Practical Book

INFORMATION TECHNOLOGY

10

basic education

Department:
Basic Education
REPUBLIC OF SOUTH AFRICA

MTN South Africa, through MTN SA Foundation, is a proud supporter of the CAT and IT digital books.

As an organisation rooted in technology, we believe in providing a new bold digital world to communities we operate in. This unique digital book provides the fundamental knowledge necessary for a sound grounding from which to make practical use of the complete and indispensable application-oriented information regarding Computer Applications Technology (CAT) and Information Technology (IT). It is a foundational reference for today's secondary school learners and teachers alike - as well as for the next generation of CAT and IT students.

Information Technology Practical Book Grade 10
ISBN 978-1-928388-50-0

Quality Assurance team for Information Technology
Allison Philander, Carina Labuscagne, David Peens, Denise van Wyk, Edward Gentle, Jugdeshchand Sewnanen, Julian Carstens, Magdalena Brits, Shamiel Dramat, Shani Nunkumar and Zainab Karriem

Contents

Dear Learner

Welcome to the *IT Practical Grade 10* textbook, and welcome to programming.

If this is your first time learning how to program, don't worry. This textbook has been designed to teach anyone – regardless of experience – how to program. If you follow along with all the examples then you will be an experienced programmer who has written more than 50 programs by the end of this book.

Programming and programming languages, much like real languages, can only be learned through practice. You cannot sit at home and learn to speak French from a textbook. In the same way, you cannot read this book and hope to be a programmer at the end of it. Instead, you will need to write every bit of code and create every program shown in this book. Even if all you do is follow the steps of the examples on your own computer, you will learn how to write code. Once you have mastered the code, you will be able to comfortably use it in your own programs.

For you to master programming, try to work through as many of the programs given to you. Each program has been designed to both teach you new concepts and reinforce existing concepts. The book will start by teaching you how to create simple programs. However, by the end of the book you will be creating useful programs and fun games to play.

Programming is not only about knowing and using the programming language. There are also important theoretical concepts that you will need to understand, and planning and problem-solving tools that you will need to master. The best-coded program in the world will not be useful if it solves the wrong problem. This book has therefore been divided into the following chapters:

- Chapter 1: Algorithms
- Chapter 2: Delphi
- Chapter 3: Variables and components
- Chapter 4: Solving basic mathematical problems using Delphi
- Chapter 5: Decision making
- Chapter 6: Validating data
- Chapter 7: Repetition
- Chapter 8: String manipulation
- Chapter 9: PAT preparation

Before getting started with algorithms, watch the video in the QR code.

To give you the most opportunities to learn, this book will give three types of programming activities:

Examples

Examples will guide you through the creation of a program from start to finish. All you need to do with examples is to follow the step-by-step guidance provided to you.

Example 1.1 Making hot chocolate

Most people have a specific way they prefer to drink their hot chocolate. The algorithm below describes one way to make hot chocolate.

1. Fetch a cup from the cupboard.
2. Place the cup on the counter.
3. Add water to the kettle until there is 500 ml of water in the kettle.
4. Turn on the kettle.
5. Add four teaspoons of hot chocolate to the cup.
6. Add 30 ml of milk to the cup.
7. Add one teaspoon of sugar to the cup.
8. Stir the mixture for 10 seconds.
9. Add boiling water to the cup until the cup is 95% full.
10. Stir the mixture for 10 more seconds.
11. Your hot chocolate is now ready to drink!

Guided activities

Guided activities have a program that you need to create on your own. Your teacher will provide you with the solution. These solutions should be used as an opportunity to compare your program, and to see where you may have made errors or left something out.

Guided Activity 1.4 Outfit selection algorithm

Different people have different ways to choose what they will wear in the morning. Some people will choose the first thing they see, while other people can spend up to an hour deciding.

Read through the following algorithms, paying careful attention to their quality.

Top selection

```
1.  Open your cupboard.
2.  Select the closest shirt.
3.  Select the closest pair of pants.
4.  Select the closest underwear.
5.  Wear the selected items.
```

Activities

Activities are programs that your teacher can give to you as classroom activities or homework. With these programs, you will only be assessed on how well your program works, so use your creativity to come up with a solution!

Activity 1.1

1.1.1 Follow each step exactly and in the sequence indicated. Do not look at your classmates' drawings and do not speak to one another. Use your own interpretation of the instructions.

a. Draw a diagonal line.
b. Draw another diagonal line connected to the top of the first one.
c. Draw a straight line from the point where the diagonal lines meet.
d. Draw a horizontal line over the straight line.
e. At the bottom of the straight line, draw a curvy line.

'Take note' and 'Did you know' boxes

The boxes provide extra, interesting content that might caution you to 'take note' of something important; or give you additional information. Note that the content in the 'Did you know' boxes will not be part of your exams.

Take note

- The PAT is a compulsory component of the final end-of-year examination for IT.
- The PAT counts 25% of your final mark for IT. It is important that you produce work of a high standard.

Did you know

The Microsoft Windows operating system is made up of 50 million lines of code. Can you imagine trying to figure out how the code works if all the variables have names like 'String1' or 'x'?

New words

loops – loops repeat certain lines of code until a specific condition is met

New words

These are difficult words that you may not have encountered before. A brief explanation for these words are given.

QR Codes, Videos and Screen captures

These will link you to online content. When you are in the eBook, you can easily access the links.

LEARNING ABOUT ORDER OF OPERATIONS

https://www.youtube.com/watch?v=dAgfnK528RA

Consolidation activities

This is a revision activity based on what you have covered in the chapter. Take time to answer the questions on your own. You teacher may also use these to assess your performance during class.

Consolidation activities Chapter 3: Variables And Components

1. In your own words, give a definition for the word 'variable' as it relates to computer programming.
2. List five Delphi naming rules and conventions for variables.
3. In your own words, what is the difference between an integer and real variable types?
4. What is the difference between syntax, runtime and logic errors.
5. What is the purpose of a variable name?
6. Describe the relationship between the variable name and the memory location.

TERM 1

CHAPTER 1

ALGORITHMS

CHAPTER UNITS	
Unit 1.1	Basics of algorithms
Unit 1.2	Algorithm quality
Unit 1.3	Creating algorithms
Unit 1.4	Flowcharts

Learning outcomes

At the end of this chapter, you should be able to:

- explain what an algorithm is
- give examples of algorithms in everyday life
- produce an algorithm to solve a problem
- test algorithms to determine the quality and accuracy
- compare algorithms considering, for example, order and precision
- use tools, such as a basic flowchart to represent an algorithm.

INTRODUCTION

Every day of your life you make use of lists of steps to complete certain tasks. For example, you might:

- follow a recipe
- download software or music
- use a car repair manual
- set up a music playlist
- follow a knitting pattern
- call a friend using a phone
- read from a music sheet
- follow written instructions to complete a task.

Each list of steps for the tasks above are an example of an **algorithm**.
An algorithm is an ordered list of steps used to carry out a task or solve a problem. It is important to both computers and programmers. As a programmer, your job will be to tell a computer what to do in different situations. To do this, you can create an algorithm and write a computer program.

Before you can start creating programs and writing code, you first need to learn how to create an algorithm. In this chapter, you will learn more about algorithms, including what algorithms are, how to evaluate the quality of an algorithm, how to create algorithms and how to create flowcharts.

Remember that this information will be used throughout this year when you create programs.

New words

algorithm – an ordered list of steps for carrying out a task or solving a problem.

Take note

Computer programs are simply lists of instructions (algorithms). If an algorithm is not correct, it will cause an error in the program you are writing.

UNIT

1.1 Basics of algorithms

In this unit you will look at some of the basics of creating algorithms. While there are no specific rules about how to write an algorithm, once it is complete it should meet the following criteria:

- there must be a limited number of steps
- the steps must be:
  - easy to understand and follow
  - detailed and specific
  - clear and **unambiguous**
- each step should:
  - consist of a single task
  - be at the most basic level that cannot be broken into simpler tasks
- all repetitions must have clear ending conditions
- there must be at least one result (or output).

New words

unambiguous – not open to more than one interpretation

Look at the following example to help you understand the importance of following a list of instructions.

Example 1.1 Making hot chocolate

Most people have a specific way they prefer to drink their hot chocolate. The algorithm below describes one way to make hot chocolate.

1. Fetch a cup from the cupboard.
2. Place the cup on the counter.
3. Add water to the kettle until there is 500 ml of water in the kettle.
4. Turn on the kettle.
5. Add four teaspoons of hot chocolate to the cup.
6. Add 30 ml of milk to the cup.
7. Add one teaspoon of sugar to the cup.
8. Stir the mixture for 10 seconds.
9. Add boiling water to the cup until the cup is 95% full.
10. Stir the mixture for 10 more seconds.
11. Your hot chocolate is now ready to drink!

By following this algorithm, you should be able to make a cup of hot chocolate. However, this hot chocolate might not be exactly to your taste. Think and talk to your friend about how your algorithm for making hot chocolate (or coffee or tea) would be different to the one in the example.

CREATING BASIC ALGORITHMS

Most algorithms can be broken into smaller steps that use algorithms designed for each of the smaller steps. For example, in the hot chocolate algorithm above, you could create an algorithm that describes how to open the cupboard, how to select a cup, how to open the hot chocolate container, and so forth. This algorithm could end up being hundreds of lines long!

To prevent this, you need to focus on completing a specific task without going into detail for each sub-task.

But let's begin by seeing how well you can follow instructions.

Activity 1.1

1.1.1 Follow each step exactly and in the sequence indicated. Do not look at your classmates' drawings and do not speak to one another. Use your own interpretation of the instructions.

- **a.** Draw a diagonal line.
- **b.** Draw another diagonal line connected to the top of the first one.
- **c.** Draw a straight line from the point where the diagonal lines meet.
- **d.** Draw a horizontal line over the straight line.
- **e.** At the bottom of the straight line, draw a curvy line.
- **f.** Draw a diagonal line for the bottom of the first diagonal to the straight line.
- **g.** Draw a diagonal line from the bottom of the second diagonal to the straight line.

1.1.2 Compare your picture with your partner's.

- **a.** Are your pictures different?
- **b.** Can you explain why?
- **c.** What was difficult about following the instructions?
- **d.** What was missing from the instructions?

1.1.3 Write down the criteria that a well-designed algorithm should meet.

Your teacher will tell you what the object was that you should have drawn. Once you know this, write a set of instructions that someone could follow to draw the object. Make sure that:

- there is only one way to interpret each step, that is, the instructions are unambiguous
- you provide enough detail in each step.

Did you know

You may have discovered that good instructions/ algorithms that work on a first try are hard to develop.

Activity 1.2

Write instructions (or an algorithm) that will enable someone to make a paper shape from one sheet of A4 paper.

Once your instructions (algorithm) are complete, swap them with another learner and test if it was easy to follow. Answer the following questions.

1.2.1 Were the instructions easy to follow? Did it work?

1.2.2 If the instructions did not work, where did your classmate go wrong?

1.2.3 **a.** What do you need to do to fix it?

- **b.** Adjust the instructions.

Activity 1.3

Write down instructions on how to do the following physical activities. Each activity must have at least five steps that are properly explained.

1.3.1 How you make coffee.

1.3.2 How you do a sit-up.

1.3.3 How to buy a packet of chips at the tuckshop using a R100-note.

1.3.4 How you travel from your house to your school.

In the next unit we will explore different qualities that can help us determine whether the quality of an algorithm is good.

UNIT

1.2 Algorithm quality

To evaluate the quality of an algorithm, you can use two criteria:

- precision
- order

This section will look at how each of these criteria are evaluated.

WHAT IS BETA TESTING?

https://www.youtube.com/watch?v=c5F-mutZ7Yo

PRECISION

Precision refers to how accurately and reliably an algorithm solves a problem. The more precise an algorithm is, the better it is at solving the problem correctly regardless of the situation.
Imagine that you created the following algorithm for baking a cake.

Imprecise cake algorithm

1. Put flour, baking powder, eggs, sugar, butter, and vanilla essence in a bowl.
2. Mix the ingredients.
3. Put the mixture in a pan.
4. Bake the mixture.
5. Remove the baked cake and eat it.

If someone has never baked a cake before, how often do you think this algorithm will result in a successful cake?

The problem with this algorithm is that it is not specific enough. Different people will use different amounts of each ingredient, use different temperatures, and bake the mixture for a different length of time. Since only a few of these attempts will result in a tasty cake, the algorithm is not very precise.
A more precise cake algorithm might look like this:

Precise cake algorithm

1. Turn the oven on and set the temperature to 180 °C.
2. Sift 200 grams of self-raising flour and two tablespoons of baking powder into a bowl and set aside.
3. Melt 100 grams of butter in the microwave at a low heat for 30 seconds and set aside.
4. Use eight large eggs and separate the egg whites and yolks.
5. Place the egg whites in a large bowl.
6. Beat the egg whites until they are frothy.
7. Slowly mix 300 grams of sugar into the egg whites.
8. Add the egg yolks to the egg white and sugar mixture, and beat the mixture for five minutes.
9. Add two teaspoons of vanilla essence to this mixture.
10. Now add the sifted flour and baking powder mixture, as well as the melted butter to the bowl.
11. Use a wooden spoon and mix the ingredients for five minutes.
12. Pour the mixture into a greased, round baking pan.
13. Place the baking pan into the oven.
14. After 30 minutes, remove the cake from the oven.
15. Enjoy the cake.

While this algorithm is more difficult to create, it is a lot easier to follow and is much more likely to deliver a tasty cake. Even if you have never baked a cake before, you should be able to use this algorithm. If you follow the instructions, the cake should always be a success. This algorithm can thus be called a **high precision algorithm**.

ORDER

Order refers to the total number of steps needed (including repeats) to complete an algorithm. The order of an algorithm is usually shown as a mathematical formula based on the number of inputs. Order will have a significant effect on the time it takes to complete an algorithm, especially when working with large numbers of items.

For example, if you have an algorithm to choose what you will wear each morning and you only have a single pair of pants, then you will only need to make one decision (wear pants or do not wear pants). If you have a pair of pants and a shirt, then you will need to choose between four options (wear pants and a shirt, wear only pants, wear only a shirt or wear nothing). If you have three items, you will need to choose between nine options!

Let's work through the following activity together, and then also look at the solution.

Guided Activity 1.4 Outfit selection algorithm

Different people have different ways to choose what they will wear in the morning. Some people will choose the first thing they see, while other people can spend up to an hour deciding.

Read through the following algorithms, paying careful attention to their quality.

Top selection

```
1.  Open your cupboard.
2.  Select the closest shirt.
3.  Select the closest pair of pants.
4.  Select the closest underwear.
5.  Wear the selected items.
```

Full comparison

```
1.  Open your cupboard.
2.  Select the closest shirt.
3.  Compare it with each pair of pants.
4.  Give each combination a score out of 100.
5.  Select the next closest shirt.
6.  Repeat step 3 to 5 until you have evaluated all combinations.
7.  Choose the combination with the highest score.
8.  Compare the selected combination with each pair of underwear.
9.  Give each underwear combination a score out of 100.
10. Select the combination with the highest score out of 100.
11. Wear the selected items.
```

Which of the following two outfit selection algorithms do you think have the highest order and precision?

SOLUTION

In terms of the order, the first algorithm has a much smaller order than the second algorithm. The first algorithm will always take exactly five steps to complete, while the second algorithm could take hundreds of steps to complete.

In terms of precision, the first algorithm will only give the best combination of clothes if the closest items make up the best combination. The algorithm will fail most of the time and is therefore not precise. The second algorithm will always give the best combination of clothes and is therefore more precise.

Activity 1.5

1.5.1 Write a definition for the terms order and precision and explain the difference between the two terms using an example.

