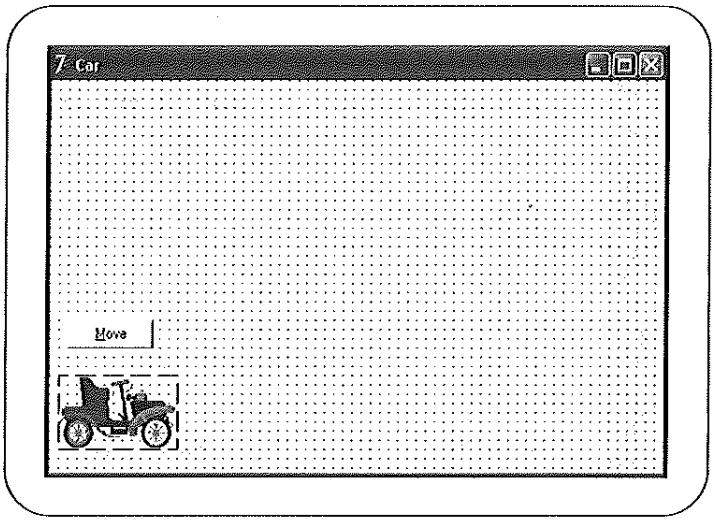
Using the While loop to control movement of an object

We are going to use the While loop to let a car move over the screen.

Create the following interface: (Use any picture.)

When the user clicks the [Move] Button the car must move over the screen. We do not, however, want it to drive 'off the screen'.

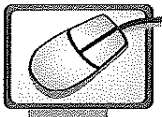
When the right hand side of the image reaches the right hand side of the form the motion must stop. We can easily test this by comparing the Left property of the car (image) with the ClientWidth property of the form.



Add the following code to the OnClick Event handler of the [Move] Button:

```
begin
  while imgCar.Left < (frmCar.ClientWidth - (imgCar.Width + 5)) do
     begin
        imgCar.Left:= imgCar.Left + 10;
        imgCar.Repaint;
        Sleep(50);
     end; {while}
end;
```

Save and run your program.



Activity

Using the While loop in numeric calculations

A computer program needs to be designed to calculate the number of years in which the value of a property will double. The program must display the increased value of the property on an annual basis as well as the number of years it will take to double in value.

The user must enter the present value of the property (E.g. R300000) and the expected percentage increase rate per year (E.g. 5%).

- Because the value of the property is recalculated annually, there obviously must be some form of *repetition.*
- As the number of years it will take to double in value is unknown, the *number of repetitions is unknown* – we can therefore use a While loop.
- While the value of the property is less than double the original value, the loop must continue – this is the condition.

Step 1: Construct the IPO table

Input	*Processing*	*Output*
Current property value, Percentage increase rate	*Determine the number of years it will take for the value of the property to double*	*Increased value (yearly) Years*

Algorithm

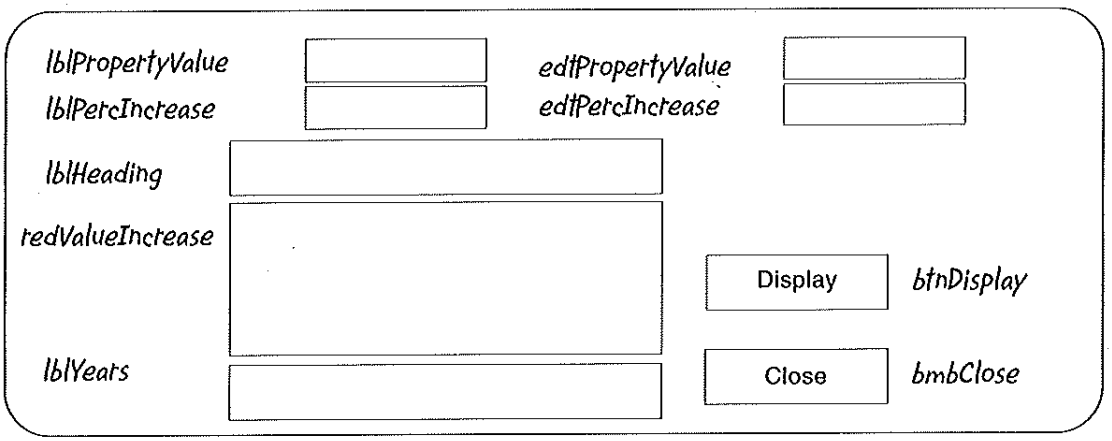
```
Read Current property value
Read Percentage increase rate
Final value = 2 x Current property value
Years = 0
While Current property value < Final value
    Increased value = Current property value * Percentage increase rate/100
    Current property value = Current property value + Increased value
    Years = Years + 1
   Write Increased value
endWhile
Write Years
```

Step 2: Choose and declare variables

```
var
    rPropertyValue, rPercIncrease  : real;
    iYears                         : integer;
    rIncreasedValue, rFinalValue   : real;
```

Step 3: Design the user interface

Decide on necessary components:



Step 4 : Code the Event handler of btnDisplay

```
begin
  rPropertyValue := StrToFloat(edtPropertyValue.Text);
  rPercIncrease := StrToFloat(edtPercIncrease.Text);
  rFinalValue := 2 * rPropertyValue;
  iYears := 0;
  while rPropertyValue < rFinalValue do
     begin
        rIncreasedValue := rPropertyValue * (rPercIncrease/100);
        rPropertyValue := rPropertyValue + rIncreasedValue;
        iYears := iYears + 1;
        redValueIncrease.Lines.Add('R ' +
                    FloatToStrF(rPropertyValue,ffFixed, 10, 2));
     end; (while)
     lblYears.Caption := 'Number of years: ' + IntToStr(iYears);
end;
```

Activity

Letting the user terminate the While loop

1. Write a program that counts, displays the numbers as it counts – and continues counting while the user does not click the [Stop] Button.
   - A new number needs to be displayed each time the computer increases the count *(repetition).*
   - The number of repetitions is *unknown* (the user can click the [Stop] at any time). Therefore we use a While loop.
   - While the user does not click the [Stop] Button the loop must execute - that is the *condition.*

Step 1: Construct the IPO table

Input	Processing	Output

Click the start button	Counter = 0;
	Stop := false;
	While Stop = false do
	Inc(Counter)
	Display Counter
	Respond to user action
	End While
		Counter

Step 2: Choose and declare variables

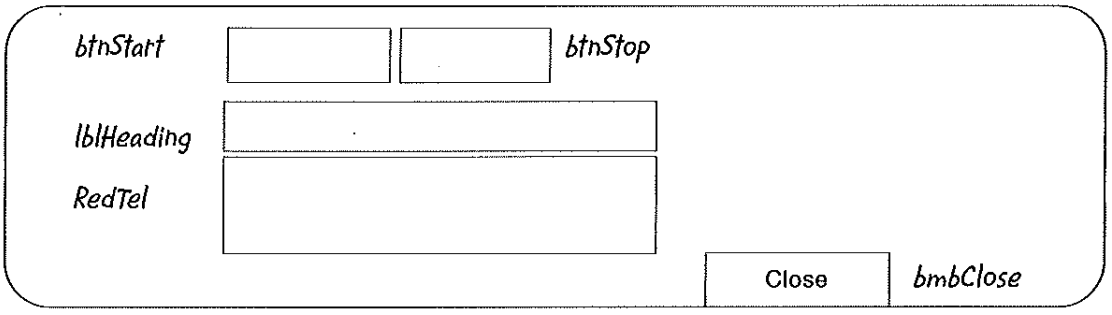
```
var
  Form1 : TfrmNumbers;
  bStop : boolean;
```

Declare the boolean variable bStop globally because it needs to be used in two procedures.

```
procedure TForm1.btnStartClick(Sender: TObject);
var
  iCounter  :byte;
```

Step 3: Design the user interface

Decide on necessary components:



Step 4: Code the Event handlers

Code of the btnStart *Event handler*

```
begin
  bStop := False;
  iCounter := 0;
  while bStop = False do
     begin
        inc(iCounter);
        redTel.Lines.Add(IntToStr(iCounter));
        Sleep(300);
        Application.ProcessMessages;
     end; (while)
end;
```

Refer to the explanation after the activity.

Code of the btnStop *Event handler*

```
procedure TfrmNumbers.btnStopClick(Sender: TObject);
begin
   bStop := True;
end;
```

Tip: Change the *Scrollbar* property of the RichEdit using the OI to add a vertical scrollbar. This will enable you to see all the numbers.

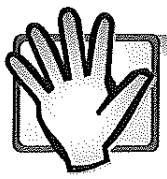
Application.ProcessMessages is a command that makes the program deal with messages and instructions that are waiting to be processed.

Remember, Windows programs react to events – moving the mouse, clicking a button, pressing a key – these are all events.

When you write the code to respond to an event and it is executed, the program executes that code only – and when it is finished it goes back to waiting for another event to happen. Events that happen whilst your code is executing do not get lost – they get put into a queue and will be processed in the order that they came up (just like when you queue up to buy movie tickets).

The problem is that the program waits for the code in the current event to finish *before* it can respond to any other event. Practically speaking, when you click a Button while a loop is executing, you are generating another event. This event will not happen until the code for the current Event handler is complete – unless you include the ***Application.ProcessMessages*** instruction in your code.

In the program above it is absolutely vital that the program responds to other events – otherwise the btnStop OnClick event will not happen and the loop will never end.



Checkpoint: Try the following yourself

1. Write a program to determine how may terms in the following sequence need to be added to obtain a total of just more than 200: 1, 4, 7, 10, ...
2. Write a computer game, that works as follows:

   A [Start] and a [Stop] Button must be placed on the form

   The computer must choose and display a random number from 1 to 99 on the screen.

   As soon as the user clicks the [Start] Button, the counting numbers from 1 to 99 must start appearing in a RichEdit. The user must attempt to click the [Stop] Button as the number chosen and displayed by the computer (at the beginning) rolls past. If the user manages to stop in time, a suitable congratulatory message must be displayed.

   Extend the game yourself. For example, you can add a RadioGroup to indicate levels of difficulty such as Easy, Difficult and Advanced. The speed at which the numbers roll by (determined by the Sleep statement) can be set depending on the degree of difficulty selected. You can also display the number of times a user played the game and the number of times he won.
3. Write your own program where you use the While loop for vertical or diagonal movement of a figure over the screen.

The Repeat loop

The Repeat loop is a conditional iterative structure, just as the While loop. The number of times it is executed is also determined by a condition. With a Repeat loop, however, the condition(s) is tested *at the end of the loop,* and not at the beginning as is the case with the While loop.

Consider the following example of a Repeat loop:

```
begin
    iNumber := 1;
    iTotal := 1;
    repeat
      inc(iNumber);
      iTotal := iTotal + iNumber;
    until iTotal > 1000;
    lblDisplay.Caption := IntToStr(iNumber);
end;
```

This program will determine how many counting numbers need to be added for the total to exceed 1000.

Note the following:

- The instructions within the Repeat loop will ***always*** be executed at least once because the terminating condition(s) is tested at the *end* of the loop.
- The Repeat loop does not require a **Begin** and an **End** (it's optional).
- The condition(s) indicates the circumstances which causes the loop to *terminate*, as opposed to the While loop where the condition(s) under which the loop *continues to be executed* needs to be specified.

General structure

Repeat

< Instruction(s) >

Until < terminating condition(s) >

- There is no semi-colon after the Repeat.
- There is no **Begin** and **End** statements which indicate the start and end of the loop.

The 'ICT-principle' holds true for the Repeat loop:

Initialise:	The variable(s) tested at the end of the loop must be assigned value(s) *before* the loop starts
Change:	The variable(s) used in the test at the end of the loop needs to be changed inside the loop otherwise the condition(s) will always be False, causing the loop to run infinitely (an infinite loop).
Test	The variable(s) is tested at the end of the loop. If the condition(s) is *False*, then the statement(s) within the loop is executed again.

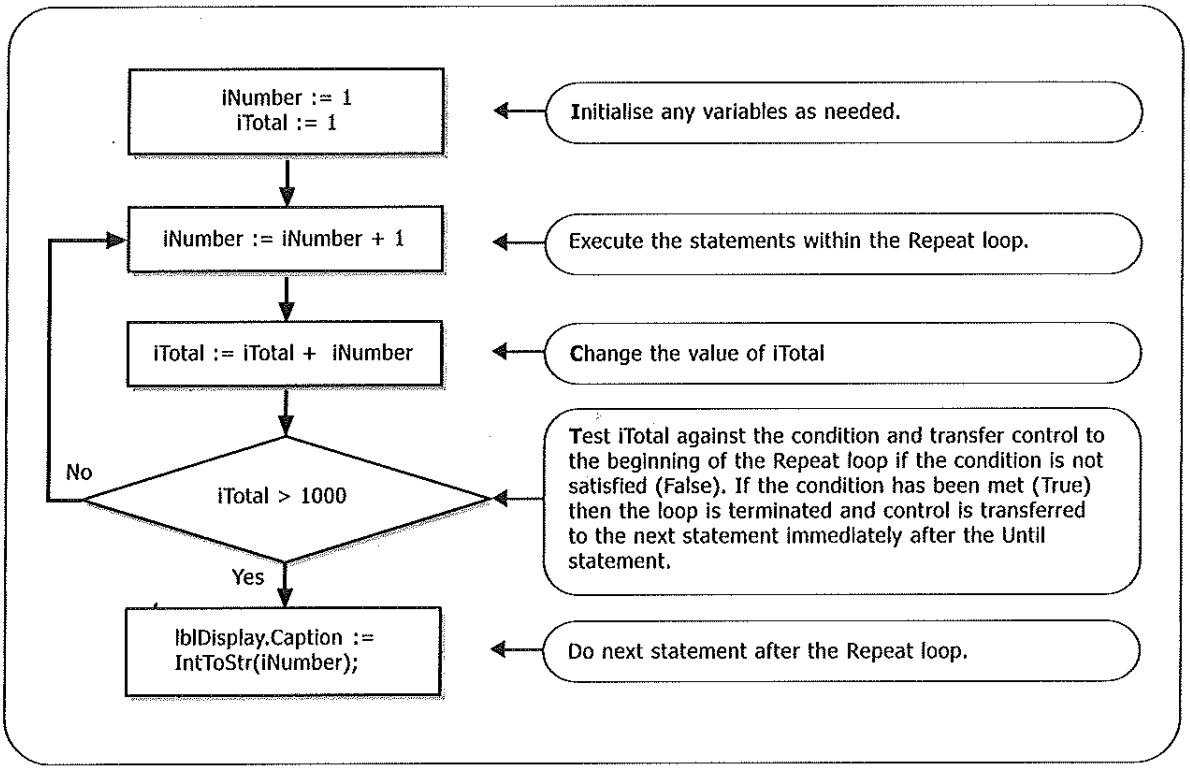
Note: An *infinite loop* will arise if the processing that takes place in the loop is such that the condition(s) is never met.

Flowchart of the Repeat loop:

The loop structure:

```
iNumber := 1;
iTotal := 1;
repeat
    inc(iNumber);
    iTotal := iTotal + iNumber;
until iTotal > 1000;
lblDisplay.Caption := IntToStr(iNumber);
```

can be represented in flowchart form as:



Tip

Be careful with conditions involving numbers and testing for equality. For example, if we made the terminating condition in the example: *iTotal = 1000*, the loop will become infinite as the total will at some stage become 990 and then 1035, in other words, never equaling 1000 exactly.

Rather test for *iTotal >= 1000* in order to make sure that the condition will be true at some stage.

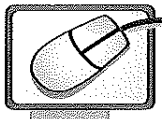
Often the condition needs to be *greater or equal to* rather than just *equal to* with conditional loops, for this very reason.

The use of the Repeat loop in problem solving

To decide whether a Repeat loop is required in the solution of a problem, you can ask similar questions as was the case with the While loop namely:

- Is there repetition of some code (activities)?
- Is the number of repetitions unknown?
- When must the repetition stop and what is the condition required for it to stop?

What you have to take into account when using a Repeat loop is that the code inside the loop will ALWAYS be executed at least once. In most cases this is what you want to happen, but there are situations where you may not want the loop to execute depending on certain conditions, in which case a While loop is required.

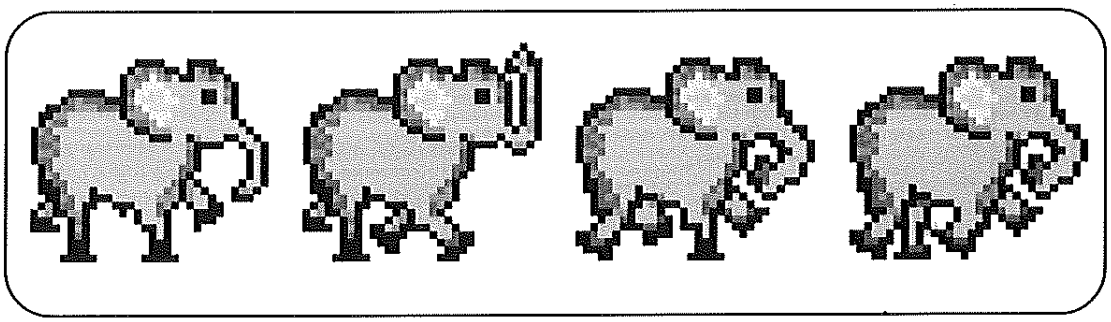


Using a Repeat loop to control the movement of an object

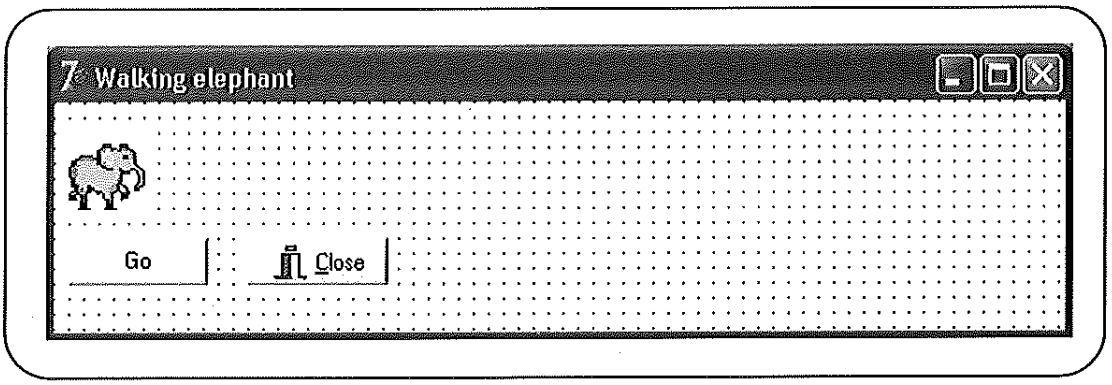
Activity

We are going to let an animal walk over the screen. He must walk until he reaches the end of the screen.

Use Ms Paint to create bitmap pictures. The following are examples of what you can draw. Save them as Walk1.bmp, Walk2.bmp, etc.



Then create an interface similar to the following:



Add the following code to the OnClick Event handler of the [Go] Button:

```
var
  iCount : integer;
begin
  iCount := 0;
  repeat
     Inc(iCount);
     if iCount = 4
        then iCount := 1;
     imgElephant.Picture.LoadFromFile ('Walk' + IntToStr(iCount)+ '.bmp');
     imgElephant.Left:= imgElephant.Left + 10;
     Sleep(100);
     frmLoop.Repaint;
  until imgElephant.Left > (frmLoop.Clientwidth - (imgElephant.Width + 10));
end;
```

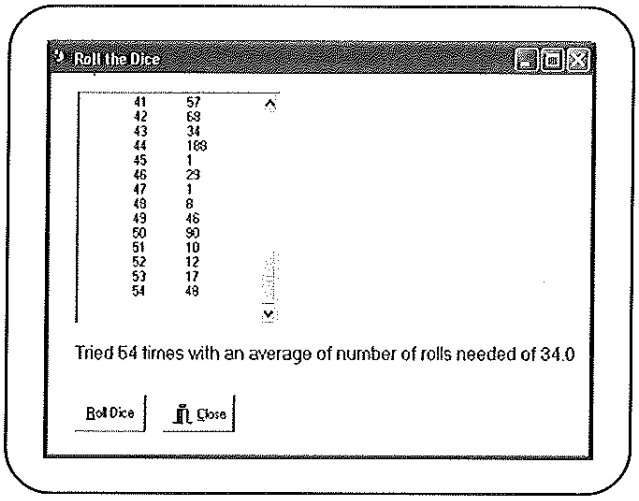
Save and run your program.

Compare this with the Activity on Page 115. You can see that you can often use either a Repeat loop or a While loop in a program with the same result.

Activity

Using the Repeat with the AND

We want to work out the odds of getting a 'double six' with the rolling of two dice. The user must be able to click a Button and the computer must simulate the throwing of two dice by generating two random numbers from 1 to 6. The number of throws required each time before a double six is thrown must be displayed in a RichEdit and the average number of throws required needs to be displayed in a Label.



- Because the dice must be repeatedly thrown we have *repetition*.
- The *number of times* the dice must be thrown each time to get a double six *is not known*. We therefore need a conditional loop (i.e. either a Repeat loop or a While loop).
- Obviously the dice must be thrown at least once; therefore the loop construct required is a Repeat loop in this case.
- The loop must repeat until die1 shows a six and die2 shows a six, which represents the terminating *condition* of the Repeat loop.

Step 1: Construct the IPO table

Input	Processing	Output

None	Increment number of tries (Number of times user clicks (Roll dice) button)
	Set Number of rolls to zero
	Repeat
	  Increment number of rolls in this try
	  Generate two random numbers from 1-6
	until both show a six
	Recalculate average number of rolls needed
		Rolls needed per try
		Average number of rolls needed in total

Step 2: Choose and declare variables

Globally:

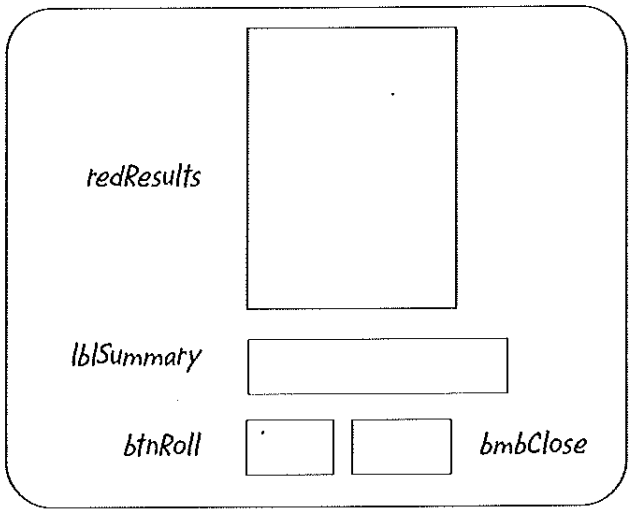
```
var
  iTries, iTotal : integer;
```

Locally in the Roll Procedure:

```
var
  rAverage                          : real;
  iRolls, iDie1 , iDie2, iSumDice   : integer;
```

Step 3: Design the user interface

Decide on required components:



Step 4: Code the Event handlers

Code the OnActivate Event handler of the form

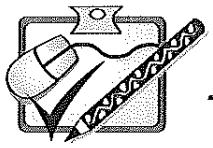
```
Randomize;       // reset random number generator
iTries := 0 ;    // initialise count of number of 'rounds' of throwing
iTotal := 0 ;    // initialise total of times rolled to get double 6
```

Code the Event handler btnRoll

```
begin
  Inc(iTries); // number of 'rounds' attempted to get a double 6
  iRolls := 0; // Initialise number of rolls in this 'round'
  repeat
    Inc(iRolls);                // increment total number of rolls
    iDie1 := Random(6)+1;        // simulate rolling of first die
    iDie2 := Random(6)+1;        // simulate rolling of second die
  until (iDie1 = 6) AND (iDie2 = 6) // terminate when both are six
  iTotal := iTotal + iRolls;   // add on to total number of throws
  rAverage := iTotal / iTries;  // re-calculate average number of rolls
  redResults.Lines.Add(#9+IntToStr(iTries)+#9+IntToStr(iRolls));
  lblSummary.Caption := 'Tried ' + IntToStr(iTries) +
                  ' times with an average of number of rolls needed of '
                  + FloatToStrF(rAverage,ffFixed,6,1);
end;
```

Checkpoint: Try the following yourself

1. A school's maximum capacity is 1000 learners. Write a program that asks the user for the current number of learners in the school, as well as the percentage growth in numbers per year. The program must indicate the growth in numbers per year until the maximum capacity is reached. The number of years it will take to reach this maximum capacity must also be displayed.
2. It takes 2 minutes for an average oven to raise its temperature by 10°Celsius when switched on. Write a program to determine how long it will take the oven to reach a certain temperature (e.g. 300°Celsius) typed in by the user. The program must use a RichEdit to display the time against the temperature (every 2 minutes) in table format until the required temperature is reached. The user must type in the required temperature in an Edit. A [Display] Button must trigger the Event handler.



Test, Improve, Apply

Written Exercises

1. Consider the following For loop:

```
for K := M to N do
begin
// instructions
end;
```

How many times will the instructions in the For loop be executed if:

a) M > N

b) M = N

c) M < N

2. Consider the following code:

```
iNum := 0;
for K := 50 downto 47 do ;
inc(iNum);
lblAnswer.Caption := IntToStr(iNum);
```

What will be displayed by *lblAnswer* once this code is executed?
Briefly explain your answer.

3. Consider the following code:

```
iSum := 0;
iTotal := 1;
for K := iNum1 to iNum2 do
begin
  iSum := iSum + iNum1;
  iTotal := iTotal * iNum2;
end;
lblAnswer.Caption := IntToStr(iTotal);
```

a) What will be displayed by *lblAnswer* once this code is executed if *iNum1* has a value of 5 and *iNum2* has a value of 7?

b) What will appear in *lblAnswer* once this code is executed if *iNum1* has a value of 5 and *iNum2* has a value of 7 and the *Begin* and *End* pairing was removed?

4. Rewrite the following For loop as a While loop:

```
for K := iNum2 downto iNum1 do
begin
  inc(iSum);
  iTotal  := iSum + iTotal;
end;
```

5. The following given statements contain syntax errors. Write the statements down correctly without any syntax errors.

   a) `for iCount = 1 upto 10 do`

   b) `while iNumber < 100 and iNumber > 0 then`

   c) `repeat`

   ```
   ..................
   until iValue < 100 do
   ```

6. The following program segment is an effort to determine how many times the value of 3 can be subtracted from the value of 10. The reduced value must be displayed on the screen in a Label with every subtraction.

```
iCount := 0;
iNumber := 10;
while iNumber <> 0 do
   begin
      iNumber := iNumber - 3;
      iCount := iCount + 1;
      lblNumber.Caption := IntToStr(iNumber);
   end; {while}
lblCount.Caption := IntToStr(iCount);
```

   a) Determine the logical error in this program segment.

   b) Rewrite only the While statement in order to let the program function correctly.

7. The given program code has to display the following sequence of numbers in a RichEdit:

   48 36 24 12 0

```
begin
  iNumber := 0;
  while iNumber > 0 do
     begin
        iNumber := 48;
        iNumber := iNumber - 12;
     end; {while}
  redNumbers.Lines.Add(IntToStr(iNumber));
end;
```

   a) Use a trace table to determine what the output of the program will be.

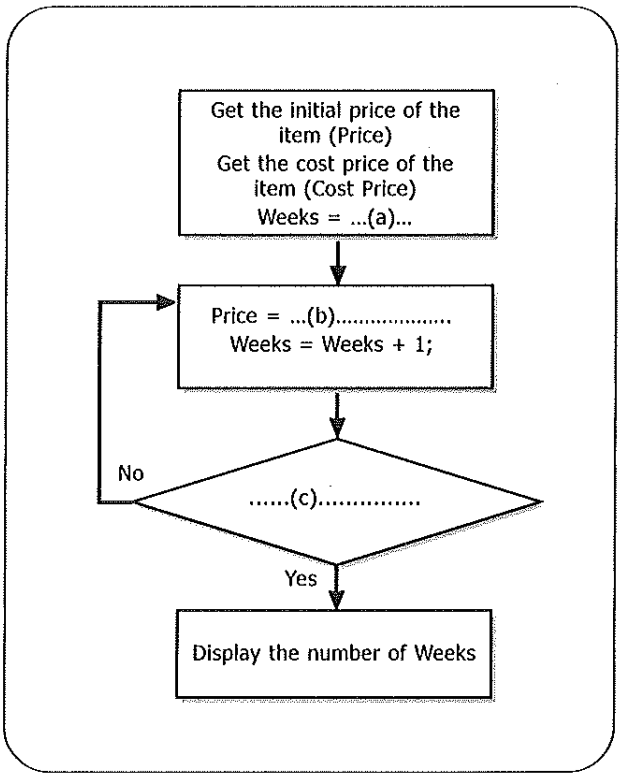
   b) From the information in the trace table you can see that the output is incorrect. Rewrite the given code so that the correct output will be generated.

8. A shop has a sale on some of their stock. The current prices of these items are reduced by 10% each week. The program must determine how many weeks an item will stay on sale. The reduced price cannot be less than the cost price. Study the following flowchart compiled for this program.

   a) Write down the missing statements (a), (b) and (c)

   b) Which loop structure is used in this flow chart?

   c) Use the given flowchart and write down the program code for the Event handler.



Practical Exercises

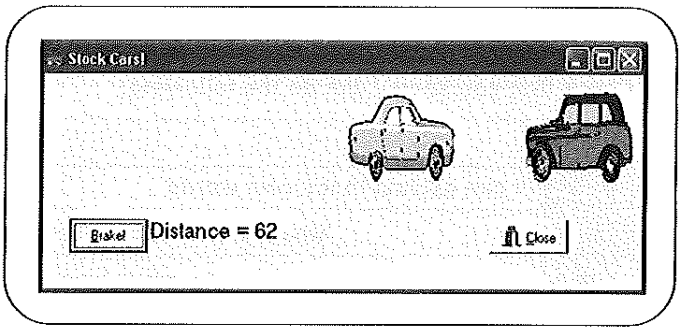
9. The water level of a dam fluctuates each year, as the amount of water being used by the population and the annual inflow of water into the dam varies.
   Write a program that asks the user for the current volume of the dam, the annual inflow of water into the dam and the amount of water used annually by the population. The program must determine how many years it will take for the volume of the water in the dam to *either* double *or* halve in capacity in terms of its original volume. The program must display the annual volume of water in the dam in a RichEdit.
   Note: There are two conditions that need to be tested for in this problem.

10. A code consisting of four unique digits must be generated as part of a system that creates passwords. Write a program to generate the four random digits, which are restricted to the range 1..9. The four digits must be different.
    If you like a challenge, you can write another program to calculate how many possible combinations there are!

11. A Tri-Prime number is a number with at least FOUR factors other than itself and 1. The first Tri-Prime number is 12 because its factors other than itself and 1 are the following: 2, 3, 4 and 6.
    Write a program to determine if a number input is a Tri-Prime number or not.

12. We want to have two cars racing towards each other. The user needs to click a button to stop them before they crash! The program must provide feedback of how far part the cars were when they were stopped!

    Tip: It is very easy to alter the program to get it to stop when the cars crash by looking at the distance between them and setting the global variable bStop to True if the distance is zero (or less).



13. When a car is purchased, monthly interest is charged on the outstanding balance of the purchase price. In addition, the buyer pays off a fixed payment or instalment each month.

    For example, if a buyer purchases a car worth R125000 initially at an interest rate of 2.5% and the instalment or payment is R3500 per month, then the interest after the first month is calculated as follows:

    Interest = 2.5% X 125000 = R3125.

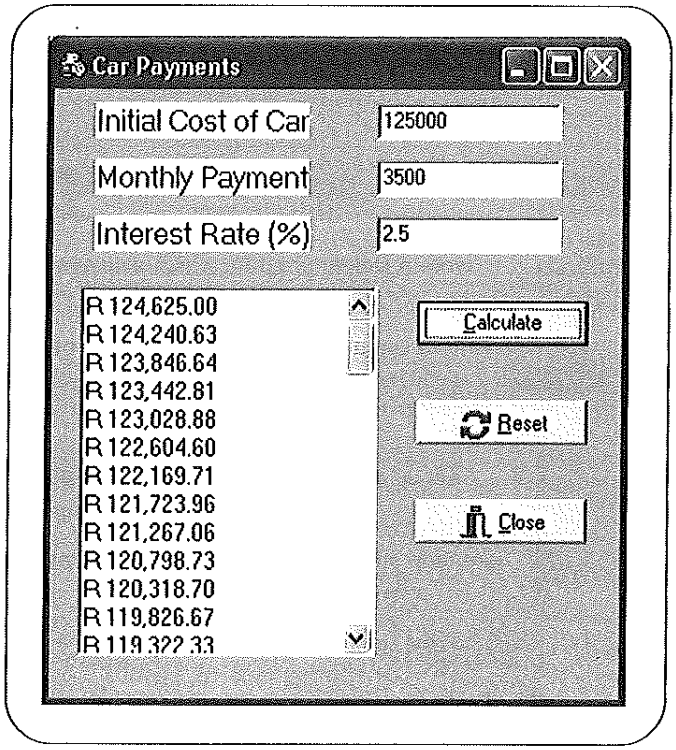
    The new balance (outstanding amount) is calculated by adding the interest to the previous balance and subtracting the instalment of R3500 as follows:

    Balance = 125000 + 3125 - 3500 = 124625

    These calculations will be repeated monthly on the outstanding amount (balance) in order to determine the new balance.

    Write a program to ask the user to type in the initial price of the car, the monthly payment and the interest rate as a percentage.

    The program must then determine how long it will take before the outstanding balance is *less than half* the original cost of the car. Only the outstanding balance per month needs to be shown in the RichEdit.



    Display the number of months at the bottom of the list in the RichEdit.

14. The following sequence of numbers is known as a Fibonacci sequence:

    0 1 1 2 3 5 8 13 ...

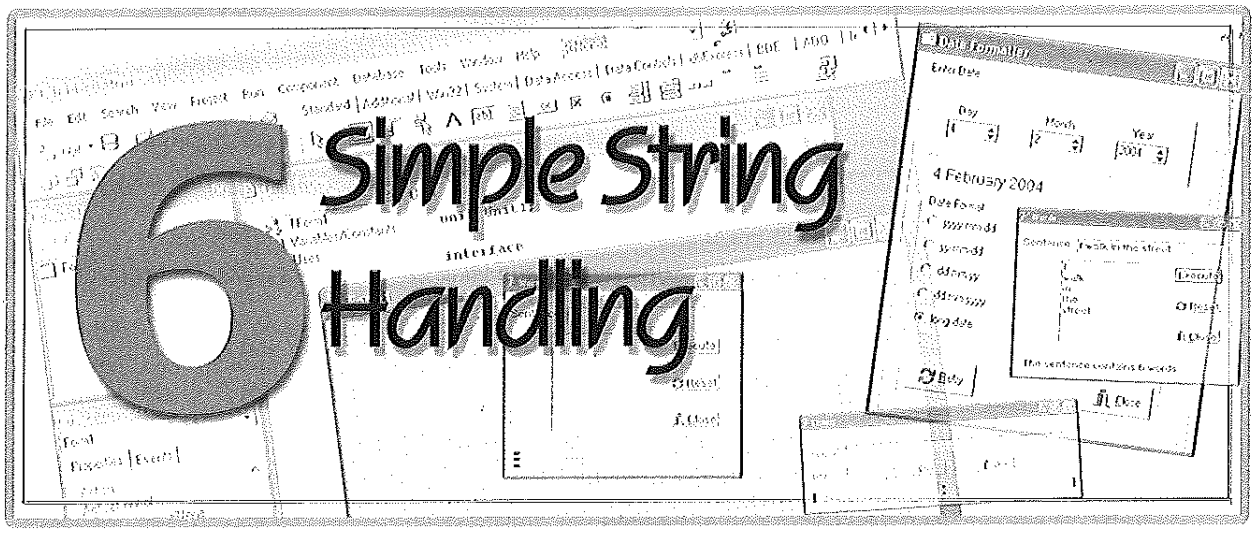
    Open the program called FibonacciP on the CD. The program has to determine and display the first 15 terms of the Fibonacci sequence. The output produced by the program is not correct. Use the Delphi debugger to identify and correct the errors.