1.5.2 Different people have different ways to choose where they go shopping. Some people will choose the shop close to where they live, while other people will drive a long distance to get to a shop where they think they will get better discounts.

Read through the following algorithms, paying careful attention to their quality.

Shop selection

```
1.  Estimate the distance to the shop.
2.  Select the closest shop.
3.  Select the groceries you want.
4.  Select the teller you want to pay at.
5.  Go home with your shopping.
```

Full comparison

```
1.  Estimate the distance to the shop.
2.  Select the shop that will offer the greatest value for money.
3.  Compare it with the second choice of shop.
4.  Give each shop a score out of 100.
5.  Select the next closest shop.
6.  Repeat step 3 to 5 until you have evaluated all combinations.
7.  Choose the combination with the highest score.
8.  Compare the selected combination with each distance to shop.
9.  Give each shop and distance combination a score out of 100.
10. Select the combination with the highest score out of 100.
11.  Visit the shop of your choice.
```

Which of the two shop selection algorithms do you think have the highest order and precision? Explain your choice.

UNIT

1.3 Creating algorithms

There are many possible algorithms that can be used to complete the same task and achieve the same goal. However, not all algorithms are equally good. If someone has ever made you bad coffee or your search engine has ever given you the wrong results, then you have seen what happens when an algorithm fails!

So how can you make sure your algorithm does what it is supposed to do? Let's look at some of the steps you can follow to make sure that you are able to create high quality algorithms:

- **Understanding the problem:** The first step in creating an algorithm is to understand what problem the algorithm should solve. If you don't know this, you can easily create an awesome algorithm, but it will solve the wrong problem.
- **Defining the desired solution or output:** Defining what your solution or output is, will depend on the problem you are trying to solve. In other words, defining your desired output is closely linked to understanding what the problem is. You may need an exact answer, or you may simply need a close enough estimate. Defining the desired output could therefore have a significant effect on the time it takes to solve the problem.
- **Defining the inputs:** The third step is to define what information you will need for the algorithm to work. This information can be entered by a user or obtained from a different source, such as an existing file.
- **Designing a set of steps to complete the task:** The fourth step is to design a set of steps that will use your inputs to complete the task. These steps need to be specific and detailed enough that they will always give you the correct answer.
- **Testing the algorithm:** Every algorithm should be tested, and not just using the situations or data you are most likely to come across. One technique for testing an algorithm is to create a trace table. You will learn about this in Chapter 4.
- **Updating the algorithm:** If your tests reveal any problems with the algorithm, you should change the steps to fix the problem. Once changed, the new algorithm should be tested to ensure it works as expected.

To see how these steps can be used to create an algorithm, work through the example below.

Example 1.2 Getting dressed

You have been asked to write an algorithm that will help your little sister to get dressed in the morning. Using the steps listed above, create a 'Getting dressed algorithm'.

Understand the problem

The first step is to understand the problem. In the case of your little sister getting dressed, this means understanding what your little sister can do. For example, if your little sister does not know how to put on clothes, then you will need to create algorithms for how to put on each piece of clothing. If it she does not know how to identify different types of clothes, then you will need to create visual identification algorithms for her.

After speaking to your mother, you find out that your little sister already knows how to identify and put on clothes, so you only need an algorithm for the general process of getting dressed. With this information in mind, you can continue to the next step.

Define the desired output

Now that you know what the problem is, you need to determine what the desired result is. How many pieces of clothing should your little sister be wearing? Do the clothes need to match or be fashionable? Depending on what the desired output is, your algorithm will change.

Example 1.2 Getting dressed *continued*

Based on your conversation with your mother, the output is as follows:

- Your little sister needs to be dressed in underwear, skirt or shorts and a t-shirt.
- Your little sister should not wear any duplicated pieces of clothing.
- The clothes do not need to match or be stylish.

Define the inputs

The final piece of information you need is what the inputs into the algorithm are. This is any information available to you that needs to be used in the algorithm. According to your mother, the inputs are as follows:

- Your little sister will be wearing pyjamas at the start of the algorithm.
- There will always be exactly five t-shirts, three pairs of shorts or three skirts, and five pieces of underwear available.

Design a set of steps to complete task

Finally, you can write your algorithm:

1. Walk to the cupboard.
2. Open the cupboard.
3. Open the washing basket.
4. Remove your pyjamas.
5. Place your removed pyjamas in the washing basket.
6. Take one pair of underwear from the cupboard.
7. Put on the underwear.
8. Take a shorts or skirt from the cupboard.
9. Put on the shorts or skirt.
10. Take a t-shirt from the cupboard.
11. Put on the t-shirt.
12. Close the cupboard to complete the algorithm.

Test the algorithm

Once you have created an algorithm, you need to test it. By following the steps exactly as they are written, you can see if the algorithm returns the desired result. With your dressing algorithm, perhaps you will realise that there is more than one cupboard in the house and that your little sister often tries to find underwear in your cupboard!

Update the algorithm

The final step is to update the algorithm. This gives you a chance to fix any problems identified while testing the algorithm. For example, you can update your algorithm to specify exactly which cupboard your little sister should be used when getting dressed. Once the change has been made, you need to test the algorithm again to make sure the changes did not introduce new problems.

While designing an algorithm might sound time consuming at first, it will save you a lot of time eventually. Professional programmers spend half their time fixing problems and these problems are often caused by problems with their algorithms. By making sure your algorithm is correct from the start, you will stop yourself from having to first identify the mistake and then fix it later.

Watch out!

A computer program or algorithm is not like another person. It will never fill in 'obvious' missing information or make assumptions. Instead, it will always do exactly what you tell it to do, even if it is obviously wrong or pointless.

Activity 1.6

Someone that has never seen your school needs to find your IT classroom when entering the school: Apply the steps when creating the algorithm to guide the person from the gate to the IT classroom.

1.6.1 Write down what the goal of the algorithm is.

1.6.2 Write down how you will determine if the algorithm is successful.

1.6.3 Write down any information the user will need to complete your algorithm.

1.6.4 Write down the algorithm to walk from the gate to the IT classroom.

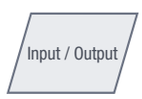
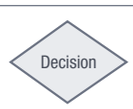
1.6.5 Swap algorithms with one of your classmates, then test each other's algorithms. Make sure you follow the steps exactly as they are written.

1.6.6 Update your algorithm if it did not successfully guide your classmate to the IT classroom.

UNIT

1.4 Flowcharts

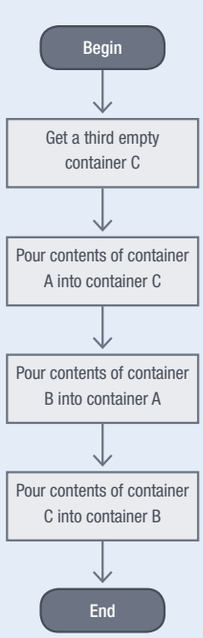
A flowchart is a visual representation of an algorithm. It can be made up of five different elements.

ELEMENT	FUNCTION	SHAPE
Terminal	Indicates the start and end of an algorithm.	
Input/Output	Shows when data is added to the algorithm or given to the user.	
Instruction	Gives an instruction that the algorithm must follow.	
Decision	Shows a decision (or condition) that affects the algorithm's behaviour.	
Connector	Connects one element of the algorithm to the next element. Shows the direction in which you move from one element to the next.	

By combining these items, you can visually show any algorithm. As you progress you will use different flowchart symbols.

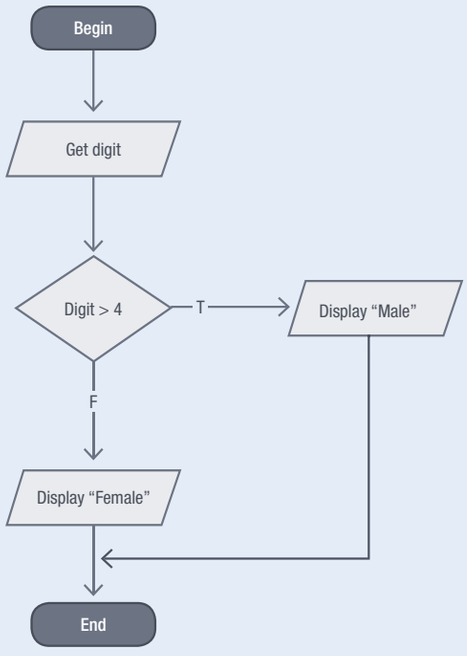
Example 1.3 Swapping contents of two containers flowchart

Here is a flowchart of an algorithm that you can use to swap the contents of container A that contains milk with contents of container B that contains water. Study it carefully.



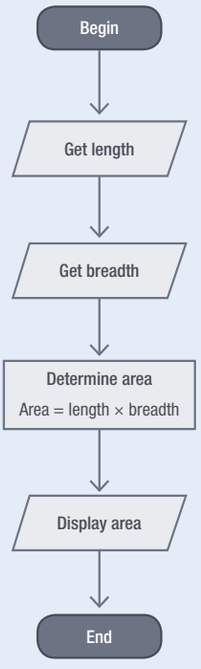
Example 1.4 Determining whether a person is male or female using their ID numbers

Here is a flowchart of an algorithm that you can use to determine whether a person is a male or a female using the seventh digit of their ID number. If the seventh digit is greater than four, then the person is male, otherwise the person is female.



Example 1.5 Determining the area of a rectangle

Here is a flowchart of an algorithm that you can use to determine the area of a rectangle.



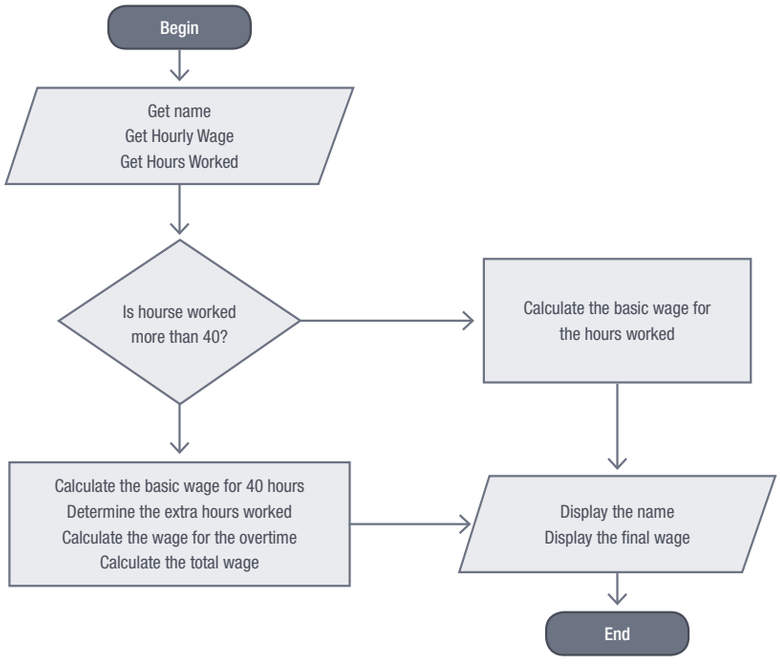
Activity 1.7

1.7.1 Draw a flow chart for the following algorithms:

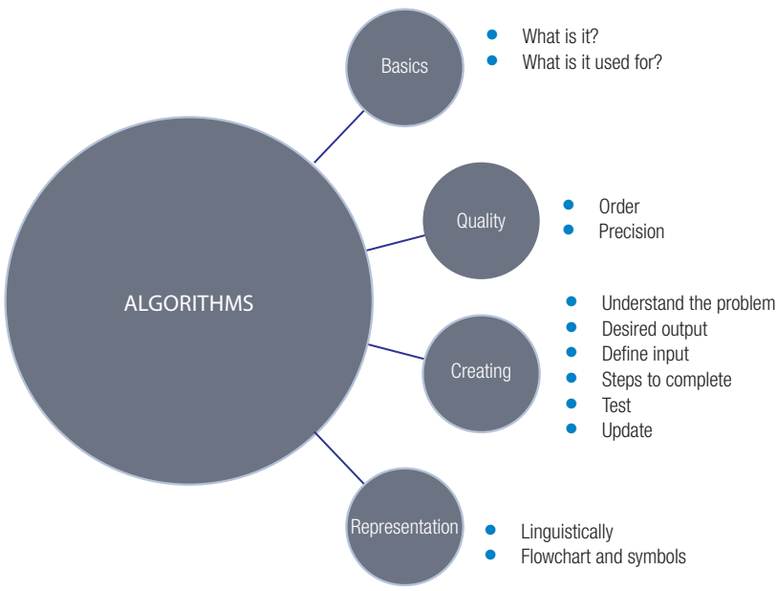
- **a.** Making hot chocolate (refer to algorithm on page 3).
- **b.** Crossing the street.
- **c.** Finding the area of a circle.
- **d.** Entering a contact into your cell phone.

1.7.2 Evaluate the following flowchart.

- **a.** Check that all the steps listed are needed.
- **b.** If there is anything wrong or missing, identify the missing step and improve the flowchart.
  This flowchart is based on the following scenario: A employee receives a weekly wage for 40 hours worked. The wage is calculated by multiplying the number of hours with the hourly wage. Hours above 40 are overtime and the wage increase to one and a half of the normal hourly wage.



Consolidation



Consolidation activity — Chapter 1: Algorithms

1. Answer the following questions in your own words.
   - a. What is an algorithm?
   - b. What criteria should an algorithm meet?
   - c. Briefly describe the steps that you should take to create a high-quality algorithm.
   - d. What criteria do you need to evaluate the quality of an algorithm?
   - e. Name each element in a flowchart and draw their relative shapes.
2. Sequence the following steps for washing hands so that they are in a logical order:

Turn off the tap
Dry your hands
Rub your hands together
Rinse the soap away with
Rinse your hands with water
Putsoap on your hands
Turn on the tap

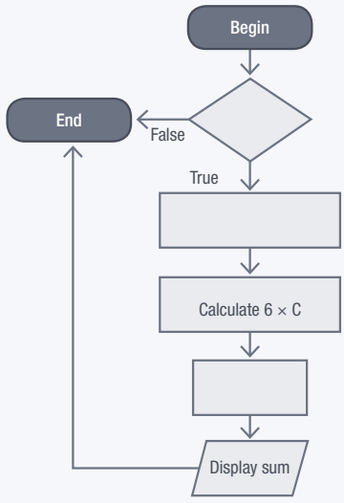
[Source: SA Computer Olympiad Talent Search competition, sourced from the Berbras competition]

3. The following steps, if completed correctly, will represent an algorithm to multiply a two-digit number by a one-digit number in the form: AB × C where A, B and C represent digits, for example:

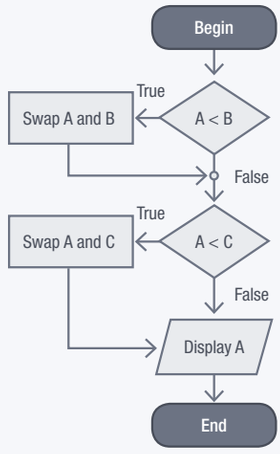
A	B		C
1	6	×	7

Fill in the missing instructions to complete the algorithm for multiplying 16 by any one-digit number (C).

Hint: C cannot be 0.