    Tip: First study the Fibonacci numbers before you look at the code and try to understand how the numbers are generated. The first two values are constant values. The third value is generated from the first two given values.

Test yourself

15. Check that you know / can do the following:

		✓
Knowledge	**I can explain in my own words**	
	what the difference is between a conditional and an unconditional loop	
	for which type of programs the use of a For loop is essential	
	what the ITC principle for the While and Repeat loops entails	
	what the difference is between a While loop and a Repeat loop	
	for which type of programs the use of a While loop or a Repeat loop is essential	
	how the Application.ProcessMessage method is utilised in a Delphi program	
	how a trace table and Delphi's debugger can be utilised	
Skills	**I can do the following**	
	write down the general structure of a For loop	
	write down the general structure of a While loop	
	write down the general structure of a Repeat loop	
	write a Delphi program which displays a series of numbers of the screen using a For loop	
	write a Delphi program which repeats basic mathematical calculations through the use of loop structures	
	set up a trace table	
	set up a Watch List and a breakpoint and then use the Delphi debugger to trace the content of variables during execution of a program	
	use loop structures and the Canvas property of a form to create the effect of movement on the screen	



After you have complete this chapter, you should be able to:

- explain in which situations you would typically use the String data type
- use basic String functions and procedures to process strings
- process a string character by character using a loop.
- write programs, which read in data of the String type and manipulate and display the results of the processing

Introduction

What is a string? Simply put, a string is a sequence of characters. These characters include, amongst others, any character on the keyboard.

Often data that we expect, at first glance, to be of another type, turns out to be best stored in a string once we start working with the data. Examples of such data include:

- Telephone numbers
  These seem to be just integers but often the user wants to enter other characters – e.g. (011) 23 4514.
- ID numbers
  We don't really envisage doing mathematical operations with ID numbers, so we don't need them to be stored as numbers. Often we have to split ID numbers into sections to get bits of data from them such as the date of birth. We can easily do this with strings.

- Account numbers
  They also need to be manipulated and sometimes they contain a mixture of letters and numbers.

Data stored in strings often needs to be manipulated and changed. You might need to extract pieces of data out of strings, change parts of strings, etc. All of this is called *string handling* or *string manipulation.*

To be able to work with and manipulate strings you have to understand how the contents of a string are stored in memory.

How strings are represented

A string is represented in memory as a sequence of numbered spaces. The memory spaces reserved for a string variable are normally dynamic. A dynamic number of spaces means that the number of spaces grows and shrinks as needed.

Example

```
var
  sName :string;
```

If we execute the following line of code

```
sName := 'Peter';
```

the data in memory will look as follows:

P	e	t	e	r
1	2	3	4	5

We can refer to a specific character in a string by using the index. The index refers to the position of the character in the string.

sName[1] will therefore be equal to 'P', sName[2] ← 'e', etc.

Checkpoint: Using a specific character in a string

The school has two sports teams, namely the Lions and the Leopards. Learners are divided into the teams according to the first letter of their surnames. All learners with a surname that starts with the letters A – M are in the Lion team, and all learners whose surnames start with the letters N – Z are in the Leopard team.

Write a Delphi program that will display the team a learner belongs to, when his surname is entered.

Simple processing with string functions

There are Delphi functions used specifically to manipulate numerical values, such as Trunc and Sqrt. There are also functions used to manipulate strings.

Length function

Length is a function that accepts a string and returns a number indicating the number of characters currently stored in the string. The Length function will always return an Integer value.

Voorbeeld

```
sName := 'Elizabeth';
iHowLong := Length(sName);     // The value of iHowLong will be 9.
```

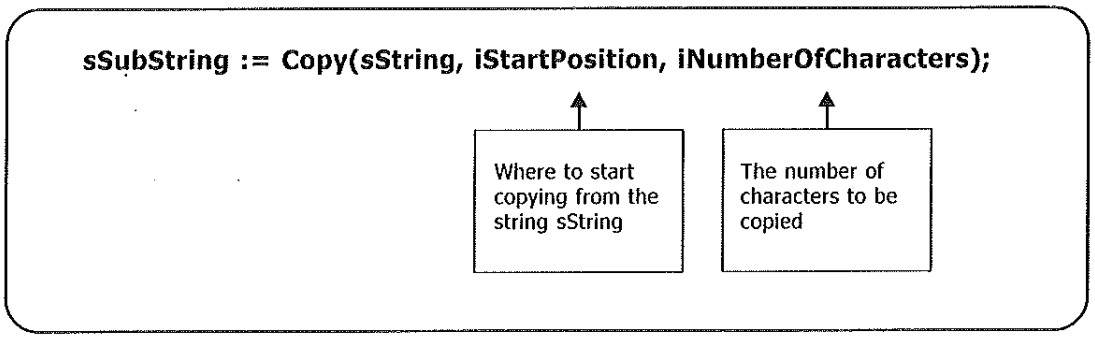
Checkpoint: Using the Length function

Each learner in the Computer Studies class gets a turn to be responsible for the neatness and general appearance of the computer lab. One of the parents offered to make a set of wooden blocks that can be used to display the name of the learner that is on duty. These blocks will all be of the same size and will slide into a frame. To be able to determine the length of the frame the parent needs to know the number of characters of the longest name in the class.

Write a program that will display the length of the longest name, if the names of all the learners in the class are entered.

The Copy function

Copy is a string function that copies a sub-string from a string. A String function means that it gives a string as a result. The original string itself remains unchanged.



Examples

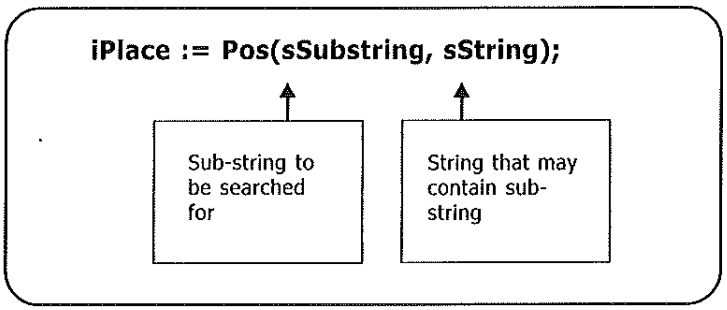
```
var
  sOld, sNew1, sNew2, sNew3   : string;
  cCharac1                    : char;
begin
  sOld := 'I love Computers';
  sNew1 := Copy(sOld, 1, 1);  // sNew1 contains 'I'
  sNew2 := Copy(sOld, 3, 4);  // sNew2 contains 'love'
  sNew3 := Copy(sOld, 20, 1);  //  sNew3 contains ''
  cCharac1 := Copy(sOld, 1, 1); // Invalid - Copy returns a string
  cCharac1 := sOld[1];          //  Valid
end;
```

Since sOld contains only 16 characters, an empty string will be stored.

You can't assign the result of **Copy(sOld, 1,1)** to a Char data type – you will get the error *type mismatch* although the result consists of only one character. The result of the Copy function is of type String. You can however assign **sOld[1]** to a character.

The Pos function

The Pos function finds the position of a sub-string in a string. An integer value is returned.



Examples

```
var
  sOld, sSearch1, sSearch2    : string;
  iPlace1, iPlace2            : integer;
begin
  sOld := 'I love Computers';
  iPlace1 := Pos('love', sOld); //iPlace1 contains 3
  iPlace2 := Pos('o', sOld);  // iPlace2 contains 4
  sSearch1 := 'put';
  iPlace1 := Pos(sSearch1, sOld); // iPlace1 contains 11
  sSearch2 := 'Put';
  iPlace2  = Pos(sSearch2, sOld);// iPlace2 contains 0
end;
```

The position of the *first* occurrence of an "o".

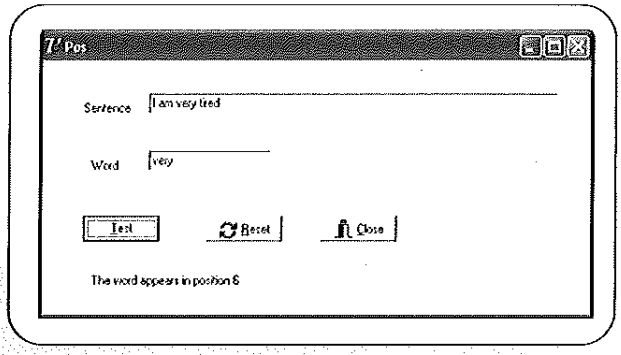
The sub-string does not appear in the string, as the search is case-sensitive.

Activity

Using Pos, Copy and Length

1. We want to determine whether a certain sub-string appears in a string and display the position in which the sub-string appears.

   Create the following interface:

   Enter the following code in the Event handler for the [Test] Button:

```
var
  sSentence, sWord  : string;
  iPosition         : integer;
begin
  sSentence := edtInput.Text;
  sWord := edtWord.Text;
  iPosition := Pos(sWord, sSentence);
  if iPosition > 0
     then lblOutput.Caption := 'The word appears in position ' +
                               IntToStr(iPosition)
     else lblOutput.Caption := 'The word does not appear';
end;
```

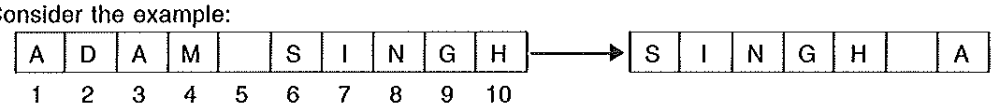
   Also type in the code for the Reset BitBtn.

   Save and run the program.

2. Names and surnames are entered as one string in the format NAME SURNAME. Convert it to the format SURNAME INITIAL. (You may assume that *one* name and a surname will be entered in an Edit.)

   E.g.: **Adam Singh** has to be converted to **Singh A**

   Consider the example:



   First describe the steps to be executed in your own words:

   Algorithm:

   Initial ← NameSur[1]

   Determine the position of the first blank. (Blank ← Pos(' ', NameSur))

   The Surname consists of all the characters *following* the blank *up to the end of the string.*

   (Surname ← Copy (NameSur, Blank+1, Length(NameSur) - Blank))

   Write the program to do the conversion mentioned above.

Checkpoint: Try the following yourself

1. A person's name and surname are entered in an Edit. A password needs to be compiled by combining the first three letters of the name with the first three letters of the surname.

A	D	A	M		S	I	N	G	H

→

A	D	A	S	I	N

Write a Delphi program to compile the password.

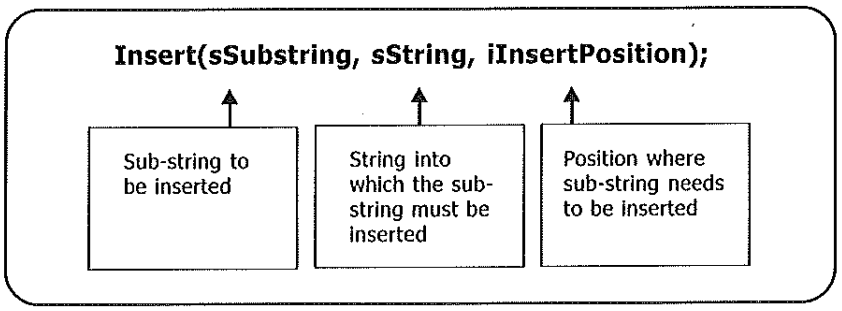
2. Read a person's ID number and then display his date of birth.
E.g: **9012260009883** will become **26 December 1990.**

Simple processing with string procedures

The Insert and Delete procedures are Delphi procedures used specifically to manipulate strings. This will change the original string by adding or deleting text.

The Insert procedure

Insert is a procedure that inserts a sub-string into a string. The original string will therefore change.



Example

Assume a string called sTelNo contains a telephone number with a dialing code of 012. The digits '42' need to be added to the dialing code.

Original contents of sTelNo:

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
(	0	1	2	)	5	6	5	4	3	2	1

The first character of the sub-string '42' will be in the **5th** position in the altered string.

[1]	[2]	[3]	[4]	**[5]**	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
(	0	1	2	**4**	**2**	)	5	6	5	4	3	2	1

The following Delphi code will insert the sub-string '42' into the original string:

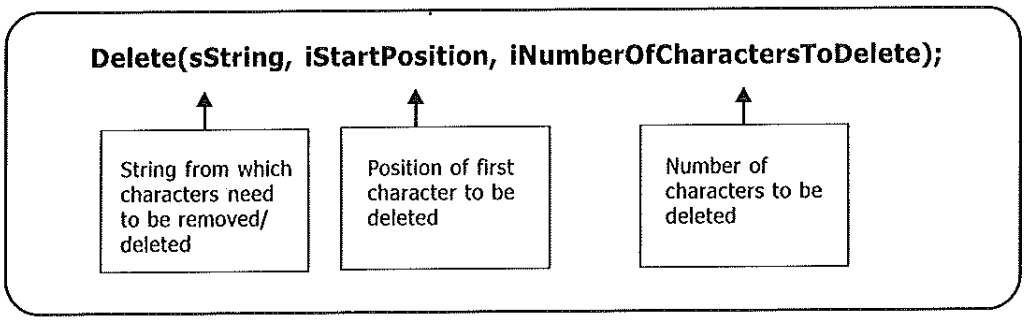
```
Insert('42', sTelNo, 5) ;
```

The rest of the characters will be 'moved to the right' to accommodate the sub-string. The length of the altered string will now be 14 since two additional characters were added.

The Delete procedure

Delete is a procedure that deletes a number of characters in a string. The original string will therefore change.

Example



Assume a string called (sTelNo) contains a telephone number with a dialing code of '01245'. The dialing code must be changed to '012'.

Original contents of sTelNo:

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
(	0	1	2	4	2	)	5	6	5	4	3	2	1

The digits '42' in positions **5** and **6** need to be removed from the dialing code.

The following Delphi code will delete the sub-string '42' from the original string:

```
Delete(sTelNo, 5, 2) ;
```

The number **5** in the Delete procedure indicates the position of the first character to be deleted in sTelNo. The number **2** indicates how many characters must be deleted, starting at the character in position 5. The rest of the characters will be 'moved to the left' to fill up the spaces where the two characters were deleted. The length of the altered string will now be 12 since two characters were deleted.

Contents of the string sTelNo after the Delete procedure has been executed:

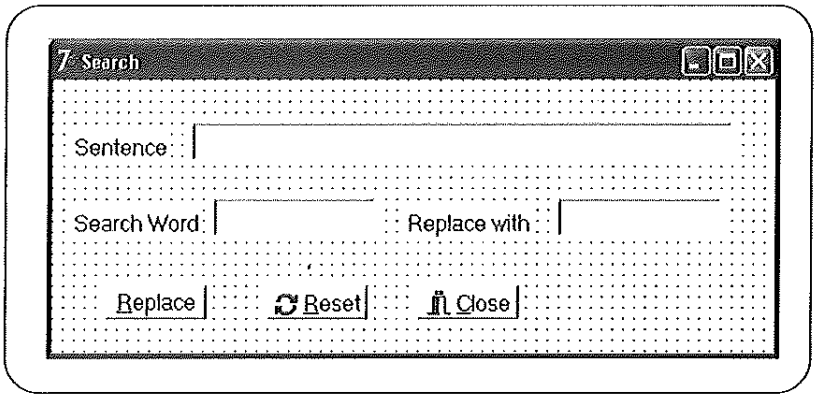
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
(	0	1	2	)	5	6	5	4	3	2	1

Activity

Using the Insert and Delete Procedures

We want to replace a certain word with another word in a sentence.

Create the following interface:



Look at the following IPO table

Input	Processing	Output
Sentence Search word Replace word	Find position of Search word Delete Search word Insert Replace word	Sentence

We are going to display the changes in the Edit where we entered the sentence and therefore will not need another component for output.

Type the code for the Event handler of the [Replace] Button.

```
var
  sSentence, sSearch, sReplace    : string;
  iPosition                       : integer;
begin
  sSentence := edtInput.Text;
  sSearch := edtWord.Text;
  sReplace := edtReplace.Text;
  iPosition := Pos(sSearch, sSentence);
  Delete(sSentence, iPosition, Length(sSearch));
  Insert(sReplace, sSentence, iPosition);
  edtInput.Text := sSentence;
end;
```

Also add code for the Reset BitBtn.

Save and run the program.

Checkpoint: Try the following yourself

When telephone exchanges are updated, changes in some telephone numbers and area codes are necessary.

Suppose that all numbers that have a three-digit area code of the form 09? (e.g. 092, 093 etc.) must have their area code changed to 09nn where n is third digit of the area code (the digit after the 9). For example 092 becomes 0922 and 096 becomes 0966.

In addition all telephone numbers starting with 34 or 36 must be changed to 374 and 376 respectively. For example 011-343969 becomes 011-3743969 and 095-3651270 becomes 0955-37651270.

Write program to read in a telephone number and apply these new changes where necessary. The new number must then be displayed. The original number gets displayed if no changes are required.

The following IPO table will help you:

Input	*Processing*	*Output*
Telephone number	*Copy dialing code and number part of telephone number to separate variables.* *If first two digits of dialing code are 09 then copy 3'rd character and insert it at 4'th position.* *If first two digits of actual telephone number are '34' or '36', insert a 7 in the third position.* *Form a new string with area code + '-' + number*	*"New' telephone number*

Using loops to work with strings

Character by character processing of strings almost always involves a ***For*** loop and the ***Length*** function.

```
for K := 1 To Length(sName) do
```

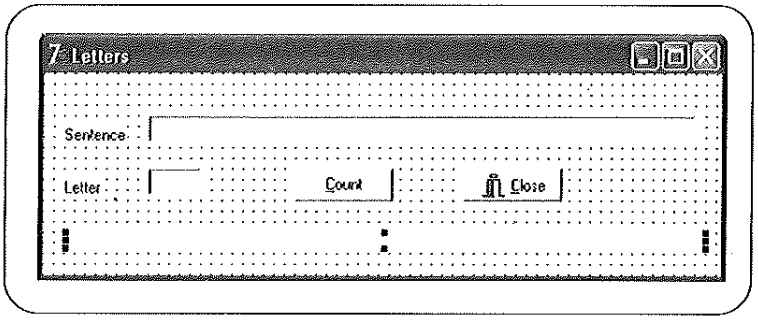
We use the For loop to step through the string, character by character. The counter variable of the For is used to point to each character, one by one. This can be done because each character in the string has an index or position within the string.

Working with a single character in a string

Activity

Use the For loop to determine how many times a letter appears in a string.

Create the following interface:



Consider the following IPO table

Input	Processing	Output
sSentence sLetter	iCounter ← 0 for K from 1 to Length(sSentence) do if sSentence[K] = sLetter Inc(iCounter)	How many times the letter appears i.e. iCounter

To make sure that the program will work whether the user type uppercase or lowercase letters, we will use the uppercase function to change the input to uppercase before we compare it.

```
if UpperCase(sSentence[K]) = Uppercase(sLetter)
  then Inc(iCount);
```

Complete the OnClick Event handler of the [Count] Button.

Checkpoint: Try the following yourself

1. Let the user enter a sentence. The program should convert the sentence to code which cannot be read easily.
   - Every letter in the sentence must be replaced with the next letter of the alphabet.
   - Do not replace spaces or other special characters.
   - The last letter of the alphabeth (z) must be replaced by the first letter in the alphabeth (a).

   You can extend the program by adding another Button to convert the coded sentence back to the original readable sentence.

   Hint: You can also use the Inc function with characters.

2. A company wants to determine the cost of SMSs to be able to decide if they should make use of SMS messages for communication. Assume the cost is 2 cent per character. Write a program so the user can enter a message and the program will calculate and display the cost of the SMS on the screen. There is no charge for a space, but all other characters are charged.

Building a new string

Certain problems require you to build a new string as you work through an existing string. You will therefore be able to keep the original string.

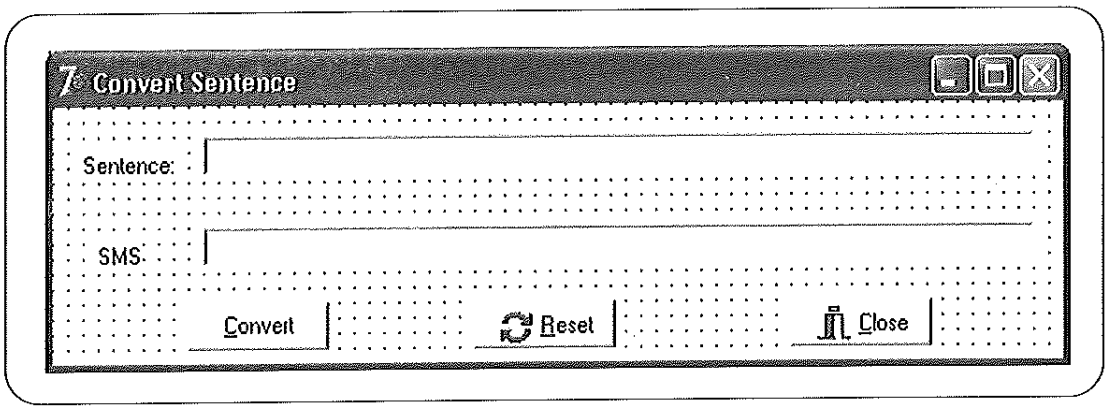
Activity

To build a string using characters from an existing string

Write a program that will drop vowels in a sentence to convert it to a shortened sentence.

Example:
Happy New Year pretty lady – are they going together 2 the party tonight?
becomes
Hppy Nw Yr prtty ldy – r thy gng tgthr 2 th prty tnght?

Create the following interface:



The important concept is to declare a second variable that can be used to build the new string.

Add the following code to the OnClick Event handler of the [Convert] Button:

```
var
  K                : integer;
  sSentence, sSMS  : string;

begin
  sSentence := edtInput.Text;
  sSMS := '';

  for K := 1 to Length(sSentence) do
     if NOT(upcase(sSentence[K]) IN ['A','E','I','O','U'])
        then sSMS := sSMS + sSentence[K];

  edtSMS.Text := sSMS;
end;
```

Checkpoint: Try the following yourself

1. Design a program to process a person's information to have it placed in a telephone directory. The program reads the person's first name, surname and telephone number. Each person's initial, surname and telephone number must then be placed on a separate line in a RichEdit. Only capital letters may be used in the telephone directory.

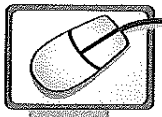
   Challenge: Display the items in the RichEdit neatly in columns so all the telephone numbers are displayed underneath one another next to the initials and surnames.

   Tip: Use the information in Appendix A.

Working with words

Sometimes it is necessary to identify words in a string. How can we subtract words from a string?

Words are separated by spaces. So we use that fact as the starting point to solve our problem.

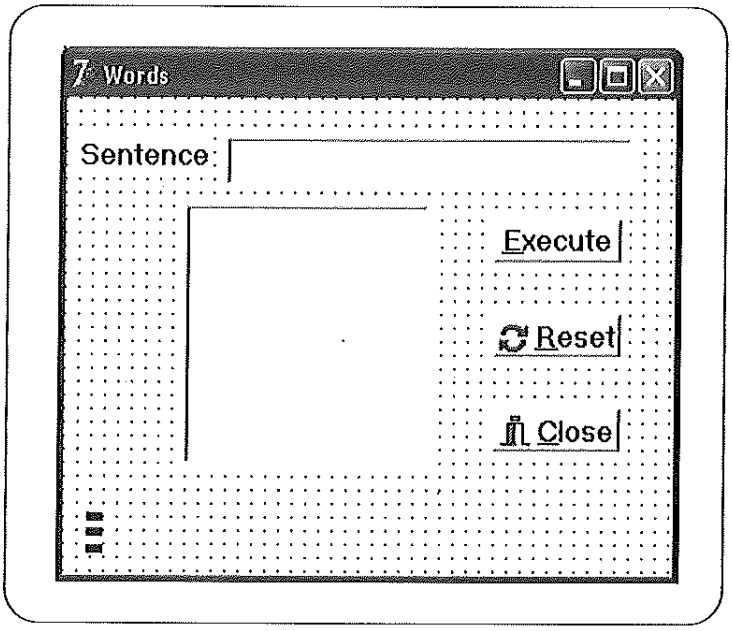


Activity

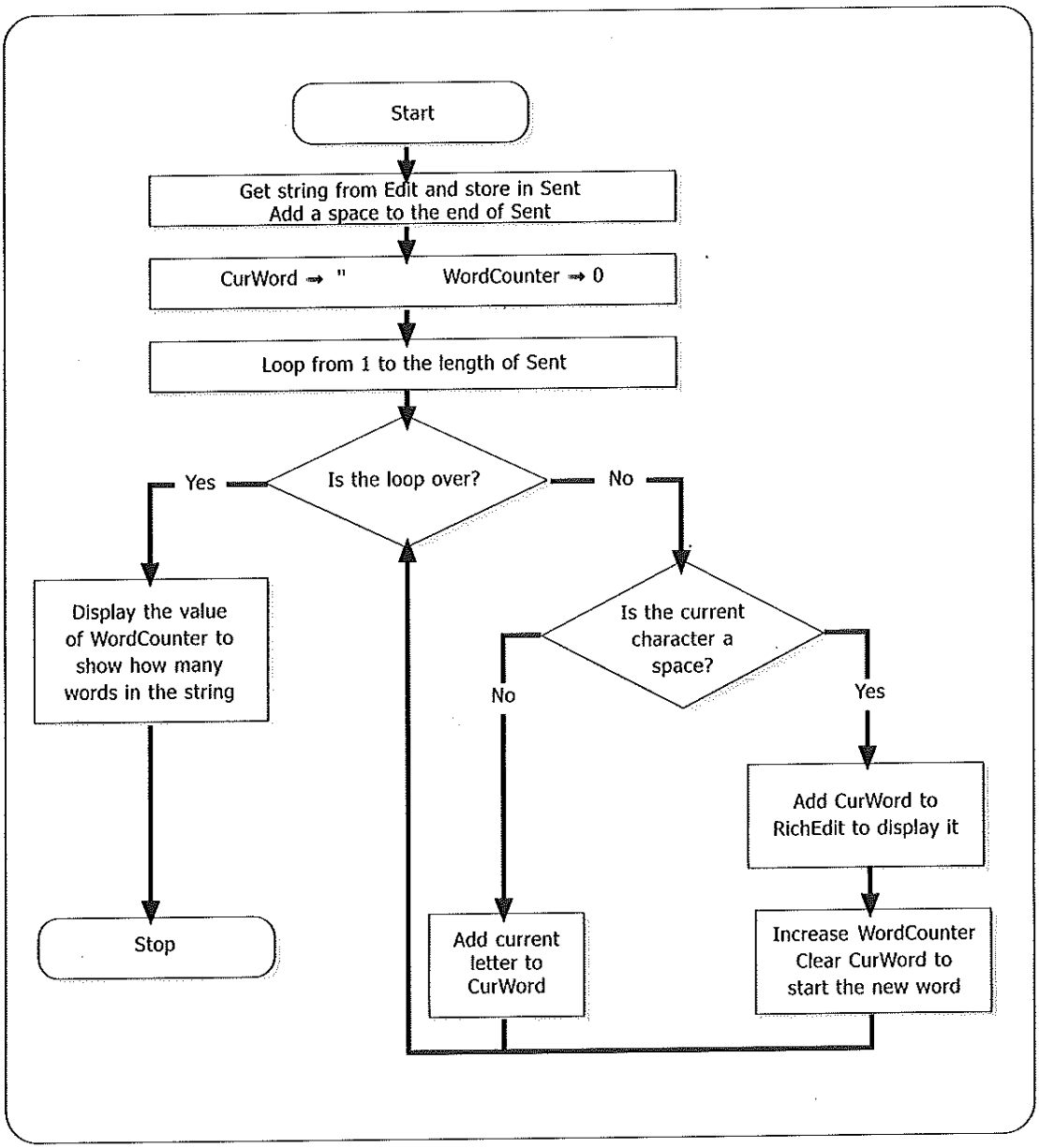
To extract words from a sentence

We are going to write a program that will extract the separate words from a sentence and display the words underneath each other in a RichEdit. The program will also count the number of words in the sentence.

Create the following interface:



We use a flow chart to explain the logic of the program:

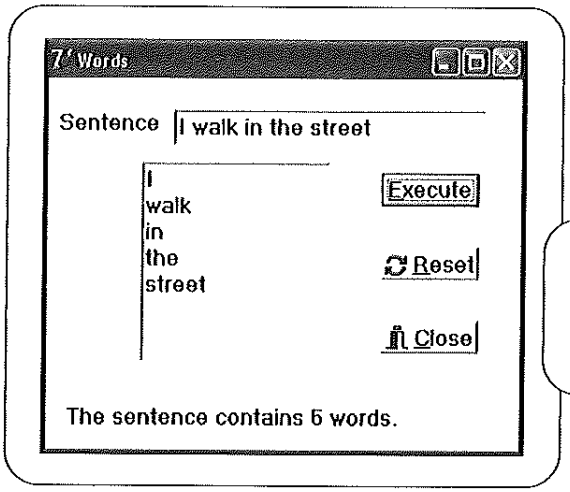


The code is actually quite simple.

```
var
  iCount, iWordCounter : integer;
  sSent, sCurWord      : string;
begin
  sSent := edtSentence.Text;
  sSent := sSent + ' '; // this is to ensure that last word is picked up
  iWordCounter := 0;
  sCurWord := '';
  redDisplay.Lines.Clear; // clear the lines of the RichEdit
```

```
for iCount := 1 to Length(sSent) do
    begin
      if sSent[iCount] = ' '
        then begin
               redDisplay.Lines.Add(sCurWord);
               sCurWord := '';
               Inc(iWordCounter);
             end
        else
             begin
               sCurWord := sCurWord + sSent[iCount];
             end;
    end;
  lblNoWords.Caption := 'The sentence contains ' + IntToStr(iWordCounter)
                        + ' words.';
End;
```

The following is an example of the program when executed:



Note

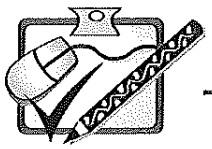
The code is not perfect – it can be fooled by, amongst other things, the user typing in more than 1 space between words.

Checkpoint: Try the following yourself

Improve the program done in the activity where we counted the number of words in a sentence to make provision if the user enters more than one space between the words.

There are two possible ways of solving this problem:

- First remove all the extra spaces.
- Determine the end of a word by using the fact that the character after a word must be a space.



Test, Improve, Apply

Practical Exercises

1. Every person who joins a specific company needs an account to log on to the computer system. The name of the account is constructed as follows:
   - The surname is converted into uppercase and then it is reversed – e.g. Smith becomes SMITH and is then reversed to become HTIMS.
   - The two-digit month number for the month that the person is born in is then added onto the end of this (e.g. January gets 01 and December gets 12).

   Write a program that accepts a person's name, surname and date of birth(yyyy/mm/dd). It must then generate and display the account name.

2. A company sends out bills at the end of the month. Write a program that asks the user to type in the amount owed and then does the following:
   - Calculates VAT at 14% and adds it to the amount.
   - Rounds off the cents to 2 decimal places and adds a R symbol.
   - Checks if the string (including the R and the decimal point) is less than 10 characters.
   - If this is so, it fills in the appropriate number of stars between the R and the first digit – e.g. R10.75 is 6 characters long. This becomes R••••10.75 (The string is now 10 characters long).

     The program must then display the string on the screen.

3. Write a Delphi application for athletes to register for a marathon. The athlete must enter the current date, his name and ID number. His age must then be determined using his ID number. The following algorithm will help you:

   Age = this year - birth year
   if birth month > current month
   then I am not quite that age so age = age - 1
   if, however, birth month = current month
   then if birth day > current day
   then I am not quite that age so age = age - 1
   display the age

   It is possible to instruct the program to get the current date from the computer's "clock" instead of letting the user type in the current date. The following code will determine the current day, month and year for you without having to type it in. This would make your program much smarter!

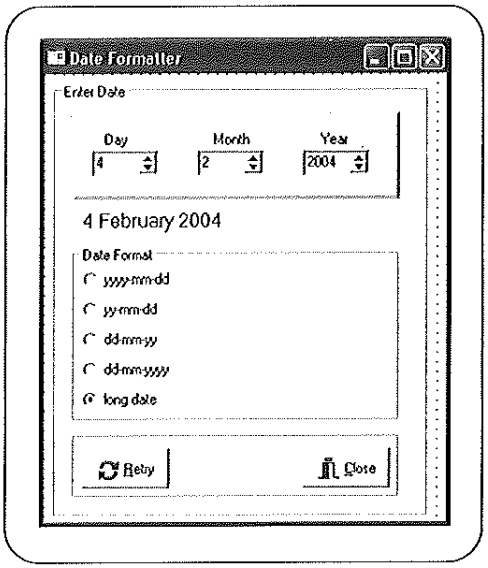
```
iYear  := StrToInt(Copy(DateToStr(Date),1,4));
iMonth := StrToInt(Copy(DateToStr(Date),6,2));
iDay   := StrToInt(Copy(DateToStr(Date),9,2));
```

4. Cell phone numbers are often stored in the format of: +CCnnnnnnnnn
   - The *CC* is a two digit number representing the country. (27 indicates South Africa and any other value represents an international number.)
   - The *nnnnnnnnn* part is the actual number with the leading zero omitted.

     The first two digits of the actual number indicates the service provider. This can be Vodacom (72 or 82), MTN (83 or 73) or Cell-C (84 or 74).

Design a program that reads in a cell phone number on the format described and displays whether it is a South African or overseas number, the name of the service provider as well as the actual number with the leading zero added.

5. Design and write a program that will read in a Day, Month and Year via three Spin Edits.
   The program must then format and display the date according to the selection that the user makes in the RadioGroup. Assume initially that all the values input for the Day are valid.
   Hints:
   If the day or month value input is a single digit, then its string representation will need to have a '0' added to the front of the string.
   One way of solving this problem is to 'assemble' the day (dd), month (mm) and year (yy) as separate strings, as well as the 'long year' (yyyy) and the actual name of the month represented (e.g. February). Then its just a case of putting them together in the right order according to the format choosen by the user!

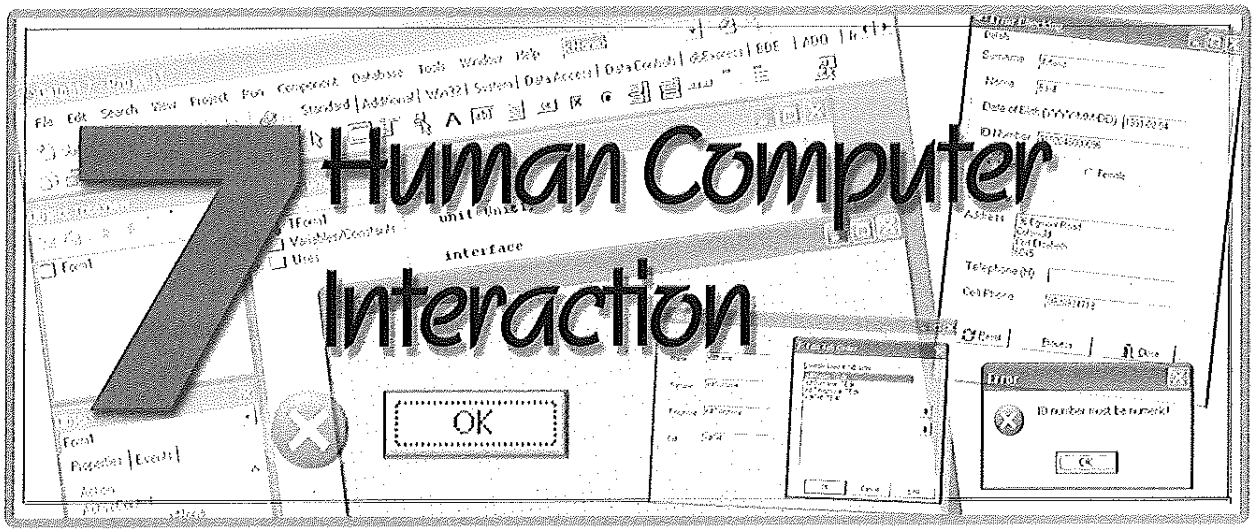


6. There is a word game where a player has to come up with a word starting with a letter that is the same as the last letter in the word the previous player used. The player who cannot do this loses the game. Write a program to simulate this game. Improve the game by flashing the last letter of the previous word on the screen.
   Extend the game by adding a time limit using the Timer.