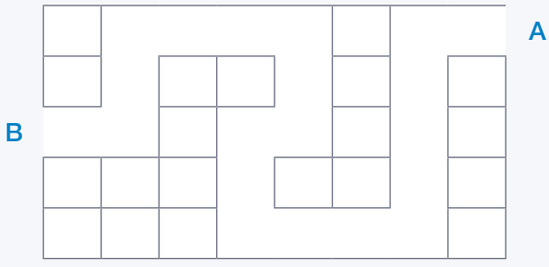


4. What is the purpose of the following algorithm?



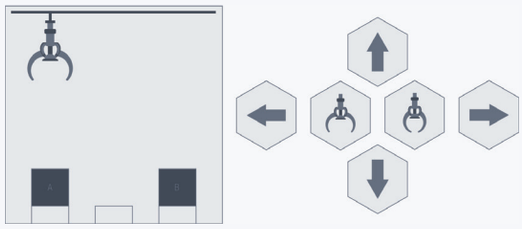
5. Create an algorithm to move from point A to point B.

Each block represents 10 steps



6. The crane in the port of Lodgedam responds to six different input commands:

a. Left
b. Right
c. Up
d. Down
e. Grab
f. Release

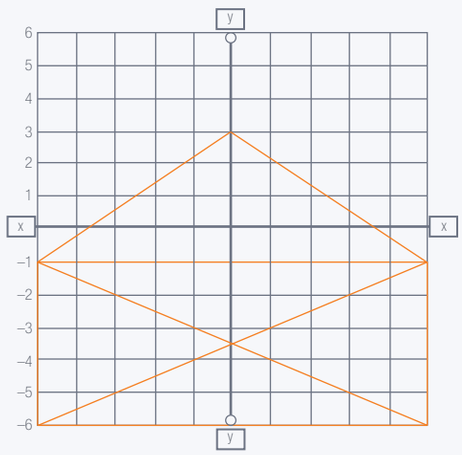


Crate A is in the position on the left, crate B is in the position on the right. Which algorithm is the correct one to swap the position of the two crates? Write down the letter of the correct answer.

A. (Down, Grab, Up, Right, Down, Release, Up)
B. (Down, Grab, Up, Right, Down, Release, Up) (Right, Down, Grab, Up, Left, Left. Down, Release, Up) (Right, Down, Grab, Up, Right, Down, Release)
C. (Right, Right, Down, Grab, Up) (Left, Left, Down, Release, Up)
D. (Down, Grab, Up, Right, Right, Down, Release, Up) (Down, Grab, Left, Down, Release, Up) (Down, Grab, Up, Right, Down, Release, Up)

[Source: SA Computer Olympiad Talent Search competition, sourced from the Berbras competition]

7. You have to draw a house as shown below:



The following 'rules' must be followed when drawing the house:

- You may not lift your hand/your pen.
- You may not draw on a line that has already been drawn.

Someone created two algorithms for drawing the house according to the above rules. Follow each algorithm to see if it complies to the rules above. If you find that an algorithm does not comply with the rules, rewrite it so that it is in line with the rules.

The coordinates are in the format (x, y), e.g. (1, 2) refers to x = 1 and y = 2 on the grid.

Algorithm 1	Algorithm 2
1. Start at (5, –1)	1. Start at (5, –6)
2. From the above position, draw a diagonal line to (–5, –6)	2. From the above position, draw a diagonal line to (–5, –1)
3. From the position in step 2, draw a straight line to (5, –6)	3. From the position in step 2, draw a straight line to (5, –1)
4. From the position in step 3, draw a diagonal line to (–5, –1)	4. From the position in step 3, draw a diagonal line to (–5, –6)
5. From the position in step 4, draw a diagonal line to (0, 3)	5. From the position in step 4, draw a straight line to (5, –6)
6. From the position in step 5, draw a diagonal line to (5, –1)	6. From the position in step 5, draw a straight line to (5, –1)
7. From the position in step 6, draw a straight line to (5, –6)	7. From the position in step 6, draw a diagonal line to (0, 3)
8. From the position in step 2, draw a straight line to (–5, –1)	8. From the position in step 7, draw a diagonal line to (–5, –1)
9. From the position in step 8, draw a straight line to (5, –1)	9. From the position in step 8, draw a straight line to (–5, –6)

CHAPTER 2

DELPHI

CHAPTER UNITS

Learning outcomes

At the end of this chapter, you should be able to:

- clearly explain the Delphi programming environment and principles
- explain how the Delphi IDE can be used to create programs
- identify, discuss and use various Delphi components
- identify, discuss and apply different component properties
- create various Delphi events
- use the correct Delphi syntax.

INTRODUCTION

Delphi is one of many high-level programming languages. This means that it was designed to be easier to write and be read by people. This quality makes it an ideal language to use when learning about programming.

In this chapter, you will learn:

- how the Delphi IDE can be used to create programs
- how to create, open and save Delphi projects
- about the basic components you can use to create a simple graphical user interface (GUI)
- how to work with component properties
- how to create a Delphi event to change component properties and behaviour
- how to create basic Delphi code.

Did you know

As you learn programming, you should focus on understanding the programming concepts and logic. Once you understand that, learning any programming language is easy!

UNIT

2.1 Opening Delphi and exploring the Delphi IDE

Take note

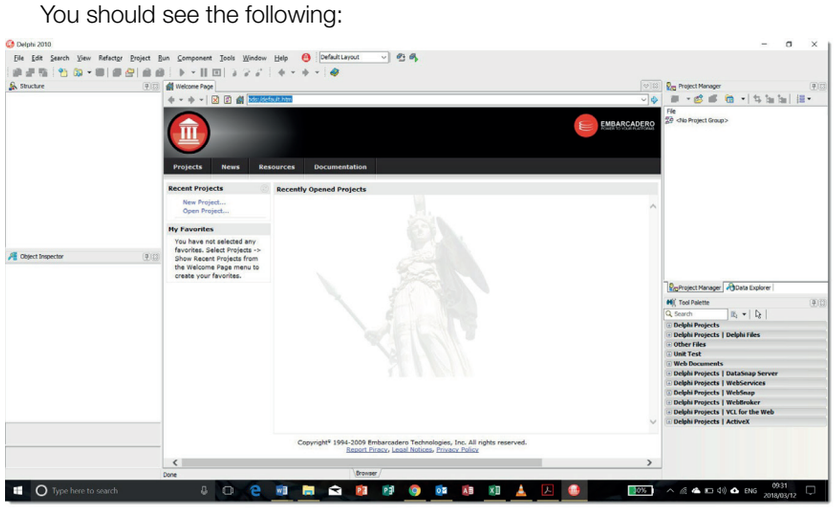
You will find the instructions for how to install the Delphi Integrated Development Environment (IDE) in Appendix A. Follow the instructions carefully and ensure that your program is able to open correctly.

In this unit you will explore the Delphi IDE and begin creating simple programs.

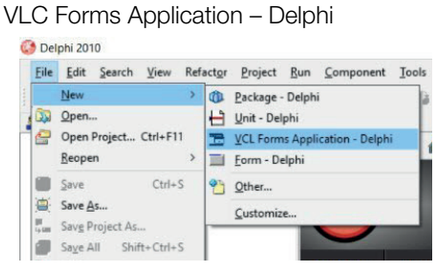
Carefully follow the step-by-step instructions below:

INSTRUCTIONS FOR OPENING DELPHI AND EXPLORING THE IDE

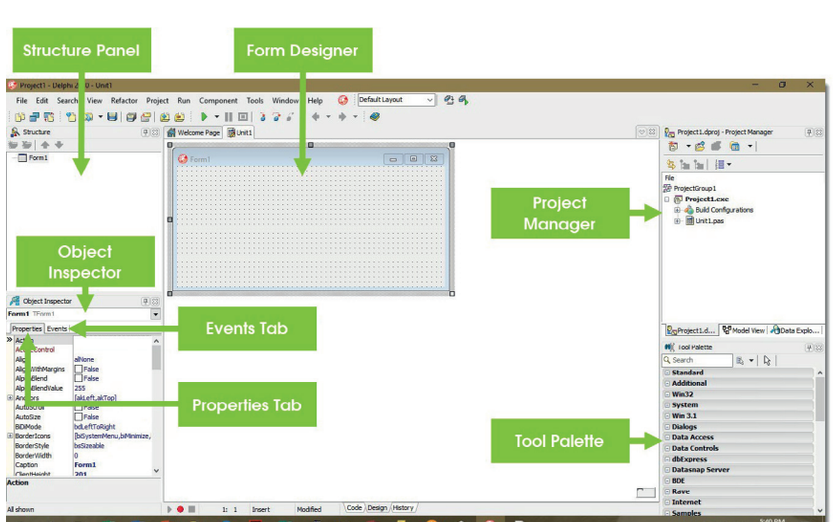
1. Open Delphi.
   You should see the following:



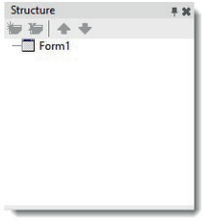
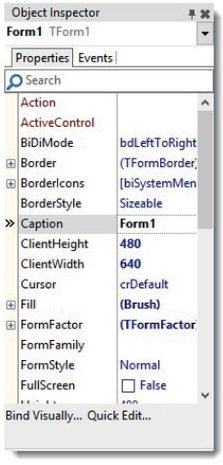
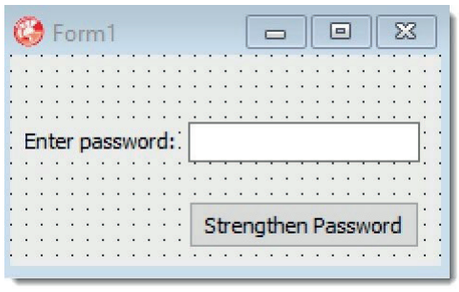
2. To create a new Delphi Application, click on:
   File → New → VLC Forms Application – Delphi

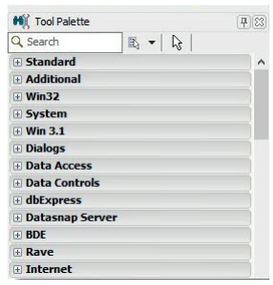
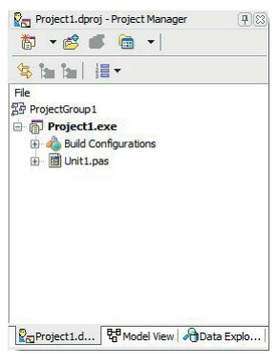


3. The following screen will open. This is the IDE where you create applications.



4. The Delphi IDE includes the following:

Menu bar	The **File** menu allows you to create new applications, open and reopen projects, save projects, and print. The **Edit** menu allows you to cut, copy, and paste. The **View** menu allows you to view various components and tools. The **Run** menu allows you to execute (or run) a Delphi program and to debug a Delphi program.	
Toolbar	The Toolbar enables you to quickly access features and tools to help you create your application.	
Structure panel	The Structure panel is a quick reference to various elements in your project. In the structure panel you will find Classes, Variables/ Constants and Uses folders. The structure panel allows you to expand and collapse subfolders.	
Object Inspector	The Object Inspector enables you to set the initial properties and the way a component behaves and appears in your application. The Object Inspector can be arranged by Category or by Name. If you want to change the arrangement of the Object Inspector, click any place inside the Object Inspector, then right-click the mouse, and a pop-up menu will appear. Click on Arrange and select the option you want to use.	
The Form Designer	The Form Designer is used to design the user interface for the program you are writing. All other components are placed on this form. These components are found on the Tool Palette and you will place it on the form during design mode. If you run a program, the form becomes executable. This is referred as the run mode.	

The Tool Palette	Delphi components are grouped together on different roll-up menus on the Tool Palette. The two pages that you will use are the Standard page and Additional page.	
The Project Manager	The Project Manager enables you to compile and build projects.	

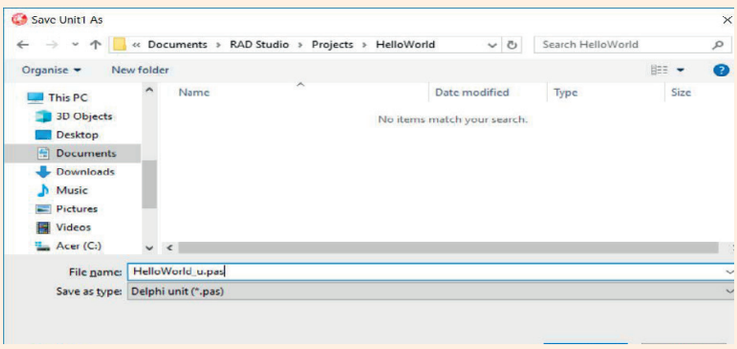
SAVING A DELPHI PROJECT

It is important that you save your Delphi project correctly. Each program you create in Delphi will have two files: a unit file and a project file. The unit file refers to the specific user interface and code you are creating in Delphi. The project file is a container for all the units used in your program. To make sure you do not lose information, always save both the unit file and the project file. Each Delphi project also needs to be saved in its own folder. Your teacher will guide you to create your folder on a network drive or local computer.

Example 2.1 Saving a project

To save your new project:

1. Create a folder for your Delphi project.
2. Open the *File* menu and select the *Save Project As* option.
3. Navigate to the folder that you created.
4. The first file that you will be saving is your unit file.
5. In the *File name* textbox, enter the name "**HelloWorld_u**" and click Save.



Example 2.1 Saving a project *continued*

6. The *Save Project As* window will open again. Enter the name "**HelloWorld_p**" and click Save. This is the project file for the 'Hello World' application.
7. Your application has now been saved. Every time you now make a change to the project, you can use the *CTRL + S* **shortcut key** to save the project.

Now it's time for you to practise saving a new project. If you feel unsure of what to do, refer to the example above to help you.

Activity 2.1

2.1.1 Create a new folder, called Activity2_1.

2.1.2 Create a new project.

2.1.3 Save the unit file in the folder you created as **Activity2_1_u**.

2.1.4 Save the project file in the folder you created as **Activity2_1_p**.

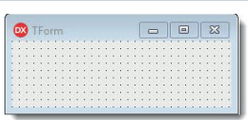
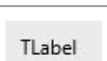
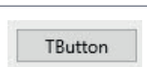
Take note

- The unit and project names may not contain spaces.
- Add _u to the unit name.
- Add _p to the project name.

2.2 Components and properties

COMPONENTS

In Delphi you can add images, buttons, labels and menus to a user interface in a few minutes. To create user interfaces quickly and effectively, you need to know which components are available to you. There are many different components that can be used in different scenarios. However, the table below lists a few of the most important components you will use this year.

NAME	DESCRIPTION	PICTURE
TForm	A form is the component that all other components are placed on. It provides the foundation for your user interface.	
TLabel	A simple label can be used to display text in the user interface (UI).	
TButton	A button that can be pressed by users to trigger an event.	
TEdit	A text box that can be used to obtain text input from users. It can also be used to display output, but this would be by exception.	
TImage	A frame that can be used to display images.	

PROPERTIES

Properties determine the way the components in your program look. You can change the look of your program by changing the properties of components. Because Delphi is an object-oriented language, it can write data to components and read data from components. For example, if you had a label component in your program, it would have multiple properties that you could read or write to. The properties for the label component include the caption, name, font colour, background colour and font size. You could also change the height and width property of the label component to resize it.

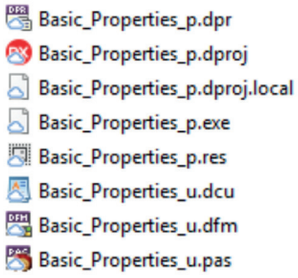
The table below shows a few of the most useful properties that you can use for Delphi components:

NAME	DESCRIPTION
Name	Sets the name of the component.
Caption	Sets the text shown by several different components (including labels, buttons, textboxes and forms).
Text	Sets or reads the text entered into a textbox (or *TEdit*) component.
Picture	Sets or reads the picture shown by image components.
Height	Takes a number value that sets the height of the object.
Width	Takes a number value that sets the width of the object.
Font.Size	Takes a number value that sets the size of the font.
Font.Color	Sets the colour of the text.