Test yourself

7. **Make sure that you know / can do the following:**

		✓
Knowledge	**I can explain in my own words**	
	what type of data should be stored as a string	
	the purpose of the following functions/procedures: Length, Copy, Pos, Insert, Delete	
	the steps required to extract a word from a sentence	
Skills	**I can do the following**	
	use the Copy, Pos and Length functions and the Insert and Delete procedures for simple string manipulation	
	write a program to replace the occurrences of one word in a string with another word	
	write a program to process a string character by character	
	write a program to construct a string from an existing string using specified instructions	
	write a program to extract words from a sentence	



After you have complete this chapter, you should be able to

- describe the characteristics of a well-designed user-interface
- list techniques that contribute to the user-friendliness of the user interface
- explain the concept of defensive programming and why it is necessary
- outline some techniques and guidelines to make a programs more defensive and robust

In the 'earlier' days of computers, programs were largely used by specialised IT and engineering personnel only. With the advent of the PC, software could be used by people from all walks of life, many of them end-users. For some time little thought was given as to how people best interacted with computers and software in particular. Nowadays the design of the UI (user interface) plays a critical part in the commercial success or failure of a program. This has lead to the field of Human Computer Interaction (HCI) emerging. HCI is a topic all on its own but many of the principles are just plain old common sense. Yet software developers still often overlook them.

In this chapter we will begin by giving an overview of the basic principles, components and techniques that can contribute to a *good graphical user interface.* (You have already encountered many of these principles. There are, however, a few new concepts that you will acquire here.)

In the last section of the chapter we will look at techniques to deal with errors that happen whilst the program is running.

The Graphical User Interface

The common characteristics of a well-designed UI are:

- A simple layout of components on the screen, arranged in such a way that the user finds it easy to use.
- To communicate in an understandable way with the user e.g. displaying clear instructions and error messages.

The UI (and therefore your program) can become far more professional and easy to use by following a few simple principles.

Clear instructions

The user must always know what is expected. The following can be used:

Labels

Use Labels to display clear instructions that tell the user what is expected of him. This means that you must

- think about what you type into the Label, and
- use a short, descriptive sentence rather than just one word!

Icons and pictures

Use icons and pictures so that the user does not have to remember what the instructions on the Buttons they need to click, mean. This means using appropriate images and components, such as Bitmap Buttons. It makes life a lot easier if users can recognise what they have to do intuitively. Use graphics where you can – that is the whole point of working in a GUI!

Tips (Hints)

You've seen it in most of the windows programs you use –little 'tips' (hints) that pop up when your mouse hovers over a Button, icon or menu item. The tip (or hint as it is known in Delphi) contains a more complete description of what you can do with that Button or what input is needed in the Edit, etc. Using hints in your program is as easy as:

- Typing text into the Hint property of the component. (Almost every component has this property.)
- Changing the ShowHint property to *True*.
- Doing this for each component that you want to display a hint or tip for.

That's it! Your program now uses hints that will be displayed when the mouse hovers over each of the components.

Easy input of data

Make the input of data as easy as possible for the user.

Logical order of input

What is Tab Order?

- You know that the user can move between components on the form by pressing the Tab key - right? This is behaviour that is built into all windows programs.
- Tab Order is the order in which components become active when you use the tab key.
- By default the Tab Order is the order in which you placed the components on the form when you were designing the form.

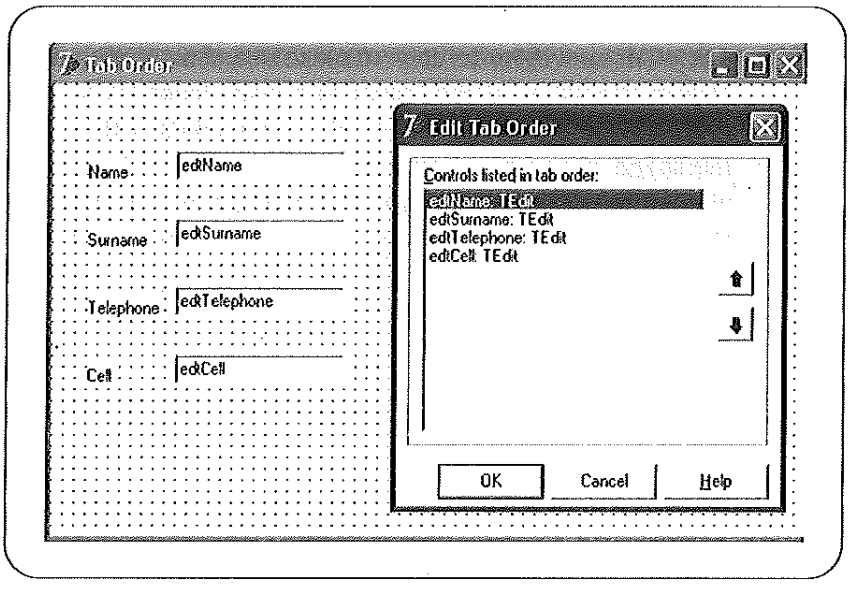
The Tab order plays an important part in making your program ***Keyboard friendly***. Use the **TabOrder** properties of the components of the form so that the data being entered is 'sequenced' so that it can be read like a book i.e. from top-to-bottom and left-to-right.

Knowing how to change the Tab Order of components is important. Often you add components onto the form at a later stage which means that the Tab Order might be completely out of sequence. When the user presses the ***Tab*** key the focus will shift around in an unpredictable way.

Changing the Tab order is simple:

- Right-click anywhere (on the form/components – it doesn't matter). Note that GroupBoxes and Panels have their own Tab order for all the components 'housed' within them.
- Choose ***Tab Order*** from the menu that pops up
- Use the dialogue that appears to move components up and down in the Tab Order – by clicking on the name of the component and then clicking on the up or down arrow.
- TIP – Name your components before trying to change the Tab Order or else it will be very difficult to distinguish between the different components on the list!

The above window shows the Tab Order dialog box. Change the Tab order by selecting the component and then clicking the Up/ Down arrows.



Determining the Focus

Set the *focus* to the first control that you expect the user to use. This means that it is the component that gets the first keyboard input.

Sometimes you want a user to be able to enter data into a series of Edits, click a Button to process it and then to start back at the first Edit entering new information. Because the Button was the last item clicked, it has the focus, so when they start typing after clicking the Button, nothing happens. They would have to click on the Edit to be able to enter data again. You want the program to set the focus back to the first Edit automatically. To do this you must add the following to the code in the Button's OnClick event:

```
edtName.Setfocus;
```

This command will transfer the focus to the Edit – as if the user had clicked on the Edit!

Shortcut keys

A shortcut key is a way of activating a Button by means of the keyboard instead of the mouse.

To create shortcut keys for Buttons put an '&' symbol into the caption of your Button. (The '&' symbol will not be displayed. The letter that comes *after* the '&' symbol will be underlined.)

When the user presses <Alt> and the underlined letter, that Button option will be carried out.

NB: Always make sure that no two items have the same shortcut key!

Regular feedback

It often helps if you can display the current status of your program. The following techniques can be used:

- It is far less frustrating if the word "Calculating'" flashes on a status bar when long calculations are being performed, instead of the program just 'sitting' apparently doing nothing.
- Another good method of providing an indicator is to provide a progress bar, if a process is taking a long time. Delphi has a StatusBar and a ProgressBar component on the Win32 page.
- The other cool thing to do is to make use of a technique that many Windows programs use – altering the shape of the Cursor to provide feedback to the user that processing is taking place after the user has clicked on a component. Often, for example, the cursor will change to the shape of an Hour Glass to indicate that the system is busy. This can be achieved by simply altering the Cursor property of the appropriate component. The code below shows how to store the current cursor shape of a form, how to alter it to an hour glass shape and then to restore the original cursor type.

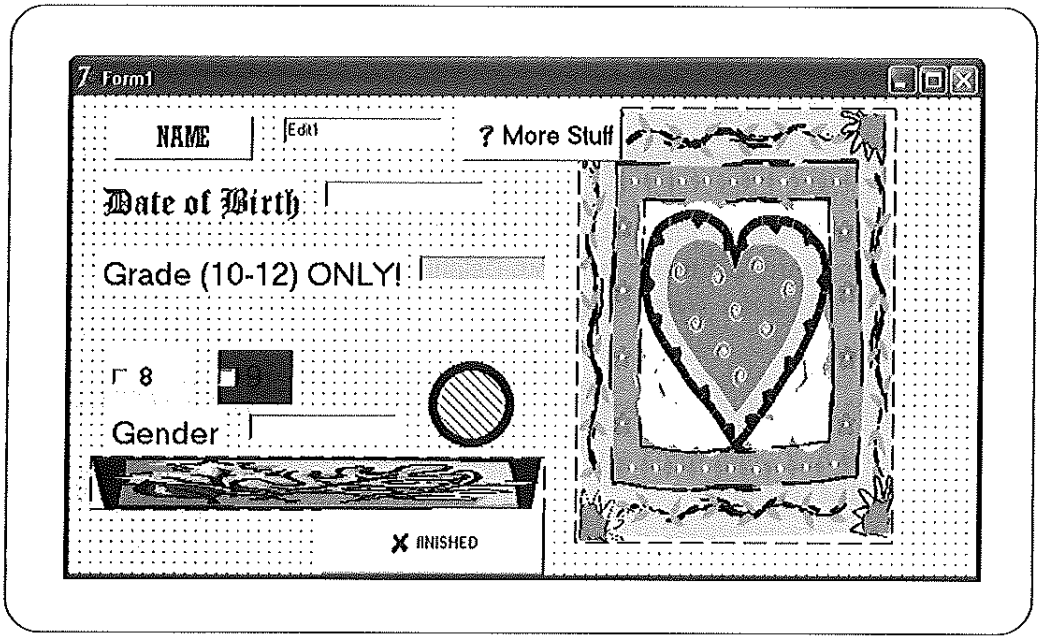
```
procedure TfrmExample.btnProcessClick(Sender: TObject);
Var CursorType : TCursor ; // variable to store a cursor type
begin
   CursorType := frmExample.Cursor; //store cursor shape of form
   btnProcess.Cursor := crHourGlass; //change cursor to hour glass
  // some code that may take a while to execute!
   Sleep(3000);  // to simulate the 'delay'
   btnProcess.Cursor := CursorType; // restore cursor type
end;
```

Checkpoint: Try the following yourself

We want to design the interface for a program that will allow a user to enter the name as well as other details of a learner. The following information concerning the learner needs to be input:

- Name and surname
- Date of birth
- Gender
- Grade (8-12)

The following interface has been designed by one of your friends. Critically evaluate the interface and make suggestions on how it can be improved. Design a better one yourself.



Defensive programming

Errors usually occur because the user did something stupid. However, they can also happen because you as the programmer made a mess of the logic of your code or did not provide code to check for possible input errors made by the user.

We are now going to look at *defensive programming*, i.e. anticipating and dealing with errors in a way that stops your program from crashing!

A good programmer

- anticipates and plans for all the possible ways that users are going to want to interact with the computer
- anticipates and plans for all the stupid things that users are going to try to do on the computer
- tries to prevent errors occurring rather than simply trying to write code to 'recover' from them
- provides meaningful error messages if an error does occur by giving a precise description of the error and suggesting a course of action to remedy the situation. (It is not acceptable to have ambiguous or rude error messages such as "An error has occurred – you had better fix it!")

One can start by ***making sensible use of the input components*** available to you in Delphi. Examples of this include:

- Using a SpinEdit to ensure that the user enters (integer) numerical data correctly.
- Using RadioButtons and CheckBoxes to make decision making and choosing options easier.
- Using a MaskEdit to control the format of input

Using components in this way is the first line / level of defensive programming. (Sometimes it is not convenient to use a SpinEdit or a MaskEdit – or you really do need to use just a plain old Edit for input.)

Communicating with the user

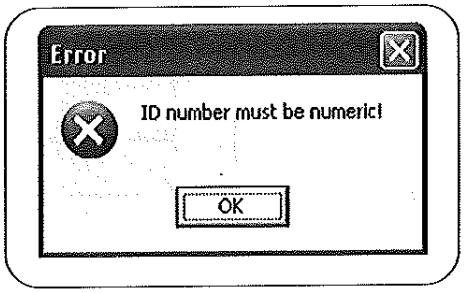
Sometimes you need to communicate something to the user – show them a message that does not fit into the design and layout of your form. Putting a label on the form will simply mess up your design – and might not be noticed by the user when you need them to notice it! The solution is to use a function built into Delphi that pops up a dialogue box to display a message to the user. This is the ***MessageDlg.***

An example is:

```
MessageDlg('ID number must be numeric!', mtError, [mbOk], 0);
```

There are four parameters separated by commas and they do the following:

- The *first parameter* is the text that will be displayed. The dialogue automatically adjusts its width and height to suit the length of the message that is being displayed.



The text can be displayed on two (or more) lines by adding #13 to the text e.g.

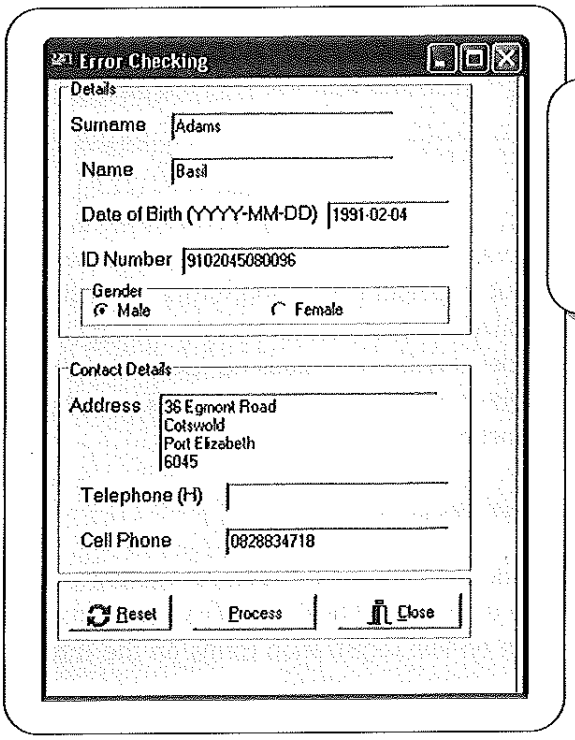
```
MessageDlg('ID number' + #13 + 'must be numeric!', mtError, [mbOk], 0);
```

There is no limit to the number of items that you can add together in this way to create the message that must be displayed.

- The second parameter affects the title and graphic displayed in the dialogue box.
  - **mtError** makes the title 'Error' and displays a big red circle with a white cross.
  - **mtInformation** makes the title 'Information' and displays a white dialogue balloon with a blue letter displayed inside it.
  - **mtConfirmation** makes the title 'Confirmation' and displays a white dialogue balloon with a blue ? displayed inside it.
  - **mtWarning** makes the title 'Warning' and displays a yellow triangle with a white exclamation mark.
  - **mtCustom** makes the title the application's name and displays no graphic.
- The *third parameter* determines which Buttons will be displayed in the dialogue box. Examples include [mbYes], [mbNo], [mbOK], [mbRetry], etc.
- The *last parameter* is the help context number value. You are not working with help and help contexts and so you should just leave this value as a 0.

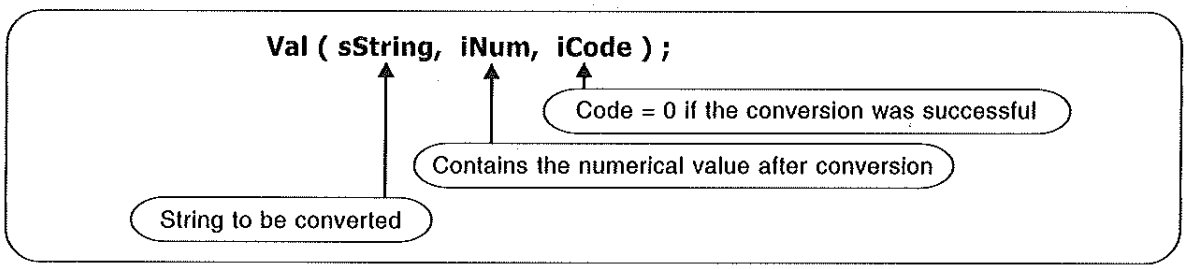
Checking that the right type of data has been entered

We are now moving to the second level of defensive programming, namely the use of error checking techniques to check that at least the right type of data has been input.



Let's have a look at some of the ways that we can carry out error checking / data validation in our programs. We will use a common example of a 'registration' type of form that a user has to fill in.

The **VAL procedure** has the following format:



The variable V can be an integer-type or real-type variable. If V is an integer-type variable, S must form a valid whole number when converted. If V is a real variable, S must form a valid real number when converted.