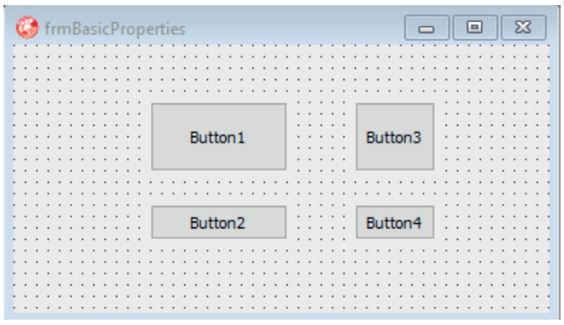
Activity 2.2

2.2.1 Open the Delphi project **Basic_Properties_p.dproj** by:

- navigating to the folder where your project is saved.
- double clicking on the project file (.dproj)



2.2.2 After opening Basic_Properties_p.dproj, you will see the following screen:



2.2.3 Do the following using the Object Inspector:

a. change the form's caption to your name and surname.

b. change Button1's name to btnRed and the caption to Red.

c. change Button2's name to btnGreen and the caption to Green.

d. change Button3's name to btnBlue and the caption to Blue.

e. change Button4's name to btnPurple and the caption to Purple.

2.2.4 a. Change the height of btnGreen to 65 and the width to 121.

b. Change width and height of btnBlue to have the same width and height as btnPurple.

2.2.5 Go to the form's font property and change the font to size 12.

2.2.6 Apart from the properties listed above, choose two other properties of BtnGreen that you can change.
Copy the table below and complete it. Describe the result of the changes you made.

COMPONENT	PROPERTY	DESCRIBE RESULT OF CHANGE
BtnGreen		
BtnGreen		

Take note

The last two properties from the table (Font.Size and Font.Color) both have a full stop in them. To understand why this is, you need to understand what a font does. The font of a component determines the way the text looks, which includes the size of the text, the style of the text and the colour of the text. Since the look of the text is determined by all these elements, they are all properties of the font. However, the font itself is a property of a button or label. So size and colour are properties of a property. To show that something is a property of a property, you use the dot notation shown above.

UNIT

2.3 Creating a simple Delphi project

So far you have learnt about the different components in Delphi and their properties and how to change it. When you create a program using the Delphi IDE, you will work with two different views of the application. These are:

- design view
- code view.

You can change from design view to code view or vice versa by clicking on the code or design tabs at the bottom of the IDE.

In this unit, you will learn more about how these views work.

New words

user interface – the part users see and interact with when using a program

DESIGN

The Designer screen is that part that you use to create a **user interface** for your application. It contains the form component that you use to design a user interface. The form serves as a container for the components (such as labels and buttons) that are used to design a user interface.

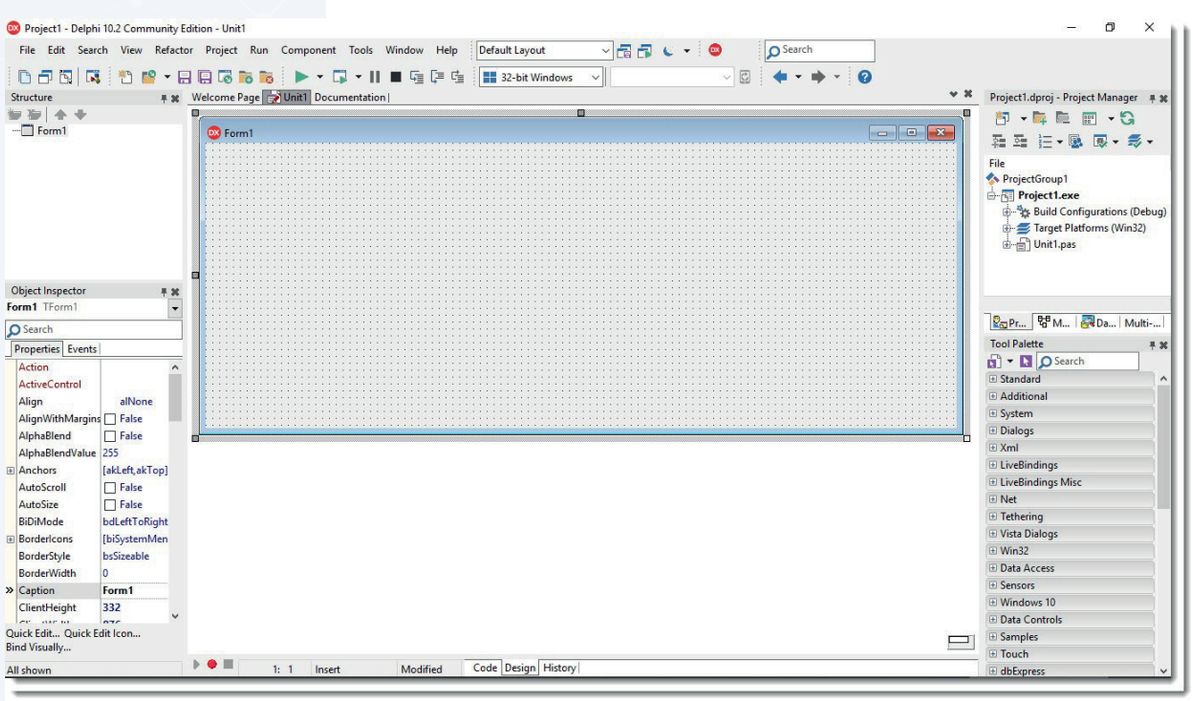


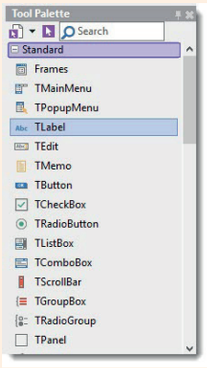
Figure 2.1: *The Design screen is used to create the GUI*

PLACING COMPONENTS ON THE FORM

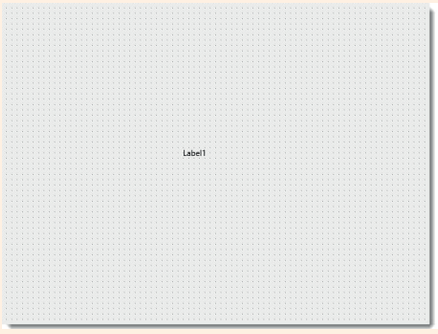
Work through the following example to help you understand how to place different components on your form.

Example 2.2 Creating a project

1. Create a new project named, **HelloDelphi_p.proj**. Save the unit as **HelloDelphi_u**. Rename the form to frmHelloDelphi.
2. Select the *Tool Palette* on the right-hand side of the screen. Open the *Standard* option in the *Tool Palette*.
3. Scroll down the list of components until you find the *TLabel* component.



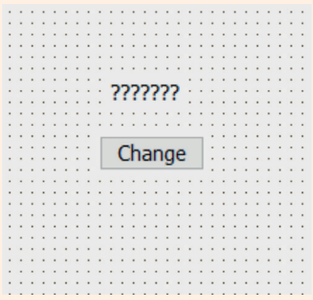
4. Drag the *TLabel* component from the *Tool Palette* onto the *Form*.



5. You should now see the text, Label1, in the *Design* screen.
6. Press the CTRL + S shortcut key on your keyboard to save your project.
7. Drag the *TButton* component from the *Tool Palette* onto the form below the label.
8. Change the properties of the label and button as follows:

COMPONENT	PROPERTY	NEW VALUE
Label	Name	lblMessage
	Caption	???????
Button	Name	btnChange
	Caption	Change

Your form should now look like this:



Take note

For Delphi to understand your code, it needs to be written using the Delphi programming language and following the Delphi language rules.

Take note

Make sure to enter the code exactly as it is shown in the example. This includes the colon before the equals sign and the semicolon at the end of the line.

Take note

A statement ends with a semicolon (;). The semicolon tells Delphi that it is the end of a statement – in this instance the assignment statement that will assign text to the label caption.

CODE

In the previous section, you saw how the Design screen could be used to create a user interface for your application. When we change a property using the object inspector, we say it is changed during design time. Sometimes we need to change properties when the program is executed (that is, during runtime).

The design screen/object inspector cannot be used to change properties during runtime. We need to use a code editor to create the code for it.

To see how a simple user interface is created, work through the following example.

Example 2.3

1. Double click the [Change] button. The codeview will open and you will see the following:

```
procedure TfrmHelloDelphi.btnChangeClick(Sender:
TObject);
begin

end;
```

2. Add the following line of code between the begin and end lines:

```
  lblMessage.Caption := 'Hello, Delphi!';
```

3. Save the project.

Take note

Label name — Label property — Text to change caption

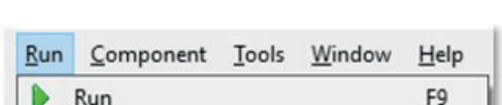
```
lblMessage.Caption := 'Hello, Delphi!';
```

Assignment operator in Delphi

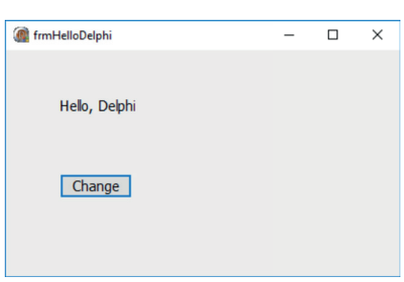
The assignment statement assigns the text/value on the right of the := to the left (the Caption of the Label)

RUNNING THE PROJECT

1. To run the project, click the [Run] button from the menu bar.



2. Click the [Change] button. The following screen displays:



3. The project is now in ***runtime*** and the code was executed to change the caption of the label.
4. To close runtime mode and go back to design view or code view, click on the ***Close icon*** ×.

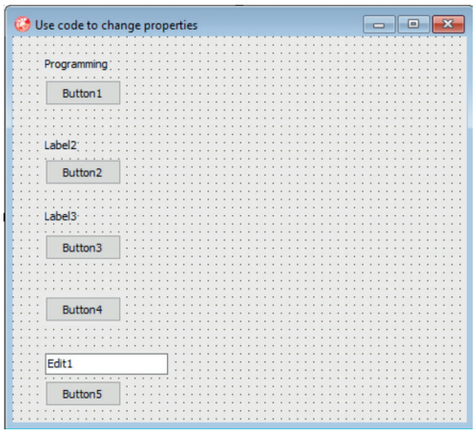
Activity 2.3 Individual Activity

2.3.1 Add code to the [Change] button for the HelloDelphi program as follows:

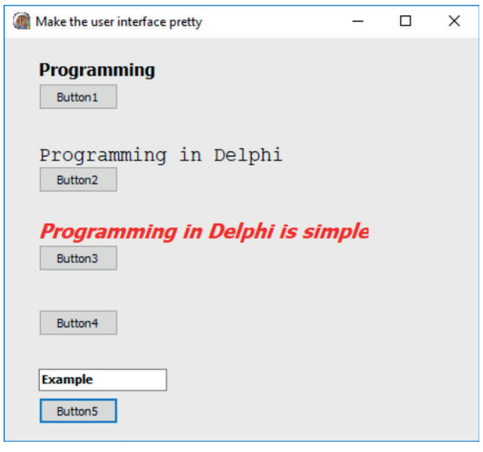
```
procedure TFrmHelloDelphi.btnChangeClick(Sender:
TObject);
begin
  lblMessage.Caption := 'Hello, Delphi!';
  frmHelloDelphi.Caption :=  'Hello';
  lblMessage.Caption := 'Hello, World';     //add this
  lblMessage.Font.Size := 48;               //code
  lblMessage.Font.Style := [fsItalic];
end;
```

2.3.2 For each of the new instructions you added, describe what happened. Hint: Change more than one font style by separating them with commas. For example, [fsBold, fsUnderline].

2.3.3 Open the project **CodeProperties_p** provided in the 02 – Code Properties folder. The following components appear on the form in design view:



Use code so that each button will change the labels above them as shown below. Button 4 changes the form's caption shown below. Hint: Use the Font.Color property to change the colour of the font.



2.3.4 After the program is run and all the buttons have been clicked, the screen above must be displayed.

Take note

The font style is placed between square brackets and starts with 'fs'. For example [fsItalic].

UNIT

2.4 Events

Delphi is an event-driven programming language. This means that all the code inside your program must be activated by an event (for example, when you clicked the [Change] button in the previous unit, you triggered an OnClick_() event). In this unit you will learn more about events.

CREATING AN EVENT

An event triggers when the state of an object or component changes. This means an event occurs when a user interacts with a component. With an OnClick_() event, the state of the object changes from 'not clicked' to 'clicked' when you click on the BUTTON. Any code executed as a result of an event is handled by an **event handler**.

For example:

Indicate an OnClick event on btnChange

```
procedure TfrmHelloWorld.btnChangeClick(Sender: TObject);
begin
  lblHelloWorld.Text := 'Hello, Delphi!';
end;

end.
```

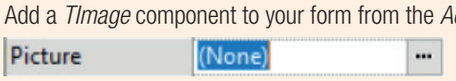
Event handler for btnChange. When the button is clicked, the code within the event handler is executed

The following example shows how an event can be added to a program.

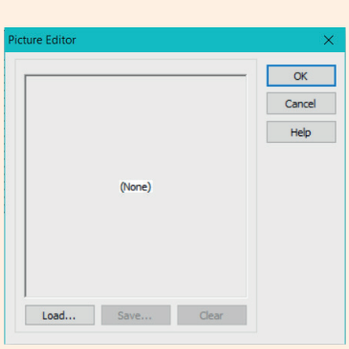
Example 2.4 Disappearing cats user interface

Let's create an application that shows a cute cat picture with two buttons. When the first button is clicked, the cat picture will disappear. When the second button is clicked, the cat will appear again.

1. Create a new folder.
2. Create a new project.
3. Save the unit file in the folder you created as **DisappearingCats_u**.
4. Save the project in the folder you created as **DisappearingCats_p**.
5. Change the form's name to "frmDisappearingCats".
6. Change the form's caption to "Disappearing Cats".
7. Change the form's height to 508 and the form's width to 668.
8. Add a *TImage* component to your form from the *Additional list* in the *Tools Palette*:

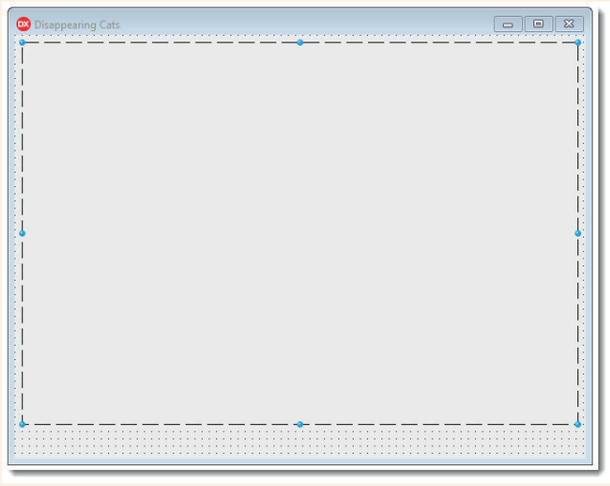


In the object Inspector, look for the Picture property, then click on the ... . This will open the picture editor. Click the [Load] button *to load a picture from your computer (navigate to the folder where the picture file is stored.)*



Example 2.4 Disappearing cats user interface *continued*

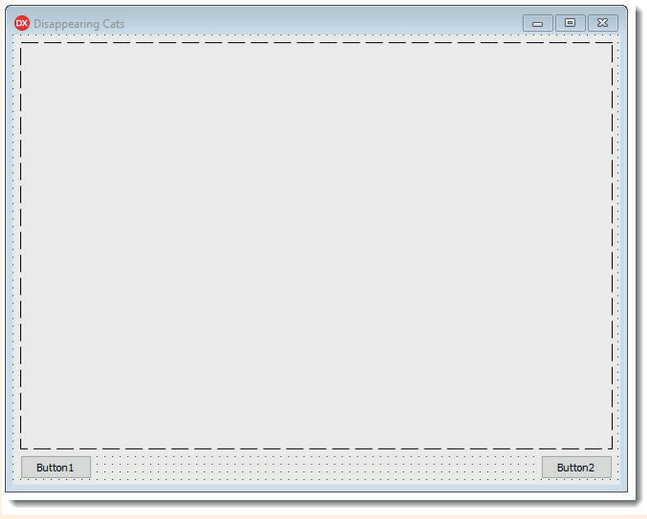
9. Place the image in the top-left corner of the form.
10. By dragging the small blue circles, increase the size of the image until it fills most of the form. Leave enough space underneath the form to add two buttons.



Take note

You can add a component to the *Form* by double-clicking on the component in the *Tool Palette*.

11. Change the *Name* property of the image to "imgCat".
12. Add two *TButton* components (from the *Standard* list) on the form.
13. Place the buttons at the bottom-left and bottom-right of the form.

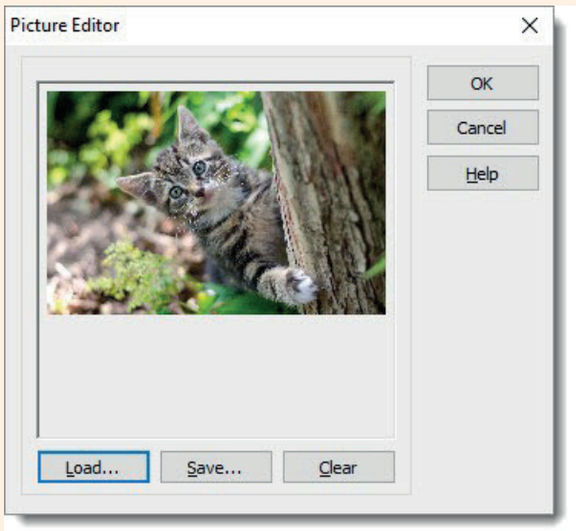


14. Change the names and captions of the buttons as follows:

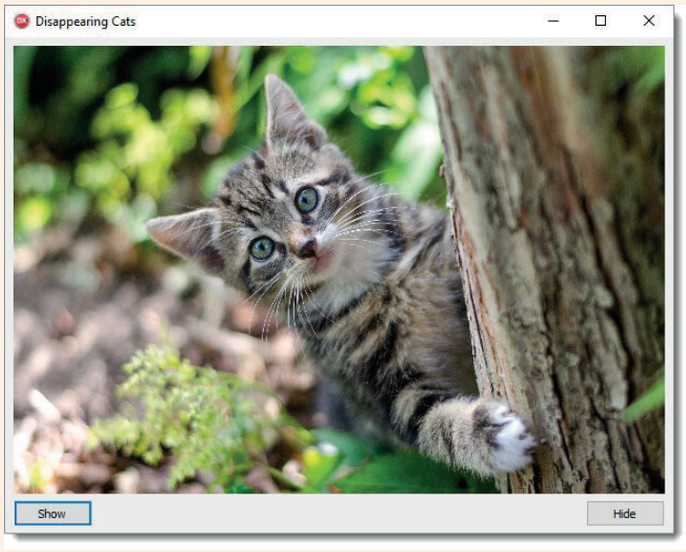
Name	Caption
btnShow	Show
btnHide	Hide

Example 2.4 Disappearing cats user interface *continued*

15. Finally, select the image component and click on the three dots button next to the Picture property in the Object Inspector
16. Click on the *Load*. You will need to select the picture from its file location and click on *Open*.



17. Click *OK*.
18. Set the *Stretch* property of your picture to True.
19. Save and run your application.



Well done! You have now created the user interface for your application.

METHODS

Components have properties and methods. You have already learnt how to change the properties of a component. Methods are actions used to change the behaviour of a component.

For example: To hide or show this cat image imgCat we use the Hide and Show methods of imgCat:

- imgCat.hide
- imgCat.show

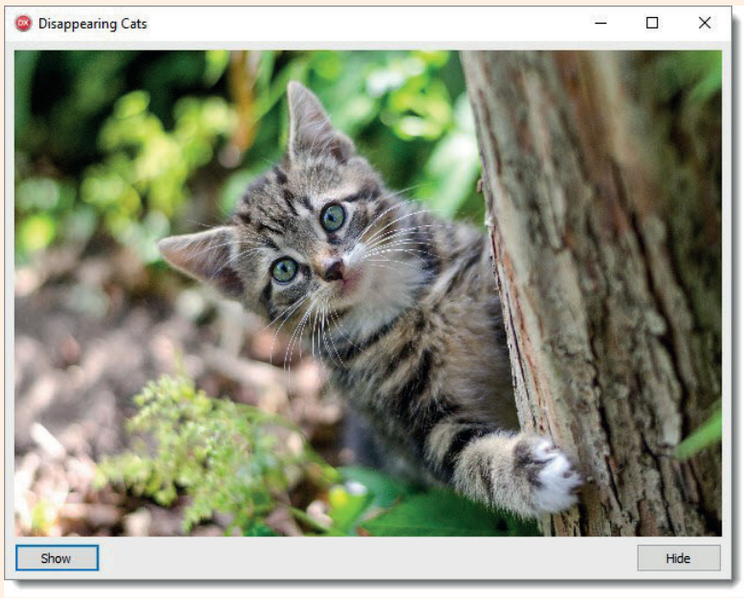
Example 2.5 Coding the hide and show methods for the cat image

Now that you have created the user interface for your 'Disappearing Cats' application, you can create events for the two events.

1. Open the DisappearingCat project.
2. Create an onClick event for the [Show] button. When this button is clicked, the image must appear.
3. Create an onClick event for the [Hide] button. When this button is clicked the image will disappear.
4. Insert the code as shown in the event handlers below

```
disappearingCats_u
TfrmDisappearingCats    TfrmDisappearingCats.btnShowClick
30  procedure TfrmDisappearingCats.btnHideClick(
    begin
      imgCat.Hide;
    end;

    procedure TfrmDisappearingCats.btnShowClick(
    begin
37    imgCat.Show;
    end;

40  end.
```

5. Save and run your application. You should now be able to hide and show the cat, by clicking on the different buttons.



Congratulations, you have just created an application that uses two OnClick events to show or hide an image!

Take note

You can also press the CTRL + SHIFT + F9 shortcut key to run your application.

Because Delphi is an event-driven language, all projects will use events, combined with lines of code, to add interactivity to the applications.

Activity 2.4

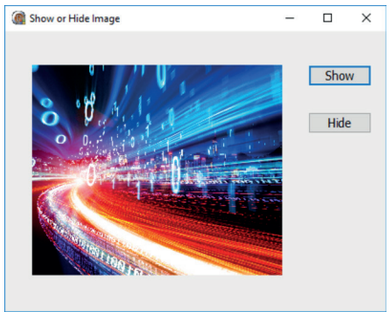
2.4.1 Read the following statement, and in your own words (using an example), explain what you understand by it.

An event triggers when the state of an object or component changes. With an OnClick_ () event, the state of the object changes from 'not clicked' to 'clicked' when you click on the BUTTON.

2.4.2 What is an event handler?

2.4.3 Complete the sentence by filling in the missing words:
A _____ ends with a semicolon (;). The _____ tells Delphi that it is the end of a statement.

2.4.4 Create a new project named **ImageShowHide_p**. Do the following:

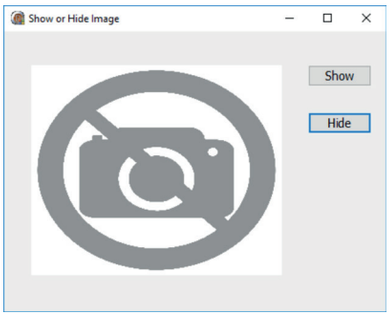


a. Add two buttons.

b. Add two Image components: one on top of another.

c. Load the Book Image for the first Image component, and the NoCamera Image for the second Image component. The images have been provided. Remember to set the Stretch property so that the pictures fit into the Image component.

d. Create onClick_ () events for the two buttons:



i. when button 1 is clicked, the Book image is displayed and the NoCamera image is hidden.

ii. when button 2 is clicked, the NoCamera image is displayed and the Book image is hidden

e. Save and run the project.

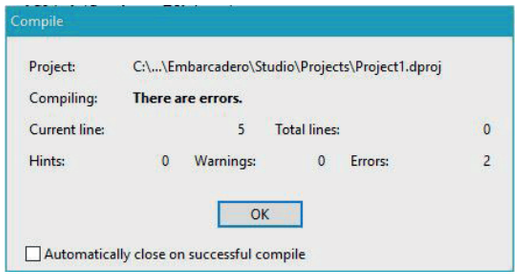
UNIT

2.5 Syntax

Syntax refers to the specific rules of a language. For example, in English, one must follow grammatical rules when constructing a sentence. If these rules are not followed, the sentence will not make sense nor will it be understood.

The same is true in programming languages. Every programming language has its own sets of rules for creating instructions for the computer. If you break even the smallest rule, you will get a syntax error and the program will not run!

Delphi uses a compiler. A compiler checks the entire program for syntax errors before it executes. The program will only execute when the entire program is free of syntax errors. If there are syntax errors, the program will not execute, and the Compile dialog box will display the number of syntax errors. This is shown in the image below:



Did you know

In the examples from the table, the two forward slashes (//) show that the text after the slashes are a comment. These comments can help you to understand the program but are ignored by Delphi.

The following lists a few of the most important Delphi syntax rules.

RULE	EXAMPLE
Every statement should end with a semicolon.	`lblHelloWorld.Caption := 'Hello, Delphi!';`
Comments (which are ignored by Delphi) can be added to the code by starting the comment with two forward slashes (//). Comments are used to explain code.	`btnChange.Width := 25;` `// This will change the width of the button.`

The 'begin' and 'end;' keywords are used to group multiple lines of code together.
Generally, for every begin, there must be an end.
	`begin`
	`  btnChange.Width := 25;`
	`  btnChange.Height := 50;`
	`end;`

The final 'end.' statement in a program must end with a full stop.	`unit u_basicCalculator;`
	
	`begin`
	`  btnChange.Height := 50;`
	`end;`
	
	`end.`

An object's properties or methods are referenced using dot notation after the object's name.	`lblHelloWorld.Caption := 'Hello, Delphi!';`
Delphi's variables or objects are not case-sensitive, so "lblText" is the same as "LBLTEXT" or "lbltext".	`LBLHelloWorld.Caption := 'Hello, Delphi!';`

Some of these rules and the concepts behind them will be covered in greater detail in later chapters.

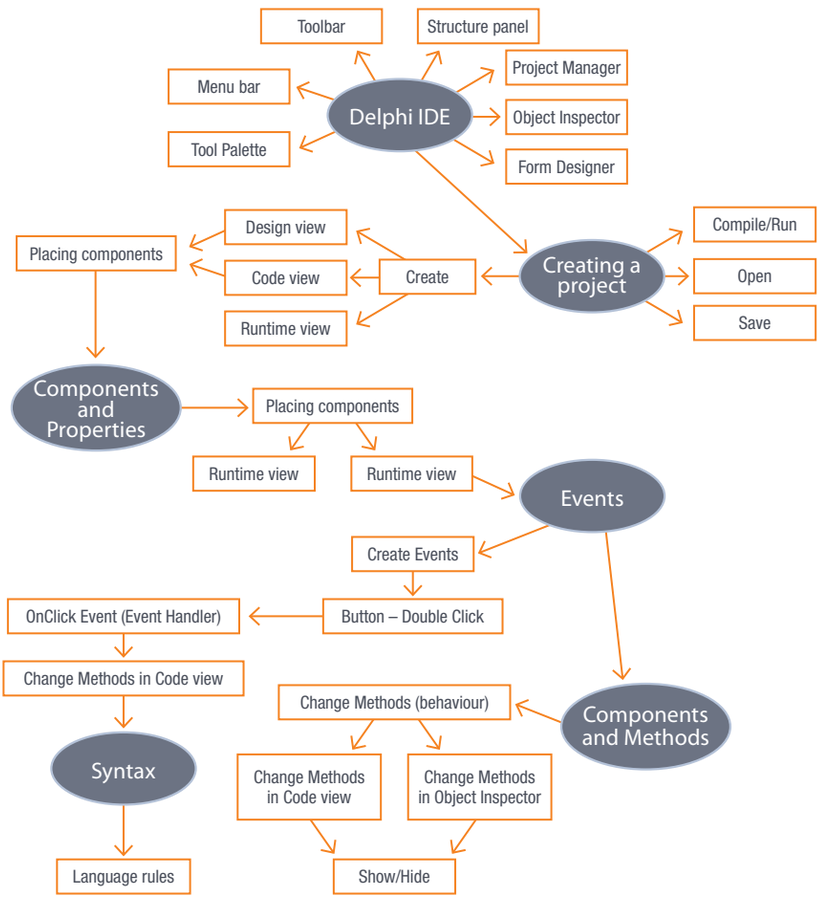
Activity 2.5

2.5.1 Looking at the code from your previous three applications, write down examples of the following syntax:

- **a.** A statement ending in a semicolon from the DisappearingCats application.
- **b.** A statement using the assignment operator from the HelloDelphi application.
- **c.** Begin and end commands from the HelloDelphi application.
- **d.** Statement referencing an object's properties or methods using the dot notation from the DisappearingCats application.

2.6.2 Open the project called **SyntaxErrors_p** from the 02 – Syntax Errors folder. The program contains several syntax errors. Correct the syntax errors and run the program.

Consolidation

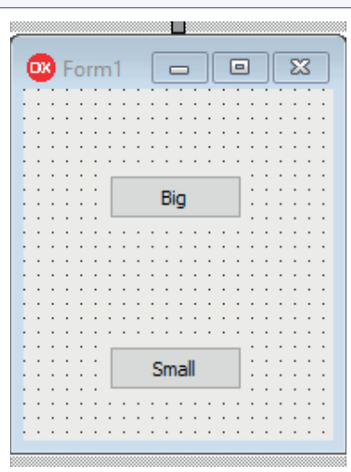


Consolidation activities

Chapter 2: Delphi

1. Explain the difference between an event and an event handler.
2. Explain the difference between design and runtime mode.
3. What does syntax refer to?
4. Explain the difference between a property and a method of a button and provide an example of each one.
5. Create a project called **DisappearingButtons_p**. Add the form **DisappearingButtons_u** with the following components:

COMPONENT	PROPERTY	VALUE
Form1	Name Caption Height Width	frmSize Size 190 190
Button1	Name Caption Visible	btnBig Big True
Button2	Name Caption Visible	btnSmall Small False

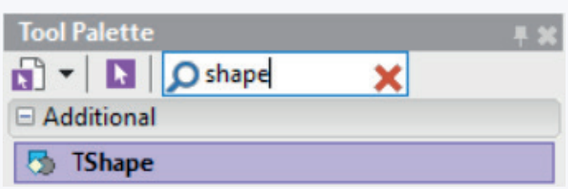


Note: If the visible property is set to False, the button will become invisible when the program is executed.