We can apply this to check whether the value entered in the edtIDNum field is numerical, as follows:

```
Val(edtIDNum.Text, rIDNum, iCode );
if iCode <> 0
   then
      begin
         MessageDlg('ID number must be numeric!', mtError, [mbOk], 0);
         edtIDNum.Color := clYellow; // to indicate a problem with the field
         edtIDNum.Clear;
         edtIDNum.SetFocus;  // clear edit and reset focus on edit
      end
   else
      begin
       // code if conversion was successful
      end;
```

Example

Very often a user is required to enter a numerical value such as the ID number in the example or a person's age. Because almost everything we input in Delphi is in string format (e.g. the Text property in an Edit), inputs must sometimes be converted to a numerical value. Normally we would use the *StrToInt* function for this purpose, but this function does not perform any error checking. We can test if the string, when converted to a number, will form a valid number by using the *Val* procedure instead of the *StrToInt* function.

Checking that the user has actually entered data

It can happen that a user leaves a field blank during the capturing of data. Sometimes you may want to permit this. At other times, you may want the user to have to fill in some fields, which we call mandatory fields. You will often see this on on-line registration forms on the Internet, for example, where some fields must be filled in while others are optional.

Example 1

We can safely assume that a surname, for example, must have at least two characters. We could add an OnExit event handler to this Edit and add the following code:

```
if Length(edtSurname.Text) < 2 then
  begin
     Beep;
     edtSurname.Clear;
     edtSurname.SetFocus;
     edtSurname.Color := clYellow;
     MessageDlg('Surname must be at least 2 characters long', mtError,
                                                          [mbOk], 0);
  end;
```

Example 2

In a RadioGroup the ItemIndex property has a value of –1 if no Button (option) has been selected. We can therefore trap this error with the following code:

```
if rgpGenderItemIndex = -1
  then MessageDlg('Please select a gender ', mtError, [mbOk], 0)
  else
     begin
        // normal code
     end;
```

Checking that the data is valid

Data is valid if it meets some specific criteria.

Natural limits on the data

Example 1

Sometimes there are some forms of natural limits on the data, such as when the user enters a value representing a month. The value input must be an integer in the range [1..12]. This value can easily be checked with the following code:

```
if Not (iMonth in [1..12])
  then
     begin
           // display an error message
     end
  else
     begin
           // code if input is valid
     end;
```

Example 2

Sometimes the check is not easy as the range can be quite extensive. Take, for example, the case of the ID number in our example. The only real restriction is that it must consist of

13 digits. This can easily be checked with code of the form:

```
if Length (edtIDNum.Text) <> 13
   then .. // display error message
```

Logically correct

The data not only needs to be in a valid range, but also needs to be "logically correct".

Example 1

The data input might be valid in terms of its format, type and range, but that does not mean it is "correct"! A programmer must try to write a program so that it rejects input, which we as "humans" can see straight away does not make sense!

For example, if a learner is in grade 12, then his year of birth should fall in a "reasonable range". It is probably not likely that the learner would be 5 years old in Matric! Sometimes these intuitive rules are the ones we forget to program into our systems.

Example 2

Checking that data is 'logically' correct also includes what we sometimes call *combination checks.* Sometimes the data being entered might be valid in type, range and format but does not make sense when looking at all the inputs together as a combination.

For example, if a user has to enter the codes for his six subjects, he cannot enter any code more than once. Each code being entered can be valid on its own, but each code must be unique.

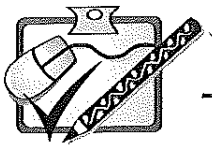
Similarly, in our example, the first six digits of the ID number entered should correlate with the date of birth entered. In addition the digits 7-10 of an ID number form a four-digit number which indicates the gender of the person. A number of 5000..9999 indicates a 'male' and any number below 5000 indicates a 'female'. Therefore the gender selected via the RadioGroup should 'tally' with the ID number input.

Remember that computers do not have any intelligence and can therefore not detect that invalid data has been typed in. Computers only do what we as programmers tell them to do! Simply put, if we allow the user to enter 'rubbish' or invalid data, we can expect our programs to produce 'weird' output. We sometimes refer to this principle as GIGO (Garbage In -> Garbage Out).

Remember, you can't make assumptions that:

- data has been entered
- the correct type, format and range of data has been entered
- the data is 'logically' correct
- critical steps have been followed

You need STRONG (robust) error handling. (Remember – the USER is in control and we all know how stupid the USER can be!)



Test, Improve, Apply

1. Design a similar interface to the one shown and apply the techniques discussed in the chapter.

   The data that is supplied by the user must adhere to the following conditions.

   For the answer, only a year between 1900 and 2000 should be accepted. The program must also validate that the year entered only consist of numbers.

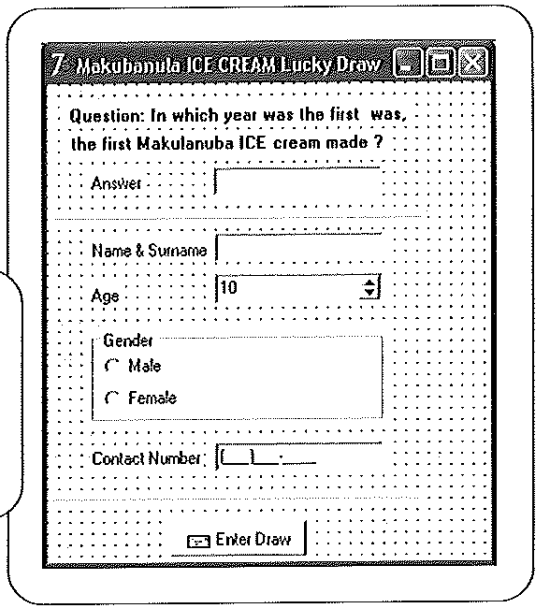
   The name and surname must at least be four characters long. A name and a surname must be supplied – they cannot be left out.

   The default age for the SpinEdit box must be set to 10.

   The user must supply a gender – it cannot be left out.

   Look at all the *possible* errors we could trap for. Note that, especially while we are still learning to use Delphi, we will not necessarily implement full error handling in every single program.

   Most of the components used for data input have an OnExit event that can be used for data validation. Good feedback should be provided by the application if the data input is invalid or inappropriate in some way. A user-friendly message should be displayed in a Message Dialog that clearly indicates what the error is.

   Remark

   Setting up a MaskEdit with the correct EditMask will ensure that only valid input may be supplied.

2. Silver Star Airlines have just purchased the latest Boeing-777 for their fleet. They wish to run an on-line competition for the public to submit a proposed name for the plane. The winner will receive two tickets aboard the first flight overseas! Design an interface like the one shown and add the error checking as directed.

   Note that the Event handler for the [Submit] Button is not used at this stage, apart from conducting some error checking. The emphasis is on getting some practice on performing good error checking and data validation! Implement the error checking for each component by using the OnExit Event handler of the component. Suitable error messages must be supplied where necessary.

   Choice of Name

   This is an Edit and the choice of name must be between 5 and 12 letters.

   Set the *MaxLength* and *CharCase* properties in the OI to ensure that the user types in his choice in uppercase and restrict him to a maximum of 12 characters.

   User's Name

   Check that this Edit is not left blank.

   ID Number

   Use a MaskEdit to ensure that the user types in 13 digits.

Age

Use appropriate values for the MinValue and MaxValue properties. Check if the age tallies with the year of birth in the ID number.

Cell Number

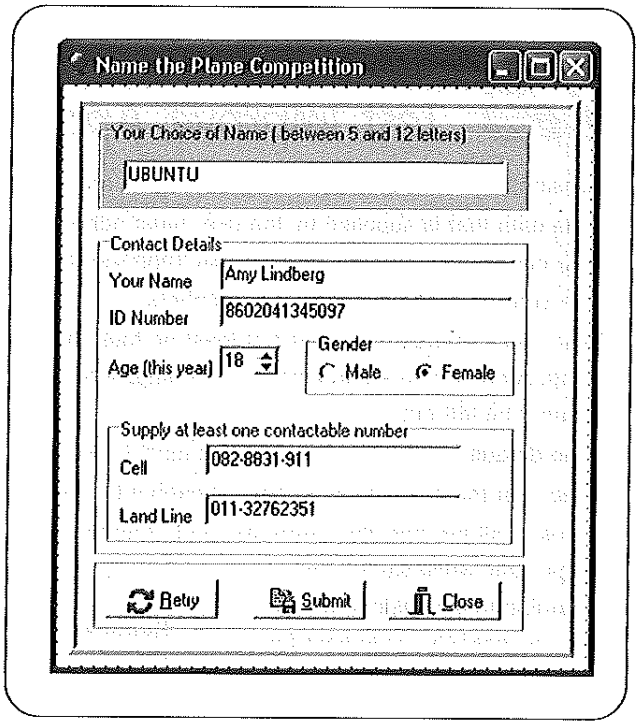
Use a MaskEdit to ensure that the user enters a number in the format: 0nn-nnnn-nnn where each 'n' is a valid digit.

Land Line

Use a MaskEdit to ensue that the user enters a number in the format: 0nn-nnnnnnn where each 'n' is a valid digit.

[Submit] Button

When the user clicks this Button, the program must check that at least one of the contact numbers has been entered.



In addition, the digits 7-10 of an ID number form a four-digit number, which indicates the gender of the person. A number of 5000..9999 indicates a 'male' and any number below 5000 indicates a 'female'. Therefore the gender selected via the RadioGroup should 'tally' with the ID number input.

If all the input is valid and no errors have been detected, then the program should display a message that the person's name, contact details and choice of name for the plane have been successfully entered for the competition.

3. A company allows the users to create their own passwords for logging on to the computers. They do, however, specify rules that have to be followed when the password is created. These rules are as follows:
   - must be between 8 and 10 characters long
   - the first character must be a letter
   - the second character should be one of the following: • ? /
   - the whole password cannot contain more than 4 letters – the rest should be numbers or punctuation.

   Write a program that allows the user to enter a password and then tells them if the password is valid. It should say exactly which rules are broken if the password is not valid.

4. The MarkWorthy College contracts outside personnel for the marking of students' exam scripts.
   A unique employee number is generated for each marker. (The employee number consists only of numerical digits)
   The last digit of the number is used as a check digit to validate the employee number as a whole.
   The last digit is the remainder after the sum of all the previous digits of the number have been added and divided by 10.E.g.

   Employee Number: 123455 would be valid because:

   1 + 2 + 3 + 4 + 5 = **15**

   15 / 10 = 1 remainder **5**

Employee Number: 2365872 would be invalid because

2 + 3 + 6 + 5 + 8 + 7 = 31

31 / 10 = 3 remainder 1 (not 2)

Employee Number: 22228 would be valid.

Develop an interface similar to the one shown. The program must work as follows:

The user must type in the employee number but will only be allowed to continue if the employee number entered is valid.

The college pays:

R12.15c per first year level script marked.

R22.50c per second year level script marked.

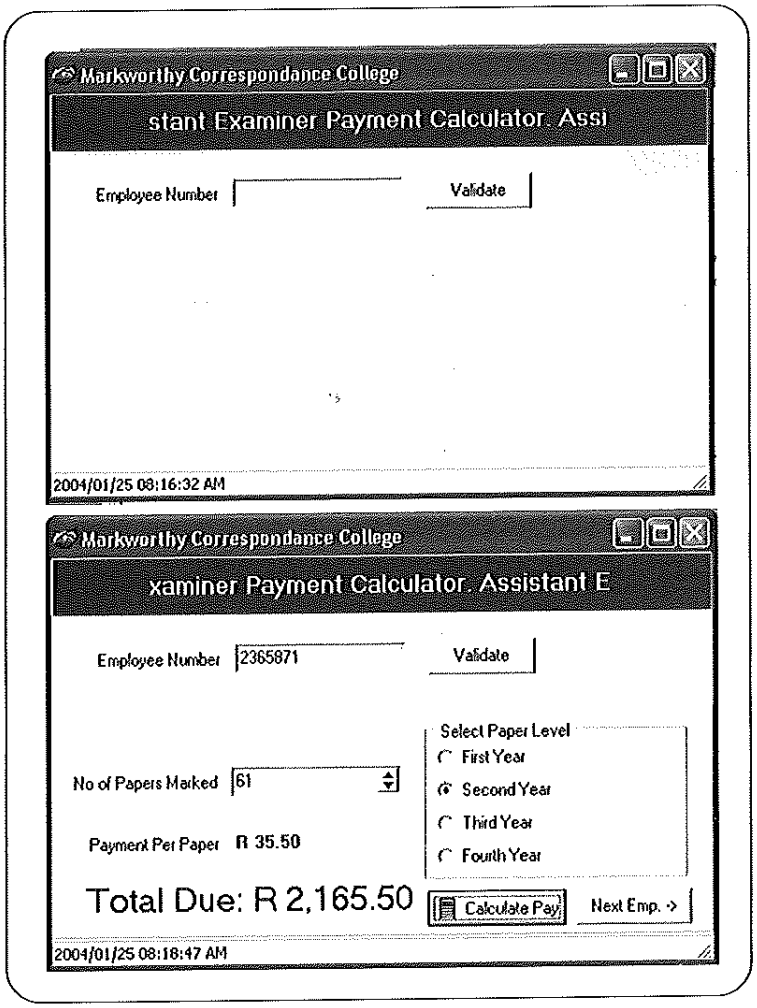
R35.75c per third year level script marked.

R45.50c per fourth year level script marked.

After the user has supplied the number of scripts marked, and indicated the level of scripts, the total amount due must be displayed.

Extra – CHALLENGE – Not Compulsory

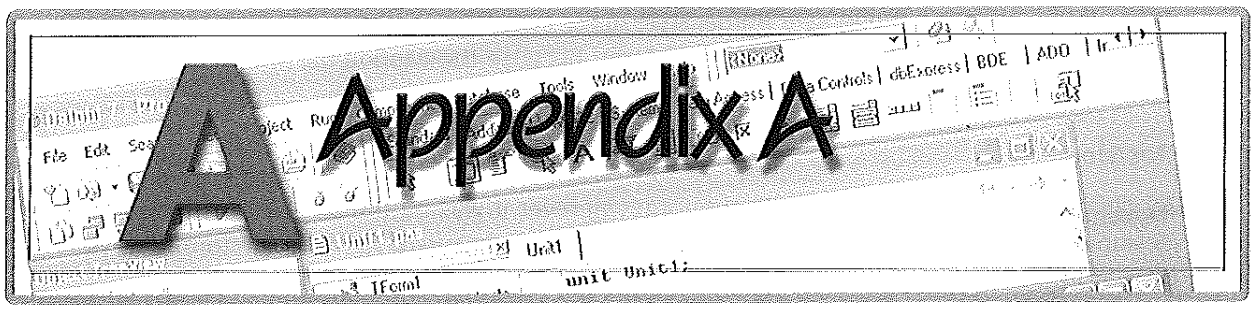
- The Panel Heading (Caption) scrolls from left to right on the top Panel.
- A StatusBar displays the current date and time.
- Check that the employee number consists only of digits and if that is not the case, display an error message and allow the user to re-enter the employee number.
- Confirm the user's intention to exit the program.



Test yourself

5. **Check that you know / can do the following:**

		✓
Knowledge	**I can explain in my own words**	
	briefly describe what contributes to a well-designed friendly user-interface	
	list techniques that can be used to create a user-friendly interface	
	explain the concept of defensive programming and why it is necessary	
	list some techniques and guidelines to make a programs more defensive	
	explain what is meant by the GIGO principle	
Skills	**I can**	
	provide the user with clear instructions using Labels, Icons, Images and Hints	
	ensure that data is input in a logical order by using the Tab order	
	set the focus to the component which the user is most likely to initially use	
	use short-cut keys for Buttons	
	design a user-friendly interface for my programs	
	make use of the MessageDlg procedure	
	test if a value input forms a valid number	
	determine whether a user has actually typed in data	
	do simple validity tests on data	



The GUI (Advanced)

There are many small things that you can do to make the user interface of your program more effective and professional looking. This appendix is going to be more of a ***reference*** than the other chapters in this book. It is not going to provide you with exercises and review points. It will however show you step-by-step how to achieve certain effects in the user interface of your program.

The user interface of your program is determined by what components you use and how you use them. Often you are too busy (or too scared) to play around with all the properties of a component to find out what the component is capable of doing. Yet it is these *little used properties* of components that can have the greatest effect on the way that your user interface looks and feels!

This appendix is going to deal with making your programs

- look slick
- professional
- user-friendly
- like (and work like) all other Windows applications
- easy to use and navigate.

This is *not* just about making things pretty.

It's about making design and interface choices that are best suited to your program and the way that the program is meant to work.

The Form

The form is a component just like all the other components we use in Delphi. It is the display surface that your program uses to communicate with the user, yet we never spend time to get to know what the form is capable of. Let's look at some of the properties of the form.

A quick look at the Object Inspector will show that the form has many properties. It is a good idea to know what these properties do!

Border

Changing the Border property has no effect whilst you are designing your program. It does affect the way the form looks and works when you run your program.

Note: If you choose bsNone as a BorderStyle make sure that you have a Button for (or some other way of) closing your program on the form, because the caption bar with its 'x' icon will not be there! Users will also *not* be able to move, maximise or minimise the form.

Position

By default your program starts with the form wherever it was on the screen when you were designing the program. Change the Position to ***poScreenCentre*** to make the form always appear in the middle of the screen.

FormStyle

Use the ***fsStayOnTop*** option here to make sure that no other windows will ever cover your program. Only do this if you think it is really necessary!

WindowState

This option allows you to make your program minimise or maximise itself when it starts up. Choose the ***wsMaximized*** option to always have a maximised window.

Font

Changing this property affects all the components that you place on the form. Rather than changing the font of all the Labels and Edits that you place on a form one-by-one, simply change the *form's* font. This can save you a lot of time and effort. The form and all the components placed on the form will inherit and use that font (unless you want to specifically change the font of one of the components).

Width and Height

The width and height of the form do not have to stay the same throughout your program. You can change them whilst your program is running. You can design a form to have an *area for input* on the left and an *area for display* on the right. Before you compile the program you resize the form so that only the area on the left is visible (the other components are there, they just can't be seen). Then you have a [Display] or [Process] Button – or something similar. All it needs to do (besides process the data) is to make the form's Width property as wide as it needs to be to display the hidden right hand side of the display.