Using code, do the following:

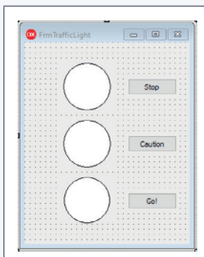
a. Double click on the [Big] button and write code to do the following:
   - Change the Height of the form to 450
   - Change the Width of the form to 250
   - Change the visible property of the [Big] button to False.
   - Change the visible property of the [Small] button to True.

b. Double click on the [Small] button and write code to do the following:
   - Change the Height of the form to 250
   - Change the Width of the form to 150
   - Show the [Small] button.
   - Hide the [Big] button



6. Create a Form to simulate a traffic light.
   Use the following components:

   Form
   - Three circle shapes (TShape component)
     Hint: Change the square to a circle in the object inspector using the Shape property.
   - Three buttons (Stop, Caution, Go).



You could also add a picture component with a picture to the form and add the shapes on top of the picture:

   a. When you click on the [Stop] button, the circle shape at the top must change to red and the colour of the other two shapes (Caution and Go!) must turn to olive and green respectively. Hint: The colour of a shape is changed using the Brush.Color property.
   b. When you click on the [Caution] button, the circle shape in the middle must change to yellow and the colour of the other two shapes (Stop and Go!) must turn to maroon and green respectively.
   c. When you click on the [Go] button, the circle shape at the bottom must change to lime and the colour of the other two shapes (Stop and Caution) must turn to maroon and olive respectively.

   Example:

btnStop is clicked	btnCaution is clicked	btnGo! is clicked
		

7. Use the knowledge and skills that you gained in this chapter and create a Delphi application to simulate a card for a special occasion. You can choose to make a birthday card, Valentine's day card, Mother's day card, invitation to a party, and so on. Refer to the **SpecialOccasion_p** project for some inspiration.

8. Open the project, **FaultyApp_p**.
   The code contains syntax errors. Find the errors and correct all the errors. Once you think that you have corrected all the errors, run to program to see if it compiles. If it does not compile, it means that you have not corrected all the errors. Repeat the process until all errors are corrected and the program compiles correctly, then click the buttons to see what happens.

VARIABLES AND COMPONENTS

CHAPTER 3

CHAPTER UNITS	
Unit 3.1	Data types
Unit 3.2	Variable and component names
Unit 3.3	Declaring variables and components
Unit 3.4	Assigning values to variables
Unit 3.5	Converting data types
Unit 3.6	Errors

Learning outcomes

At the end of this chapter, you should be able to:

- explore the use of variables
- use descriptive variable and component names, as well as correct naming conventions
- assign values to variables
- explore data types such as integers, strings, floats, Boolean
- identify, categorise and fix errors.

INTRODUCTION

Imagine that you had an invisible box that could store information. You could place anything that you needed to keep or remember (like a birthdate or phone number) inside the box. Then, when you need the information, you could simply find the information inside the box.

There are several advantages to using this invisible box:

- The box can store data. You can place data in the box now (when you have access to it) and use it later (when you may need it).
- The box always stays in the same place. This means that, whenever you need to access the data, you know exactly where to look.
- The box only stores one piece of data at a time.
- The data inside the box can change. Even though the box can store only one piece of data at a time, you can replace the old data with new data whenever you need to.
- You can create as many invisible boxes as you need.

In programming, we have an invisible box we use. We refer to this invisible box as a **variable**. A variable refers to a memory location that can hold one item of data at any given time. Each variable in Delphi is given a unique name and can be used to store data. You simply need to refer to the variable's name when you must access the data stored at a particular memory location. Variables can only store one piece of information at a time, but you can change the variable's value as often as you want to.

In this chapter, you will learn the following:

- different data types
- variable naming convention
- variable declarations
- assigning values to variables
- converting certain variable types to another type (**typecasting**)
- understanding and fixing basic programming errors.

UNIT

3.1 Data types

A variable refers to a memory location that can store data of a particular data type. Just like variables in Mathematics, variables in programming can only hold one value at a time. Whenever you create a new variable in Delphi, you must indicate what type of data the variable will store.

This year, you will work with the following five data types:

Take note

Variables in a computer program are similar to 'boxes' where information can be kept, maintained and referred to.

DATA TYPE	DESCRIPTION
String	Strings are made up of a sequence of numbers, letters and symbols. A string variable can contain anything from a single character to multiple characters. Examples of string data: • 'Stefan' • 'P@ssw0rd' • 'True' • 'The brown fox jumps over the fence!' • '123' **NOTE:** • In Delphi, a string must be included between single quotation marks as shown above. • A number within quotes refer to a string. This means that it cannot be used in numerical calculations.
Char	Char represents a single character, such as a letter, symbol or digit. Examples of char data: • 'A' • '*' • '0' **NOTE:** • A digit with single quotation marks will be treated as char, for example, '2'. This means that it cannot be used in numerical calculations.
Integer	An integer can contain any positive or negative number without a decimal point. Integers can be used in numerical calculations. Examples of integer data: • 100 • -5 • 12435
Real	A real number is a positive or negative decimal number. You can perform numerical calculations on real numbers. Examples of real data: • 12.3 • -1.0
Boolean	A Boolean value can only be one of two values, that is, it can either be True or False. This is often used in programs where a specific task is only complete if a specific condition is met. Examples of Boolean data: • True • False **NOTE:** • Boolean values are not included in single quotation marks because they are not strings.

To see how these variable types are used, work through the following guided activity.

Guided Activity 3.1 Variable types

Write down the data type for each of the following:

3.1.1 Kagiso

3.1.2 17

3.1.3 ?

3.1.4 14.3

3.1.5 False

3.1.6 Numbers should be stored as integers

3.1.7 005

3.1.8 18142.23779

3.1.9 True

3.1.10 12i34

Activity 3.1

3.1.1 What is a variable?

a. a number

b. a place in memory that keeps a data value

c. a data value that cannot change

3.1.2 What is a string?

a. a type of variable that hold data values made up of ordered sequences of characters

b. a data type that holds letters

c. a type of variable that hold characters

3.1.3 Determine whether the following statements are true or false. If false, correct the statement to be true.

a. A variable of type string can store a single character.

b. A variable of type integer can store a real number.

c. A variable of type real can store an integer number.

d. A variable of type string can store values that contain special characters such as '*'.

e. A variable of type char can only store a single digit or a single character.

f. A variable of type Boolean can only store one of two values: True or False.

3.1.4 You have been asked to create an application to capture the following information:

IT-Learner Profile

Name of School:					
Name:		Surname:		Initials:	
Grade:		ID No.:		Age:	Date of Birth: (YYYYMMDD)
Gender M/F:	M F	CELL No.:	0	Email address:	

List the variables you will need as well as their types.

Did you know

Only strings and chars are surrounded by quotation marks. Numbers and Boolean values are not.

UNIT

3.2 Variable and component names

In Chapter 2, you may have noticed that we spent time giving each component a descriptive name. This helps makes the code easier to understand.

To ensure that everyone names their variables in the same way, different programming languages have different **naming conventions**. Naming conventions are mainly used so that programmers can understand one another's code. Other than the rules below, you can name your variables whatever you want. Some of the naming conventions are simply a guideline to make the program easier to understand, while others are rules that will stop your program from compiling if they are not followed.

In Delphi the naming rules for variable names are:

- all variable and component names must be unique
- names may not contain spaces
- names may not start with a number, but they can contain a number
- names may not contain any special characters (like exclamation marks) except for underscores (_).

Here are some naming conventions used for variables:

- variable names should describe the data they will contain. For example: variable name: Amount for a variable that will hold monetary data
- names should use **CamelCase**. This means the first word or letter is in lowercase, and each word afterwards starts with an uppercase letter, for example, rAmountPaid
- variable names should start with a single letter prefix describing the data type the variable will hold.

Here are some examples:

DATA TYPE	PREFIX	EXAMPLES
String	s	sName
Char	c	cCode
Integer	i	iNumber
Real	r	rAmountPaid
Boolean	b	bCheckedIn

Naming convention for components:

- component names should describe the task they will perform or the data they will hold
- names should use CamelCase.
- component names should start with a three letters prefix that describes the component type.

Did you know

The Microsoft Windows operating system is made up of 50 million lines of code. Can you imagine trying to figure out how the code works if all the variables have names like 'String1' or 'x'?

Watch out!

Variable names have no impact on the values stored in variables. They simply make it easier for you to understand what the variable should be used for.

Here are some examples:

COMPONENT	PREFIX	EXAMPLES
Form	frm	frmChangeColour
Button	btn	btnChange
label	lbl	lblMessage
Image	img	imgCat

Refer to Annexure C for a detailed list of components and their prefixes.

Did you know

You can use a label component to place a heading, or a description of what should be in a component or even to display output.

Activity 3.2

3.2.1 State whether the following are TRUE or FALSE. If false, motivate why it is not true.

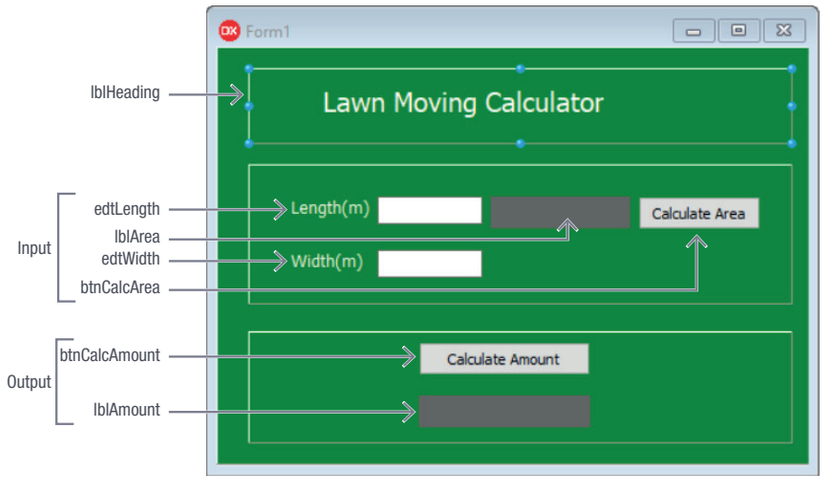
When naming Delphi variables:

a. Variable names may start with a number or a letter but may not contain any special characters.

b. Variable names should never start with a single letter that could describe the type of variable it is.

c. Variable names may only contain lowercase letters and you can include spaces in the name.

3.2.2 Look at the following GUI, and predict what each component will do based on the variable name that has been assigned to it:



UNIT

3.3 Declaring variables and Components

DECLARING VARIABLES

Before you can start using a variable in Delphi, you must tell Delphi to reserve memory space where the variable values can be kept. That is, you need to tell Delphi what the variable's name will be and what type of data you will store in the variable.

The code below shows the syntax for **declaring** a variable.
Variables are declared in the programunder the **keyword** **var**.

```
var                          //Start of section of variables
    VariableName1 : <DataType1>;
    VariableName2, VariableName3, VariableName4 :
<DataType2>;
```

In the example above, you should take note of the following:

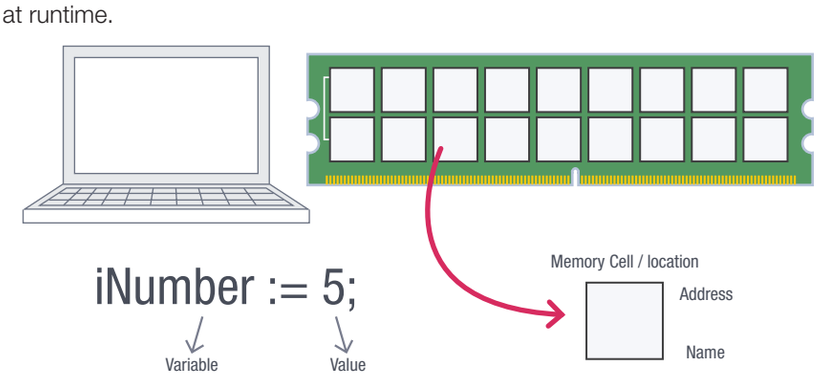
- The var statement is used to tell Delphi that variables will be declared.
- Variables are declared by typing the variable's name, followed by a colon (:) followed by the variable's type, followed by a semicolon (;).
- Variable definitions are indented (i.e. there is a little space before them). This makes your code easier to read.
- Multiple variables of the same type can be declared on one line, as long as they are separated by commas.

The code below shows an example of different types of variables being declared in Delphi.

Declaring a variable

```
var
  rTotal : Real;
  iMinimum, iMaximum : Integer;
  sName : String;
  cGender : Char;
  bValid : Boolean;
```

When the program is executed, memory locations are allocated to the variables at runtime.



New words

keyword – a word with a predefined meaning in a programming language. You cannot use keywords as variable names. It's for this reason that each new variable has to be declared before use.

Did you know

The process of creating a variable by giving it a name and type is known as declaring the variable.

Take note

The value of variables is only stored in your computer's memory. When your program is closed, the variable and its value disappear from your computer's memory.

In the case of the variable declaration above, one memory location is allocated per variable. The number of bytes allocated to a variable depends on its data type.

Take note

Read the section on data representation in your Theory Book about bits and bytes.

DATA TYPE	NUMBER OF BYTES
Integer	4 bytes
Real	8 bytes
String	available memory
Character	2 bytes
Boolean	2 bytes

Guided Activity 3.2 Declaring variables

Using a pen and paper, write down how you would declare the variables needed to store the following information. You need to use descriptive names and decide on a suitable data type for each variable.

3.2.1 0.379

3.2.2 True

3.2.3 'f'

3.2.4 1725

3.2.5 0.657

3.2.6 0152973208

3.2.7 910229 0056 08 3

3.2.8 π

Once a variable has been declared, you can only store the declared or compatible type of data in it. We can store integers in real variables BUT not real values in integer variables. An integer can be represented as a real number, but a real number cannot be represented as an integer. Trying to store the incorrect type of data for a variable will result in an 'Incompatible Type' error.

Guided Activity 3.3 Variables

Study the following variables

```
sName, sSurname : string;
cGender : char;
iAge : integer;
sEmailAddress : string;
sMobileNumber : string;
bPensioner : boolean;
iOrderQuantity : integer;
rPrice : real;
```

3.3.1 Identify different examples of variables from those provided above that meet the following requirements:

- a. A variable that contains a number without a decimal value.
- b. A variable that contains a special character (e.g. #, %).
- c. A variable that contains a number with a decimal value.
- d. A variable name that contains a single character only.
- e. A variable name that contains the value true or false.
- f. A variable name that contains letters only.

3.3.2 Explain why:

- a. sMobileNumber is declared as a string type.
- b. bPensioner is declared as a Boolean type.
- c. rPrice is declared as a real type.

DECLARING COMPONENTS

When you place a component onto a form, Delphi automatically inserts the declarations in the Type section of the code. The example below shows all the components placed on the form. The component name appears on the left-hand side of the colon and the data type on the right-hand side.

```
type
TfrmReport = class (TForm)
   edtName: TEdit;
   edtSubject: TEdit;
   edtMark1: TEdit;
   edtMark2: TEdit;
   lblName: TLabel;
   lblSubject: TLabel;
   lblMark1: TLabel;
   lblMark2: TLabel;
   lblOutput: TLabel;
   lblReport: TLabel;
   lblMarkOutput1: TLabel;
   lblMarkOutput2: TLabel;
   lblMarkAverage: TLabel;
   btnGenerate: TButton;
   pnlInput: TPanel;
   pnlOutput: TPanel;
   lblInput: TLabel;
```

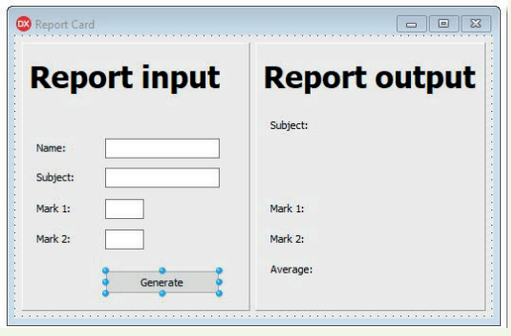
Look at this example to help you understand this.

Example 3.1 Report card variables

Use the report card user interface and identified variables to create the four variables inside the application.

To do this:

1. Open the project stored in your 03 – Report Card folder.
2. Select the [Generate] button in the Designer screen.



3. Double click btnGenerate to do the code.
4. Inside the Code screen, find the procedure called *TfrmReport.btnGenerateClick*.

Example 3.1 Report card variables *continued*

5. Add a few empty lines between the procedure's name and the `begin`.

```
procedure TfrmReport.btnGenerateClick(Sender: TObject);
begin
end;
end.
```

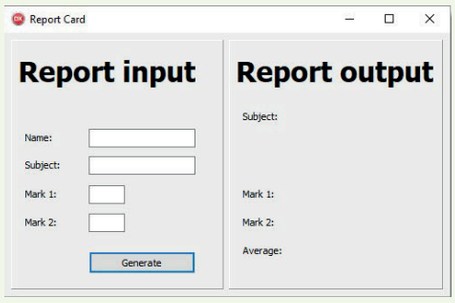
6. Add the var keyword underneath the procedure's name in the space that you have created.
7. Declare the following two string variables: sName and sSubject.
8. Declare the following two integer variables: 'iMark1' and 'iMark2.
   The variable declaration should now look as follows:

```
procedure TfrmReport.btnGenerateClick(Sender: TObject);
var
  sName, sSubject : String;
  iMark1, iMark2 : Integer;
begin
  ….
end;
end.
```

Take note

The two strings are declared on the same line with the variable names separated by a comma. The two integers are also declared on the same line, separated by a comma.

9. Save and run your application. If the variables were declared correctly, the application should open.



Well done! You have just declared your first set of variables in an application.

Activity 3.3

3.3.1 Which statement in Delphi tells it that the statements to follow are variable declarations?

3.3.2 What is the purpose of a variable declaration?

3.3.3 What is required to declare a variable?

3.3.4 Identify the errors in the following variable declarations.

```
Var
  iNumber = Integer;
  sGrade := String;
  bStatus : Boolean
  cDigit = char;
  rNumber, iTotal : Real;
  sGrade : integer;
```

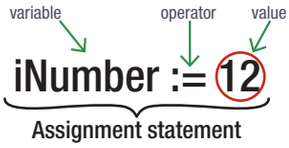
Copy this table and complete it. Briefly describe each error and correct the error.

DECLARATION ERROR	BRIEFLY DESCRIBE THE ERROR	CORRECT DECLARATION STATEMENT

UNIT

3.4 Assigning values to variables

Once you have declared your variables, you can assign values to them using an assignment statement. An assignment statement consists of a variable, an assignment operator and a value.



The table below shows how variables can be assigned in Delphi.

VARIABLE	OPERATOR	VALUE	ASSIGNMENT STATEMENT
iNumber	:=	12;	iNumber := 12;
bStatus	:=	True;	bStatus := True;
sName	:=	'John';	sName := 'John';

NOTE:

- Values are assigned in Delphi using the colon-equals assignment operator (:=).
- The variable name appears on the left-hand side of the assignment operator.
- The values being assigned to the variables appear on the right-hand side of the assignment operator.
- To ensure the value is assigned successfully, the value on the right of the assignment operator must be of the same type as the variable or of a compatible type.
- Once a value has been assigned to a variable, it overwrites/replaces any existing value stored in the variable.

The code snippet below shows how you can assign values to different types of variables.

Assigning values

```
VAR
  sName : string;
  iNumber : integer;
  rAverage : real;
  bPass : Boolean;

begin
  sName := 'Kagiso';

  iNumber := 5;

  rAverage := 67.74;

  bPass := True;
end;
```

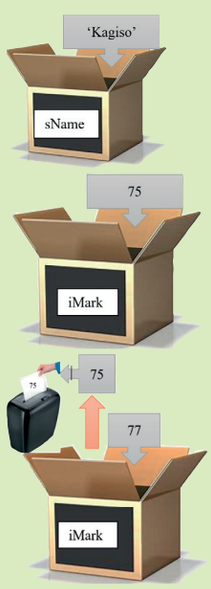
You can also change the data *value* stored in a *variable*, using ***assignment***. This causes the value to be copied into the ***memory location***, overwriting what was in there before.

Watch out!

When a new value is assigned the previous value is overwritten/replaced.

Changing values

```
VAR
  sName : string;
  iMark : integer;

begin
 sName := edit1.text
       //if Kagiso is Entered in edit1
       //sName will have the value Kagiso;

  iMark := 75;




  iMark := 77;
       //The 75 in iMark will be replaced
       //with 77
end;
```



1ST ASSIGNMENT		
Variable	sMyName	(sMyName's content is 'Kagiso')
Value	Kagiso	
Memory Address	0012FC44	(sMyName's address is 0012FC44)

2ND ASSIGNMENT		
Variable	sMyName	(sMyName's content is 'Mary')
New Value for sMyName	Mary	
Memory Address	0012FC44	(sMyName's address is 0012FC44)

Watch out!

A variable is a symbolic name for (or reference to) a value stored in memory. Every variable has a name, a type, a value and a location:

- Name: The name is symbolic and is how the programmer accesses the value of the variable. Every variable must have a unique name.
- Type: The type represents what 'kind' of data is stored within the variable.
- Value: The value represents the contents of the variable. It can change over time. When a new value is assigned to a variable, the existing value is replaced by the new value.
- Location: The memory cell where the value is kept.

In the next example, you will learn how to assign values to the variables in your report card application.

Watch out!

A variable can only store one value at a time. If you assign a new value to it, you will overwrite (or delete) the previous value.

Example 3.2 Assigning report card values

To assign values to the variables in your report card application:

1. Open the project stored in your 03 – Report Card folder.
2. Open the *Code* screen of your application and find the *TfrmReport.btnGenerateClick* procedure.
3. Assign the text entered into the *edtName* component to the sName variable using the code shown below.

Assigning a value

```
sName := edtName.Text;
```

Access the component's property by using the component's name, followed by a full stop, followed by the property. This can be used to both assign a value to a property and to read a value from a property. In the code above, the *edtName* component's text value is assigned to the sName variable.

4. Using similar code, assign the text entered into the *edtSubject* textbox to the sSubject variable.

Assigning a value

```
sSubject := edtSubject.Text;
```

With both the learner's name and the subject stored in a variable, you can now write these values to the output labels. To do this:

5. Assign the value of the sName string to the *lblOutput* caption, as shown below:

Assigning a value to a property

```
lblOutput.Caption := sName;
```

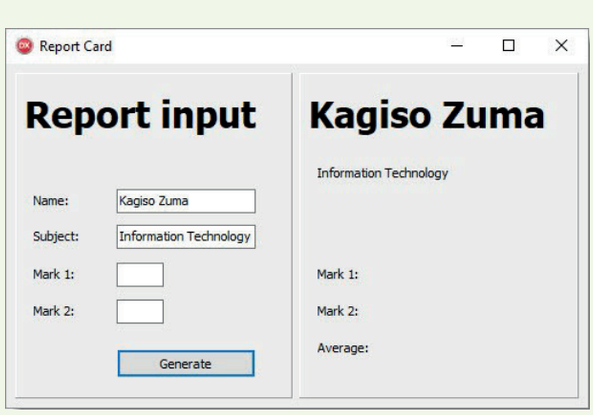
Where the previous code snippets read the value from a component and assigned it to a variable, this code does the reverse. It takes the value stored in the 'sName' variable and assigns it to the output label's caption.

6. Assign the value of the sSubject string to the *lblSubject* caption.

Assigning a value to a property

```
lblSubject.Caption := sSubject;
```

7. Save and run your application. You can now enter values in the textboxes in the *Input* section and write them to the labels in the *Output* section by pressing the [Generate] button.



Congratulations, you have just used variables to store information from your application and then display that information on your components. Later in this chapter, you will learn how the string values from the *Mark 1* and *Mark 2* textboxes can be saved as integer variables.

Watch out!

Assigning the value of variable **A** to variable **B** will never affect the value of variable **A**. In this situation, a copy of variable **A**'s value is being assigned to variable **B**.

Take note

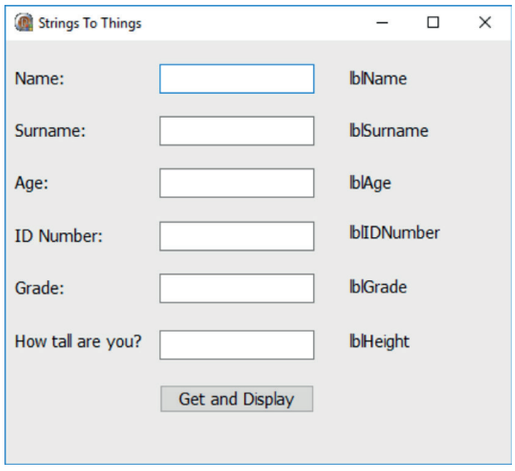
There are only a few things you can do with a variable:
Consider the statement:

iNextNumber := iNumber + 1;

- Create a variable (using the correct naming convention).
- Put some value into the variable using an assignment statement, for example, iNextNumber;
- Use the value of the variable by simply writing the name of the variable, for example, iNumber.

Activity 3.4

Open the Delphi program **StringsToThings_p** from the 03 – StringsToThings folder and write code for *btnGetAndDisplay* that will display the information entered through each edit component on the corresponding label component.



UNIT

3.5 Converting data types

EDIT COMPONENT

In Chapter 2 you used the *Edit* component to enter data. The *Edit* component displays text that can be entered at runtime or design time. The *Edit* component, like all the other components that you studied previously, has properties as well. You can change the properties, such as font, text, maxlength, readOnly and PasswordChar of an *Edit* component.

In the illustration above there are two components. These are a *Label* and an *Edit* component. The *Label* prompts the user with a meaningful message of what must be entered in the *Edit* component. Without the *Label* prompt, the user will be left guessing what they need to enter in the *Edit* component.

The code to read data from an *Edit* component and store it in a variable is:
sName := edtName.text;

Take note

The text property of the *Edit* box is used to read text from the *Edit* box and is assigned to a string variable, sName.

CONVERTING DATA TYPES

When data or values are read from an *Edit* component, it is always of type string. This poses a problem when you want to enter numeric values that will be used in calculations. For example, if you enter two test marks using *Edit* components, and you want to find the average of these two test marks, you will have to convert these values from string to integers.

To convert a string to an integer:

```
sMark := edtMark.text;
iMark := StrToInt(sMark);
lblMessage.caption := IntToStr(iMark);
```

Take note

- The data is read from the edit component, edtMark, and assigned to the variable, sMark.
- The string value in the sMark variable is converted to integer using the StrToInt function and the value is assigned to the integer variable, iMark.
- To display the integer value in a label, you need to convert the integer value to a string before it can be assigned to the caption property of the label. The conversion from integer to string is achieved using the IntToStr function.
- You can combine statements 1 and 2 into one statement as follows: iMark := StrToInt(edtMark.text).

You can also convert between different data types. This conversion process is called **casting**.

To convert (or cast) text to numbers and numbers to text, you can use the following Delphi functions.

CONVERSION FUNCTION	DESCRIPTION	EXAMPLE/S
StrToInt	Converts a string value to an integer value.	iNum := StrToInt('500'); iAge := StrToInt(edtAge.text);
StrToFloat	Converts a string to a real number	rAmount := StrToFloat('500.14'); rLength := StrToFloat(edtLength.text);

IntToStr	Converts an integer value to a string value	sPostalCode := IntToStr(4450);
		iEmployeeNumber := 1254672;
		sEmpNum := IntToStr(iEmployeeNumber);

FloatToStr	Converts a real value to a string value	rAverage := (125.56+45.23)/2 sAverage := FloatToStr(rAverage);

Did you know

Float is short for floating point numbers. Real numbers are floating point numbers.

To see how these commands are used in practice, work through the following guided activity.

Guided Activity 3.4 Converting values to numbers

3.4.1 For each of the functions below, write down the variable's value and type.

a. x := FloatToStr(7321.52);

b. y := StrToFloat('121.3');

c. z := StrToInt('15');

d. t := IntToStr(112);

3.4.2 Indicate whether each of the following functions has the correct or incorrect input. If the input is incorrect, provide a reason.

a. StrToInt('1250');

b. StrToFloat('3');

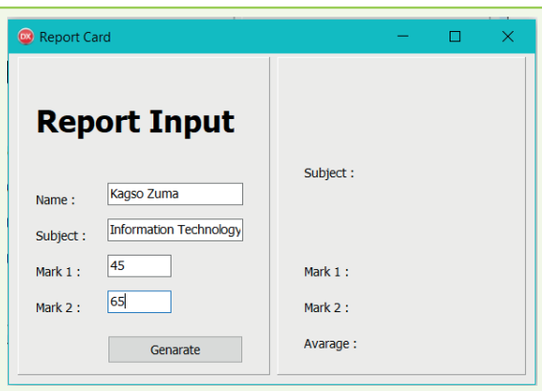
c. FloatToStr('10.5');

d. IntToStr('321');

Example 3.3 Report card casting

A user must enter the name, subject and two marks for a learner and calculate the average of the two marks. Display the subject name, marks and average as a term report.

1. Open the project, **ReportCard_p.dproj** that you worked on in Example 3.2.



Example 3.3 Report card casting *continued*

2. Double click on the [Generate] button to open the Code screen.
3. Assign the text entered into the *edtMark1* component to the *iMark1* integer variable using the code shown below.

Assigning a value

```
iMark1 := StrToInt(edtMark1.Text);
```

Looking at the right-side of this code, you start by reading the text stored in the *edtMark1* component. Since the text stored in these textboxes are always strings, you need to use the StrToInt function to convert the string to an integer. Once it has been converted, you can assign the integer value to the iMark1 integer variable.

4. Assign the text entered into the *edtMark2* component to the iMark2 integer variable.

```
iMark2 := StrToInt(edtMark2.Text);
```

5. Calculate the average of the two test marks. Remember your BODMAS rules.
6. To display the marks in the *lblMark1* and *lblMark2* labels; add the following lines of code:

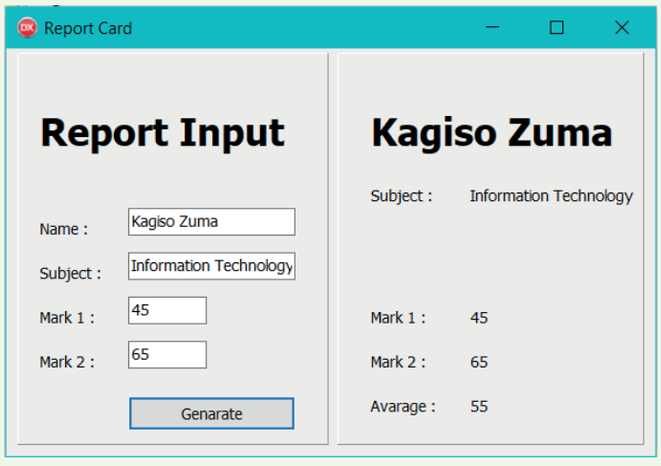
```
  lblMark1.Caption := IntToStr(iMark1);
  lblMark2.Caption := IntToStr(iMark2);
```

In first line the integer variable iMark1 is converted to a string value and assigned to the caption of the label.

7. To display the average on the label *lblAverage*, convert the real average to string:

```
lblAverage.Caption := FloatToStr(rAverage);
```

8. Save and run the application



Take note

When you assign something to a variable, it will delete the content that is already in there, and put only the new content in. You need to place what you want to use later in another variable, called sPlaceholder in this case, to use it later.