This technique works best with a form BorderStyle of bsDialog, bsSingle or bsNone (so that the user can't resize the form themselves).

KeyPreview

We'll talk more about KeyPreview in the section on Keyboard-friendly programming. Right now, all you need to know is that setting this property to *true* means that the form gets to handle all keyboard input *before* the active component does (e.g. the form gets the input before the Edit that the user is typing into).

Using Tabs in a RichEdit

More about tabs

If you want to display a number of lines containing a combinations of text and values and they should line up in columns properly, or you do not want to make use of Delphi's default tab positions, you are going to have to set the tab positions. This is done using code like the example below:

```
RichEdit1.Paragraph.TabCount := 3; //Show how many tabs you are going to
set
RichEdit1.Paragraph.Tab[0] := 100; //The first tab is set to position 100
RichEdit1.Paragraph.Tab[1] := 200; //The second tab is set to position 200
RichEdit1.Paragraph.Tab[2] := 300; //The third tab is set to position 300
RichEdit1.Lines.Add(#9 + IntToStr(Random (99)) + #9 + IntToStr(Random(99))
                                    + #9 + IntToStr(Random (99)));
```

- Each tab stop has a number – the number in the square brackets shows which tab stop position you are setting. These numbers start at 0 and are numbered sequentially to the number of tabs you have set - 1 (2 in the example above).
- The stop positions you specify (e.g. 100, 200 of 300) show the tab stop in ***points*** (a unit of measurement used in the publishing industry) measured from the left hand margin of the RichEdit.

Aligning decimal points

You have already used the FloatToStrF function. It simply converts a real number into a formatted string. By using this function you can, for example, specify the number of decimal points that will be shown.

Sometimes you want to show *all the decimal points aligned* so that they are shown beneath each other, as in the example below:

Non aligned decimal points	Aligned decimal points
74.67	74.67
445.12	445.12
0.34	0.34

You could take the output of the FloatToStrF function and manipulate it to add the extra spaces to make the text line up with the decimals below each other – but this is the complicated way of doing things.

An alternative would be to use another function called the **Format function**.

This is how it works: **Format ('R %8.2f', [rPrice])**

The *first parameter* is a string which holds instructions and placeholders for formatting.

The control codes used in the function are the following:

%: This indicates that the text which follows is formatting instructions and not normal text.

8: Formatting code showing that the inserted text must be 8 characters long

2: Formatting code showing that the inserted text must have 2 decimal points

f: Shows that the value being passed in the second parameter is a floating point value.

The *second parameter* holds the value that needs to be converted into a string. In our example it is the name of a variable rPrice and it must be placed inside square brackets. It can contain the decimal number itself: Format ('R %8.2f', [125.6777])

If the variable rPrice in our example contains the value 8.95 then the result of this will be **R ____8.95**

The underlined section indicates 4 spaces. 8.95 is 4 characters long, the format function needs to produce a string 8 characters long so it adds 4 spaces before the 8.95.

If we are going to try to line the text up in a RichEdit then there is one more thing we need to remember: Windows uses 2 types of fonts, namely Proportional fonts and Fixed fonts. Proportional fonts have different size spaces between the characters according to the shape of the characters. For our type of formatting we have to use the fixed fonts – fonts like Courier New or Lucida.

Example

Take a look at the code below to see an example of how to line up numbers underneath each other on the decimal point.

In the *FormActivate Event handler*:

```
RichEdit1.Paragraph.TabCount := 1 ; // One tab only
RichEdit1.Paragraph.Tab[0] := 100 ; // Set it to position 100
```

Set the *Font* property in the OI to Lucida or Courier New OR add

```
RichEdit1.SelAttributes.Name := 'Lucida Console';
```

to your code BEFORE you call the Lines.Add method.

The code to add a line of text to the RichEdit is:

```
RichEdit1.Lines.Add('Average' + #9 + Format('%7.2f',[rAverage])) ;
```

The output in the RichEdit should look as follows: (The grid lines will not appear they are just shown here to clarify.)

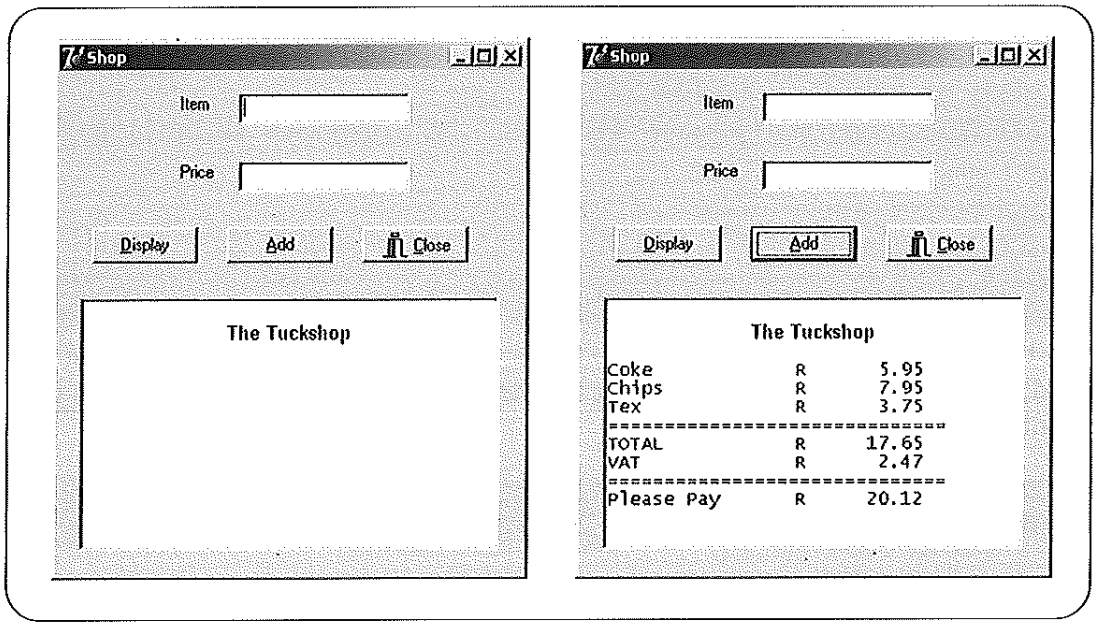
							Tab –is not shown	7 spaces used for the string						
A	v	E	r	a	g	e	→			8	8	.	4	5
A	v	E	r	a	g	e	→				9	.	5	0
A	v	E	r	a	g	e	→		1	0	0	.	0	0

Remark:

If you change the 'f' format option to an 'm', then you are telling the Format command that it is not formatting as floating point but as **M**oney. The result will be a decimal number with two digits after the decimal point and an additional symbol to show currency. The symbol shown is determined by the Windows regional settings.

Example of a program displaying decimal values

Design the following user interface:



When the form is activated, the name of the shop must be displayed in a RichEdit.

The following is an example of the code you need to enter in the FormActivate Event handler:

```
procedure TfrmShop.FormActivate(Sender: TObject);
Const
  ShopName = 'The Tuckshop';
begin
  rTotal := 0;
  redOutput.Paragraph.TabCount := 2 ;
  redOutput.Paragraph.Tab[0] := 100;
  redOutput.Paragraph.Tab[1] := 120;
  redOutput.SelAttributes.Size := 10;
  redOutput.SelAttributes.Style := [fsBold];
  redOutput.SelAttributes.Name := 'Arial';
  redOutput.Paragraph.Alignment := taCenter;
  redOutput.Lines.Add(ShopName);
  redOutput.Lines.Add('');
  edtItem.SetFocus;
end;
```

Remember:
Since rTotal is used in more than one procedure, it needs to be declared as a global variable.

When somebody buys items at the shop, an item and its price must be entered. The [Display] Button must must be clicked to display the item and its price.

```
procedure TfrmShop.btnDisplayClick(Sender: TObject);
var
  sItem : string;
  rPrice : real;
begin
  redOutput.Paragraph.Alignment := taLeftJustify;
  sItem := edtItem.Text;
  rPrice := StrToFloat(edtPrice.Text);
  rTotal := rTotal + rPrice;
  redOutput.Lines.Add(sItem + #9 + 'R' + #9 + Format('%8.2f',[rPrice]));
  edtItem.Clear;
  edtPrice.Clear;
  edtItem.SetFocus;
end;
```

When the all the items the person wants to buy is entered, the [Calculate] Button must be clicked. The total amount and the Vat must also be calculated and displayed as well as the final amount the person needs to pay.

```
procedure TfrmShop.btnCalculateClick(Sender: TObject);
const
  VAT = 0.14;
var
  VatPayable, Pay : Real;
begin
  VatPayable := rTotal * VAT;
  Pay := rTotal + VatPayable;
   redOutput.Lines.Add('============================');
  redOutput.Lines.Add('TOTAL' + #9 + 'R' + #9 +
                                  Format('%8.2f',[rTotal]));
  redOutput.Lines.Add('VAT' + #9 + 'R' + #9 +
                                  Format('%8.2f',[VatPayable]));
   redOutput.Lines.Add('============================');
  redOutput.Lines.Add('Please Pay' + #9 + 'R' + #9 +
                                  Format('%8.2f',[Pay]));
end;
```

The Edit

The PasswordChar property

The PasswordChar property controls whether the Edit acts as a 'password' input that hides what is typed into it.

- Change the ***PasswordChar*** property of the Edit to *.
  - Now the * symbol will be displayed no matter what you type into the Edit.
  - The Text property will contain what you have typed, but the Edit will only display the * character.
- To make the Edit display the text typed into it, simply change the ***PasswordChar*** property back to '#0'

You can make any character be the PasswordChar. If you change this property to 'Z' then only Z will be displayed by the Edit.

Use this to hide password inputs etc.

The OnEnter and OnExit events of the Edit

These happen when the user enters or leaves the Edit (i.e. they click on / press the <Tab> key to get to / get away from the Edit). We can use these events to let the Edit highlight itself when it is 'active'.

Write the following code

- In the OnEnter event of an Edit type
  edName.color := clcyan;
- In the OnExit event of an Edit type
  edName.color := clwhite;

You can also type in code here to check if the user has entered valid data in the Edit.

When you run your program you will see that if you click on / tab to the Edit it will change colour to cyan and when you leave the Edit it will change back to white.

Keyboard-friendly Programming

Keyboard-friendly programming means that the user can use your program even if the computer doesn't have a mouse. Every Button, Edit or command can be activated using the keyboard.

By changing the focus, using a good tab order sequence and creating shortcut keys for menu options and Buttons, we can already make our programs more keyboard-friendly. We can, however also make Edits or the form *respond to the keyboard.*

- This is done by writing code for the OnKeyUp event of the component.
- The event includes a parameter called Key that holds the code of the key that was pressed.

- Delphi has constants to represent special keys such as <Enter>, <BackSpace>, <PgDn> or <PgUp>, etc.
- Checking if the Key parameter matches one of these constants allows you to make your program respond to the keyboard.

Making an Edit 'click' a Button when the <Enter> key is pressed

In the OnKeyUp event of the Edit write the following code:

```
If Key = vk_Return then btnProcessClick (self);
```

To make this work in your program simply change the name of the Button Event handler that you are calling. This is the name of the Button followed by 'Click' - with no spaces! (e.g. DisplayClick).

Making the program act as if the Button is clicked no matter which Edit we are on

Most often we want the program to act as if the Button is clicked no matter which Edit we are on. There a quite a few ways of making this happen. They are listed below:

Method 1

Write an OnKeyUp event for each Edit on the form (not a good option; too clumsy, wastes time).

Method 2

Write an OnKeyUp event for one Edit and then point all the other Edit's Event handlers to that one.

- To do this you follow the steps described in the example for one of the Edits.
- Then you select all the other Edits.
- With the Edits selected click on the Events tab of the Object Inspector.
- Next to the OnKeyUp event you will be able to click on a down arrow which will produce a list of events.
- Choose the event that you created for the Edits onKeyUp.
- Now all the Edits will go to the same code!

Method 3

You could also write the code once in the Form's OnKeyUp event – and then change the KeyPreview property to true.

- This means that the form gets to see and process the key before it even gets to the Edit.
- Writing the code here is a good option if pressing <Enter> is only meant to activate one Button or menu instruction.

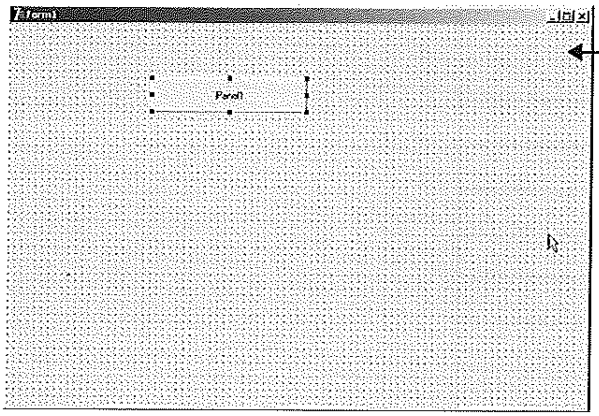
- If you need to have 'intelligent' Edits that activate the correct one of many Buttons then you should use one of the other techniques described above.

Delphi Keycode Constants	
Keycode	**Actual Key**
Vk_Return	Enter
Vk_Prior	PgUp
Vk_Next	PgDn
Vk_F1 ... Vk_F12	F1-key .. F12 key
Vk_Lbutton	Left mouse
Vk_Rbutton	Right mouse
Vk_Back	Backspace
Vk_Tab	Tab
Vk_Shift	Shift
Vk_Control	Ctrl
Vk_Menu	Alt
Vk_Pause	Pause
Vk_Capital	Caps Lock
Vk_Escape	Esc
Vk_Space	Space bar
Vk_End	End
Vk_Home	Home
Vk_Left	Left Arrow
Vk_Up	Up Arrow
Vk_Right	Right Arrow
Vk_Down	Down Arrow
Vk_Snapshot	Print Screen
Vk_Insert	Insert
Vk_Delete	Delete
Vk_Numlock	Num Lock

Being able to use these keycodes to respond to the keyboard makes your program feel richer and more professional!

The Panel

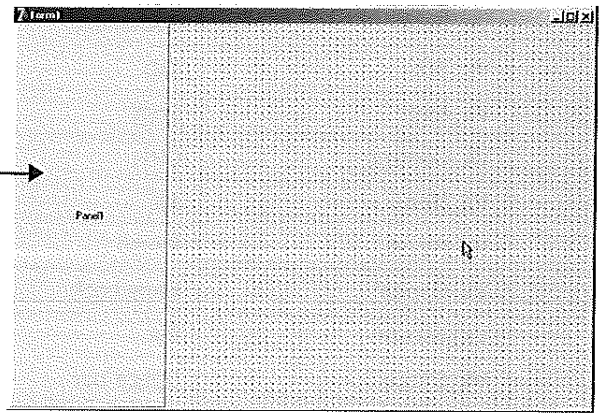
The Panel is mainly useful in structuring the layout of your form. We do this by manipulating the Align property of the Panel.

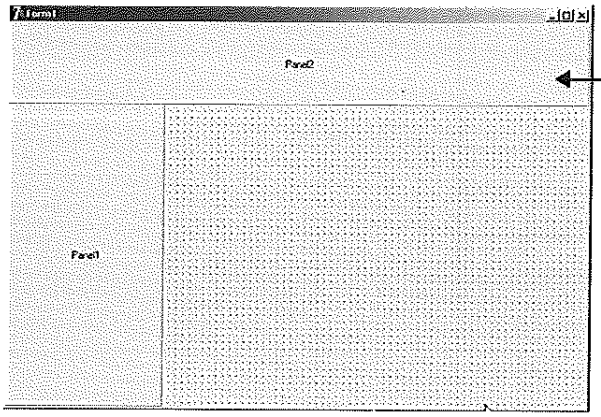


This Panel has simply been placed on the form. If the form is resized it will stay where it has been placed and its size will not change.

If you check the ***Align*** property you will see that it is set to ***alNone***.

This Panel has had its ***Align*** property set to ***alLeft***. This means that when you resize the form it will grow / shrink in height to match the size of the form but will always keep its width.



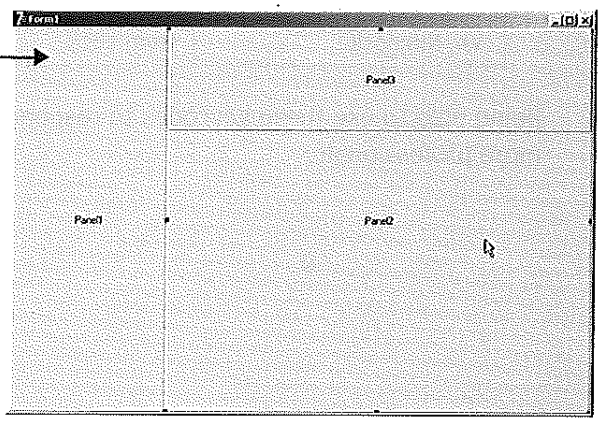


A second Panel has been added – its ***Align*** property has been set to ***alTop***. It will always have the same height, but when the form is resized its width will change.

Notice that top & bottom alignment take priority over left and right alignment.

In this example Panel2's ***Align*** property first has been changed to ***alClient*** meaning that it will always take up whatever part of the form Panel1 does not use.

To show that the Panel's ***Align*** property is relative to its parent we then placed another Panel on top of Panel2. When we change its ***Align*** property to ***alTop*** it places itself across the top of Panel2, not the top of the form!



By using the ***Align*** property of Panels you are able to create a form with 'sections' that resize themselves when the user changes the size of the form.

This makes your programs look much slicker and more professional – with very little work on your behalf!

Note:

- Panels are not the only components with an Align property. What you have learnt here applies to the alignment of all components!
- Panels become the parent of the components placed on them. This means
  - Changing the font of the Panel changes the fonts of the components on the Panel
  - Setting the Panel's Enabled property to false means that any Buttons / Edits /etc on that Panel will not be active or accessible.
  - Setting a Panel's Visible property to false not only hides the Panel but all the components on the Panel as well!

You can use all of these facts creatively to manage your user interface and make your program seem to do amazing things!

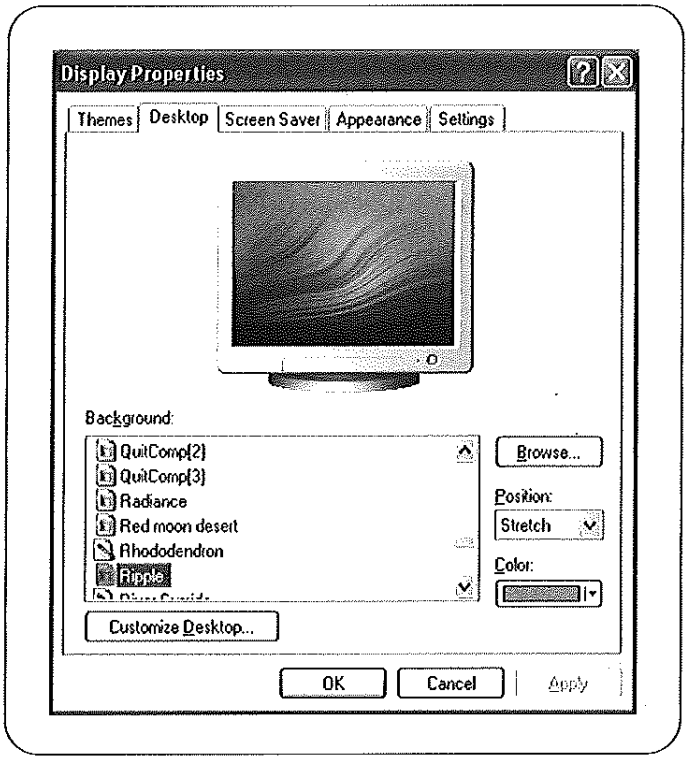
The PageControl

This component is found on the ***Win 32*** tab of the component palette.

A PageControl is like having a multi-page notebook on your form. Each page can be displayed one at a time (it is ***not*** possible to see all the pages at once, just like it is not possible to see all the pages of a notebook at once).

By giving you many different pages to work with, the PageControl allows you to put many different 'layouts' on your form representing the different pages. Only one 'layout' will be visible to the user at any time. Each 'layout' or 'page' has a 'tab' attached to it. When you click on the tab, that specific page becomes the one that is displayed.

You have seen page controls many times – like when you change windows settings such as the display properties / screen saver, etc.



The Delphi component palette is also created using a page control.

The best way to learn is to do, so.....

How to use a PageControl

- Place one PageControl on the form.
  It should look just like a Panel – and you are probably thinking to yourself that it is no big deal as components go....
- Now
  - Right-click on the PageControl.
  - Select ***New Page*** from the menu that pops up.
  - Suddenly you will see a tab appear on the PageControl.
  - Right-click the PageControl again (up where the tab is).
  - Select ***New Page*** again and now you PageControl will have 2 pages to it.

  The great thing is that each page (called a ***TabSheet***) can contain its own components.
- Go ahead
  - Click on the TabSheet called ***TabSheet1*** and then place a Label and a Button on the TabSheet
  - Now click on the TabSheet called ***TabSheet2*** and you should see a blank page. Place an Edit and a BitBtn on this page.
- Run the program and click on the tabs to see that you have now created different pages that can be displayed to the user at different times.

This is a great concept – instead of having the different interface parts of your program appear on different forms that you have to show and hide, you simply *create as many pages as you need* and put the different parts of the interface on each page.

- Make sure that the TabSheets have captions that describe their content accurately!

Interface with no tabs on it

What if you don't want the interface to show the tabs?

- Use the example above – i.e. the PageControl we created that already has 2 pages with components on each page. Make sure that the pages look different from each other.
- Click on each TabSheet in turn (click on the tab and then on the TabSheet itself) and change its TabVisible property to false.
- Put a Panel on the ***form*** – not the TabSheet.

- Set the Panel's ***Align*** property to alLeft.
- Set the PageControl's ***Align*** property to alClient.
- Put two Buttons on the Panel.
- Change their captions to read ***Page 1*** and ***Page 2***.
- For the Button with the ***Page 1*** caption create an *OnClick* event and then type the following code into that event:

```
PageControl1.ActivePage := TabSheet1;
```

- For the Button with the ***Page 2*** caption create an *OnClick* event and then type the following code into that event:

```
PageControl1.ActivePage := TabSheet2;
```

- Run the program and you will see that you now have an interface with 2 pages – and no tabs to mess up your screen!

The use of PageControl in programs

The following are examples of how PageControls can be used in programs:

- Create an introduction screen to your program – like a splash screen. When the user clicks on the Button to continue, the application moves to the second page of the PageControl that contains the program itself... (it is a good idea to hide the tabs of your pages).
- Use the PageControl to allow your program to contain help or instructions for your user.
- Create a 'tutorial' program that teaches the user about the hardware of a computer
  - Use a PageControl to divide the tutorial into sections – such as Input / Output etc.
  - Add a page that has quiz questions.
  - Use a Panel on the side to switch between the different tutor pages and the quiz pages.

Using a PageControl is a much neater and more efficient way of creating a good user interface than using multiple forms! Using multiple forms can be confusing to the user; forms can hide each other, the user can switch between forms without meaning to and so 'get lost', and managing which form the user is working on can be difficult. Besides this, having everything on one form appears 'neater' and looks more professional. Using and managing variables is also easier on a single form than when using multiple forms.

Next we will look at using a PageControl as the essential ingredient in creating a ***Wizard*** style interface.

Creating a 'Wizard'

What is a 'Wizard'?

It is a program that takes control and guides the user through the accomplishment of a task by

- breaking the task into steps
- presenting the steps to the user in small chunks and making sure that they complete each 'chunk' before they can move on to the next 'chunk'.

Where do you get to see wizards when using your computer?

- When you install a program.
- When you choose 'Add a new printer' from the printers control Panel applet.
- In Word / PowerPoint / Excel you use Wizards to create documents from pre-defined templates.
- When you create reports / queries / forms in Access.

  These are just some of the places were wizards are commonly encountered.

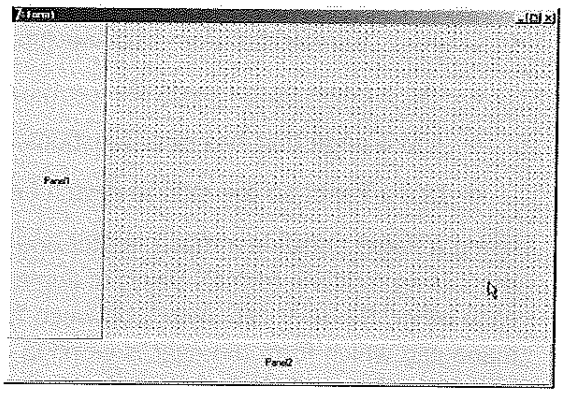
When should you use a wizard in your own programs?

- When you need to control the users actions step by step.
- When you need to make sure that a user follows a specific sequence through your program.

How to create a Wizard

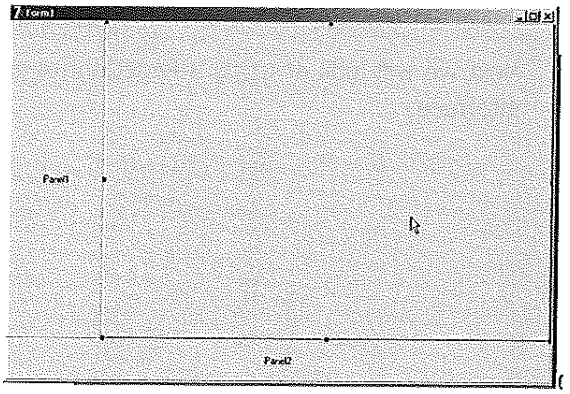
Step 1

Place two Panels on a form. Change their ***Align*** properties to ***alLeft*** and ***alBottom***. You should end up with a screen like the one on the right.



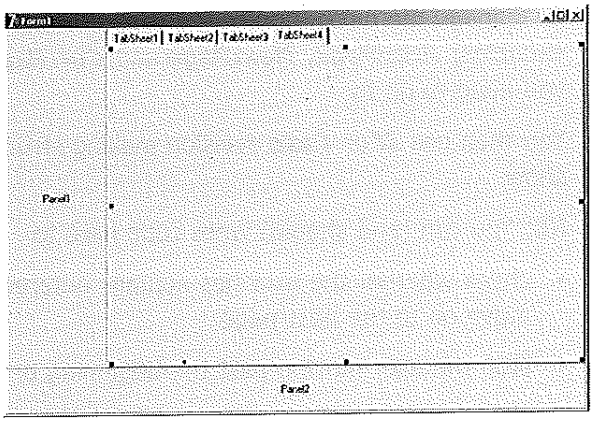
Step 2

Place a PageControl (from the Win32 tab) in the open area of the form and change its Align property to alClient. Your form should now look like the example on the right.



Step 3

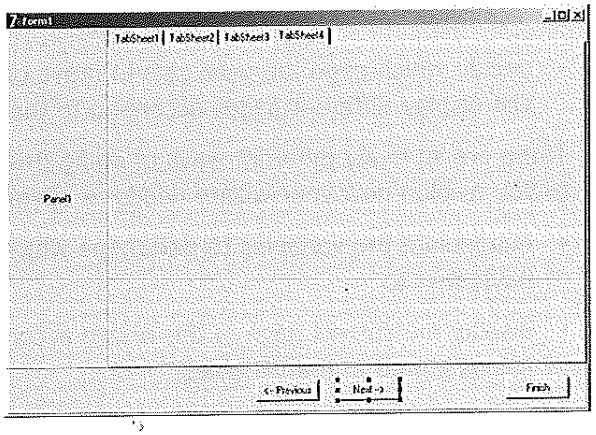
Right click on the PageControl and add the correct number of pages that you need to have in your wizard by clicking on New Page.



Step 4

Place three Buttons on the lower Panel. Change their names to btnNext, btnPrev & btnFinish. Fix their captions to display appropriately.

Make the Visible property of the Previous and Finish Buttons false (we don't want them to be displayed on the first page).



Step 5

Put the appropriate components, etc on the appropriate pages of the PageControl.

NB: Make sure that you set up the pages in the correct order, i.e. Page 1 of your Wizard should be on TabSheet1 of the PageControl.

Hide the tabs on each TabSheet by changing the ***TabVisible*** property to false.

Step 6

The last step is to write the code to manage moving between the pages of the Wizard.

There are many different ways to do this. The simplest way is to use the ***SelectNextPage*** method of the PageControl. This works as follows:

```
PageControl.SelectNextPage(True, False);
```
(for the ***Next*** Button) and

```
PageControl.SelectNextPage(False, False);
```
(for the ***Previous*** Button)

The first parameter is True to move forward and False to move backwards.

The second parameter must be set to False if you want to display pages where the tab has been hidden.

The problem with this method is that you have no real control over the operation of the wizard.

More Controlled Navigation

To manage the navigation of the wizard better it is recommended that you create 3 integer variables; ***PrevPage, NextPage*** and ***CurrPage*** to keep track of where you are and which page you need to go to.

You then use ***If*** or ***Case*** decision making structures to let the program decide which page to show.

NB: You can (and should) write code here to

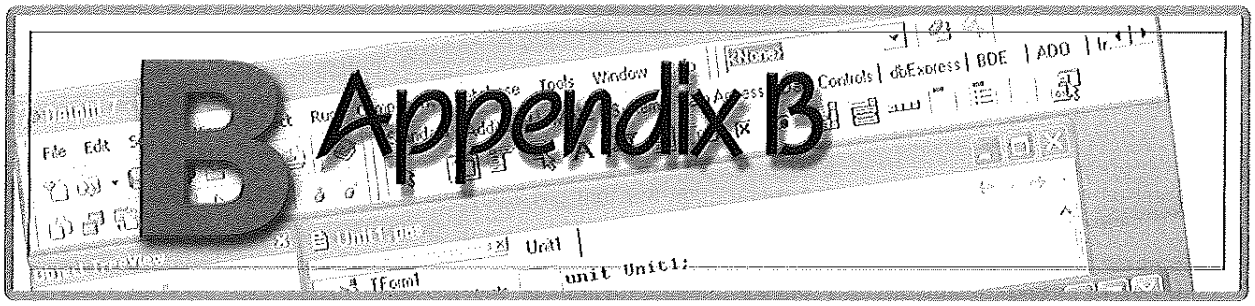
- check whether the user has actually entered the data needed before her / she is allowed to move on to the next page
- stop the user moving past the beginning or end of the wizard
- change whether Buttons are enabled / disabled / visible according to the page that is being displayed.
- decide which page the user gets to move on to. (Sometimes a complex and powerful wizard will display different pages according to the decisions that the user has made.)

There are so many different ways of doing this that we will not try to describe them all. It is up to you to plan how your wizard needs to operate and then

- manipulate the variables to determine which pages are navigated to
- display the correct page in the wizard according to the values stored in the variables.

Using a wizard style interface is an excellent way to make your program look and feel professional. It might take a while to get used to writing the necessary code, but it is really worth trying.

For a good example of a wizard style solution to a typical programming problem take a look at the source code to the *Fuel Calculator* program included on the CD.

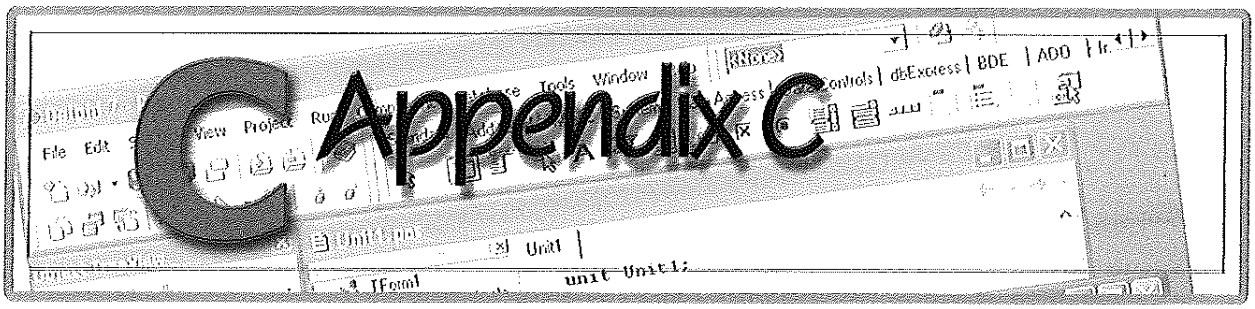


Component	Icon	Prefix	Use
Standard			
Label		lbl	Displays text such as title text.
Button		btn	Can be used to initiate an action.
Edit		edt	Displays an area where the user can enter or modify a single line of text.
RadioGroup		rgp	Groups radio buttons, which presents an option that a user can toggle between selected/unselected states when the choices are mutually exclusive. (Only 1 selected at a time.)
RadioButtons		rad	Select only one option at a time.
CheckBox		cbx	Presents an option that a user can toggle between a checked/ unchecked state where more than one option can be selected.
Panel		pnl	Used to contain other components on a form.
Additional			
Image		img	Used to display a bitmap, icon, or metafile.
BitBtns		bmb	A Button that can also display a bitmap.
Shape		shp	Used to show geometric shapes such as an ellipse, circle, rectangle, square etc.
MaskEdit		med	Similar to an Edit, but also allows the programmer to specify particular formats for input e.g. dates and telephone numbers.
Samples			
SpinEdit		sed	Input of Integers

Win32			
RichEdit		red	Rich Text Format component, which supports properties and font formatting options.
ProgressBar		prb	A rectangular bar that gradually fills to provide visual feedback to the user about the progress of an operation.
PageControl		pgc	Used to make a multi-page dialog box.
StatusBar		stb	Provides an area to show the status of actions at the bottom of the screen.
System			
Timer		tmr	A non-visual component to trigger an event after a specified time interval.

The Component palette contains a wide array of components that we can drop onto our forms. It is important not only to know what components are available but also how to best utilise and combine these to create the most effective and easy-to-use interface.

This table is a brief summary of the components we have encountered thus far. Note that the list is by no means complete and while more components will be discussed in later grades, you are encouraged to browse through the different component palettes. There are many other useful ones that we have not yet used.



Data Types

Integer data types

Type	Lower Limit	Upper Limit	Size in Bytes
Shortint	-128	127	1 byte
Byte	0	255	1 byte
Word	0	65535	2 bytes
Integer/Longint	-2147483648	2147483647	4 bytes

Boolean data types

Type	Size	Contents
Boolean	1 bytes	0 (False) or 1 (True)

Characters

Type	Size	Contents
Char	1 byte	ANSI character set (ASCII codes)

Real data types

Type	Lower Limit	Upper Limit	Size in Bytes
Single	1.5×10^{-45}	3.4×10^{38}	4 bytes
Real/ Double	5.0×10^{-324}	1.7×10^{308}	8 bytes
Extended	3.6×10^{-4932}	1.1×10^{4932}	10 bytes

String data types

Type	Maximum length
ShortString	255 characters
String	Dynamic. Limited by available memory.

Refer to the Help for a comprehensive list of data types.

Bibliography

Brendan Murphy. 2000, *DELPHI in easy steps.* UK: Computer Step.

Frank Eller. 2002, *Delphi 6 Nitty Gritty.* Great Britain: Pearson Education Limited.

John Barrow et al. 1999, *Introducing Delphi Theory through practice Second edition.* South Africa: Unisa Press.

John Barrow et al. 2002, *Introducing Delphi Programming Theory through practice Third edition.* South Africa: Oxford University Press.

Miller, Todd & Powell David, et al. 1997, *Using Delphi 3 Special Edition.* USA: QUE Indiana

Reisdorph, Kent. 1998, *Teach Yourself Delphi 4 in 21 Days.* Indiana USA: Sams Publishing.

Index