Guided Activity 3.5 Name and Surname Swap application

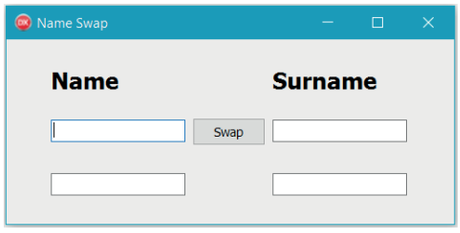
The name and surname have been stored in the wrong order. The algorithm to swap the name and surname is as follows:

- Store the text entered in the name component in a variable, sName
- Store the text enter in the surname component in a variable, sSurname
- Declare a third variable, sPlaceholder
- Move the value of sName to sPlaceholder
- Move the value of sSurname to sName
- Move the value of sPlaceholder to sSurname
- Display the values of sName and sSurname on *edtName* and *lblSurname*

Now that you have a Name and Surname swap algorithm, you can implement it in a program.

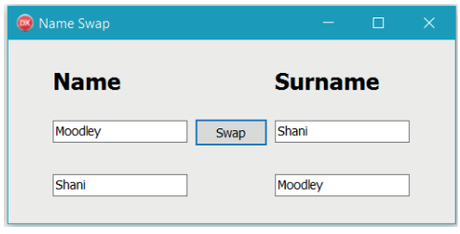
3.5.1 Create a project **NameSwap_p**.

3.5.2 Create the interface as shown below:



3.5.3 Create an OnClick event for the [Swap] button to swap the name and surname.

3.5.4 Save and run the project.

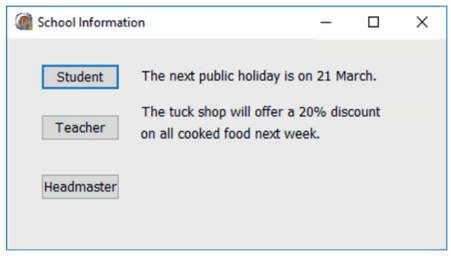


Activity 3.5

Create the following short Delphi programs that will help you practice what you just learned:

3.5.1 Program 1:

Create a project called **School_p**. At school, you, your teachers and your principal may all be interested in different information. For this program, you need to create an application that stores the followingsix pieces of information usingvariables:



- The next public holiday is on 21 March.
- In the previous year, 99% of learners passed matric.
- This year's exam dates start in November.
- In mathematics, learners struggle the most to understand the laws of exponents.
- The tuck shop will offer a 20% discount on all cooked food next week.
- The school is currently ranked as the third best school for academics in the province.

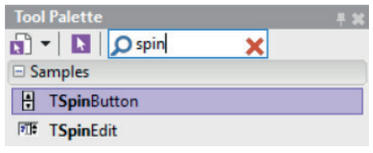
Activity 3.5 *continued*

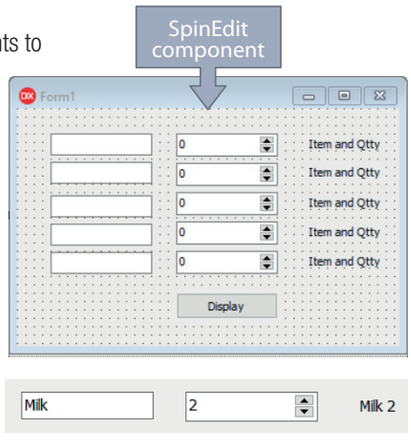
Once this information has been stored in variables, allow the user to click on one of three buttons: Student, Teacher or Headmaster. Depending on the button they click, display two pieces of information that would be useful or interesting to that person.

3.5.2 Program 2:

Open the Delphi program **ShoppingListBasket_p** provided in the 03 – Shopping List folder.

The program uses TSpinEdit components to select the quantity of the item entered in the edit component





The user enters the shopping list item, e.g. milk, then selects the quantity from the corresponding SpinEdit component. When all the items are entered and their quantities are selected, the item is displayed on the corresponding label component once the [Display] button is clicked, e.g.

Write the code for the [Display] button that will display the item and quantity on the [Label] button when the user clicks the [Display] button.

Take note

To access the value from the SpinEdit you must use its Value property. The Value property contains an Integer value. Look at the following code:

```
iItem1 := sedQuantity1.Value;
```

To add the item and the quantity to the [Label] button, use the following code:

```
lblItem1.Caption := sItem1 + ' ' + IntToStr(iItem1);
```

The single quotes contain a space. It will ensure that a space appears between the item name and the quantity value.

Take note

The value property selected from the SpinEdit is an integer value. If you want to display this value on the *Label* component, you need to convert it to a string type first as the caption of the label is a string.

UNIT

3.6 Errors

Programming errors can generally be grouped into three categories:

- syntax errors
- runtime errors
- logic errors.

Since different errors require different solutions, identifying the type of error you are encountering is an important first step in solving the problem. This unit will look at each of these errors in more detail.

SYNTAX ERRORS

Syntax refers to the form and structure of the programming language, which defines how the symbols and keywords must be combined to format a statement. In Delphi, the compiler checks your code against this rule. If the compiler does not understand the statement, then a syntax error is reported.

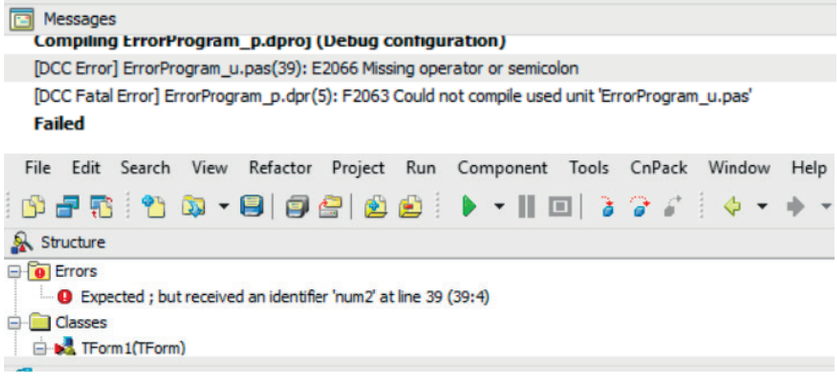
Common syntax errors include:

- leaving out a semicolon at the end of a statement
- we use a 'begin …end' block when grouping statements. Leaving out either the begin or the end will result in a syntax error
- leaving out the command, var, when declaring variables
- assigning a variable using the equals to sign (=) instead of the assignment operator (:=)
- misspelling variable names
- using keywords as variable names
- using variables that have not been declared
- not surrounding string values with the single quote marks.

TOP TEN DISASTROUS SOFTWARE BUGS

https://www.youtube.com/watch?v=AGI371ht1N8

For syntax errors, the IDE will generally warn you that you have made a mistake and indicate the line in which the mistake occurs. Delphi will provide you with information about the error in the Structure panel at the top left or the Messages panel at the bottom of the Code screen. You can use the information provided to correct the errors.

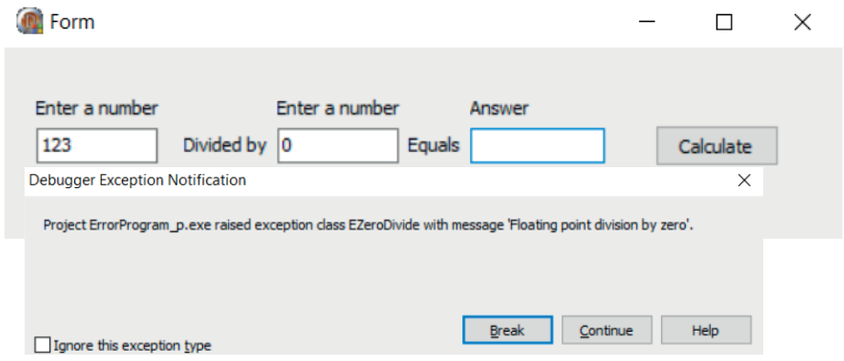


RUNTIME ERRORS

The second type of error is known as a runtime error. These errors occur when the program is running. For example, when division is taking place, the user enters a zero for the divisor, the computer will immediately give a runtime error because division by zero is inadmissible. In Delphi, a runtime error will display as follows:

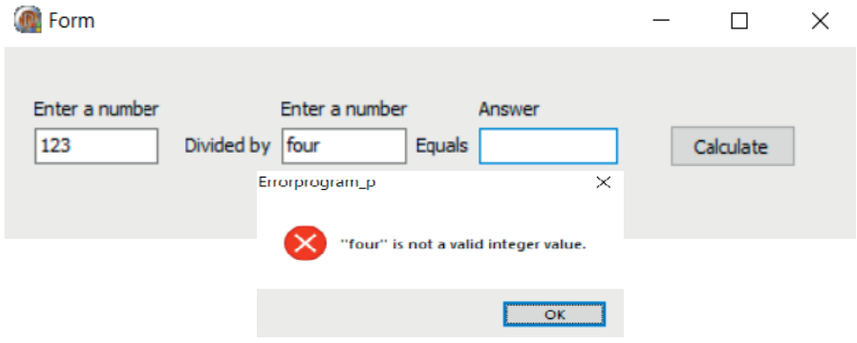
```
Runtime error nnn at xxxxxxxx
```

where nnn is the runtime error number, and xxxxxxxx is the runtime error address.



Runtime errors occur when you ask your program to do a task that is either impossible or is impossible under certain circumstances. The program will then terminate (shut down).

Another example of a runtime error occurs when a program expects an integer as input, but the user enters a real number. The error only occurs when the function StrToInt is executed.



While creating programs this year, you may run into many different runtime errors. One way to solve a runtime error is to **step through** your program by seeing how the value in variables changes with each step of the application. In this way you can usually identify the point at which a mistake was made. You can also search for the error code online to see if it helps you to identify the cause of the error. Later in the year, you will be taught how to step through a program.

LOGIC ERRORS

The final, and most difficult type of error to solve is a logic error. These errors occur when there are errors in the logic of the program or algorithm. The program compiles and runs correctly but produces the incorrect results.

Guided Activity 3.6 Identifying logic errors

The two snippets of code below are an attempt to swap a person's name and surname.

SNIPPET ONE

```
SNAME := edtName.text;
SSurname := edtSurname.text;
SPlaceholder := sName;
SName := sSurname;
SSurname := sName;
```

SNIPPET TWO

```
SNAME := edtName.text;
SSurname := edtSurname.text;
SPlaceholder := sName;
SName := sSurname;
SSurname := sPlaceholder;
```

Work through both program snippets and answer the following questions.

3.6.1 Which one of the code snippets contains an error?

3.6.2 Explain the error

Copy and use the following table to record your findings for each program snippet:

LINE	COMPONENT NAME	VARIABLE NAME	VARIABLE VALUE
1	EdtName.text	SName	
2			
3			
4			
5			

Activity 3.6

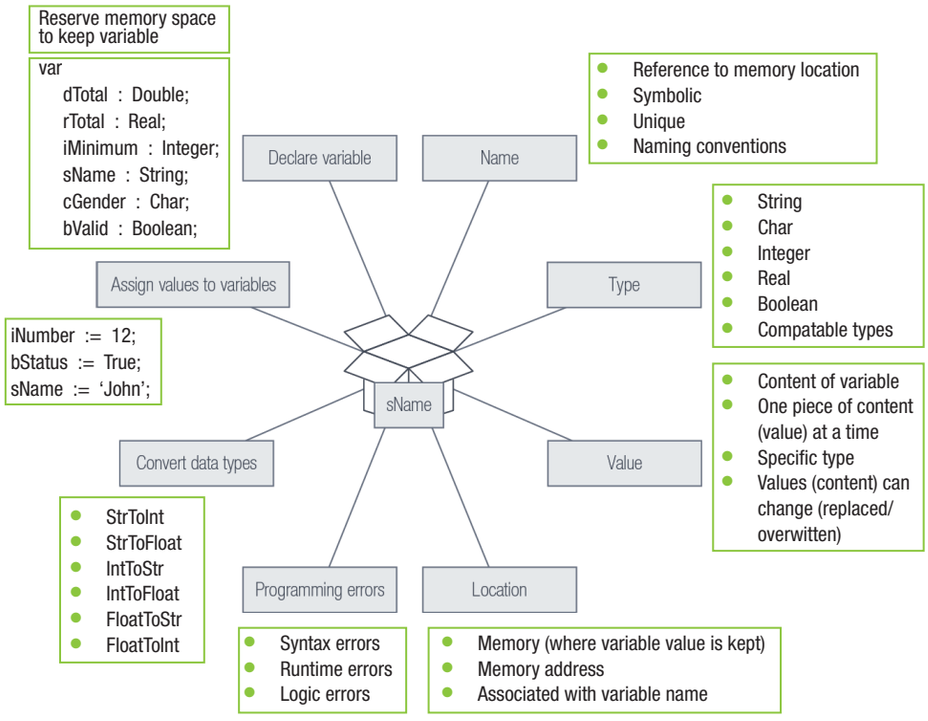
3.6.1 Your teacher will give you three small programs. The programs can be found in the 03 – Error 1, 03 – Error 2 and 03 – Error 3 folders. For each program:

a. find and fix any coding errors that you find in the programs.

b. write down each error, provide a short explanation of the error, the type of error and the corrected code.

ERROR	EXPLANATION OF ERROR	TYPE OF ERROR	CORRECTED CODE STATEMENT

Consolidation



Consolidation activities

Chapter 3: Variables And Components

1. In your own words, give a definition for the word 'variable' as it relates to computer programming.
2. List five Delphi naming rules and conventions for variables.
3. In your own words, what is the difference between an integer and real variable types?
4. What is the difference between syntax, runtime and logic errors.
5. What is the purpose of a variable name?
6. Describe the relationship between the variable name and the memory location.
7. Explain the purpose of converting data types.
8. Study the following code:

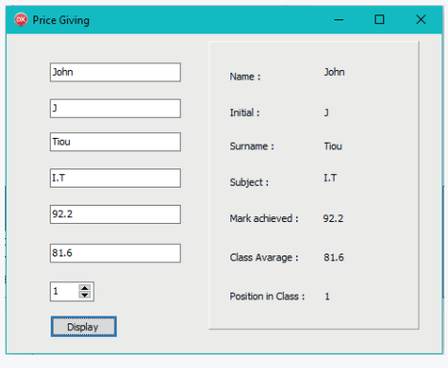
```
Var
  sName, sBirthDate : string;
  iAge : Integer;
  rHeight : real;
  cGender, cGrade : char;
  bSACitizen : Boolean;
Begin
  sName = 'Lerato';
  cGender := 'Female';
  iAge := 15
  rHeight := 1.66 m;
  sBirthDate := 12-09-2003;
  iAge := rHeight;
  bSACitizen := Yes;
  cGrade := '9';
End;
```

Identify the errors in the code. Briefly describe each error and correct the error using the following table:

DECLARATION ERROR	BRIEFLY DESCRIBE THE ERROR	CORRECT DECLARATION STATEMENT

9. Study the following code and answer the questions that follow:

```
procedure TfrmPrizes.btnDisplayClick(Sender: TObject);
var
  sName, sSurname, sSubject:string;
  cInitial : char;
  rMarkAchieved, rClassAvg : real;
  iPosition : integer;
begin
 // assign value to the variables
   sName := edtName.Text;
   cInitial := edtInitial.text[1];
   sSurname := edtSurname.Text;
   sSubject := edtSubject.Text;
   rMarkAchieved := StrToFloat(edtHighMark.Text);
   rClassAvg := StrToFloat(edtClassAverage.text);
   iPosition := spdPlace.Value;
 // display the values from the variables
   lblName.Caption := sName;
   lblInitial.Caption := cInitial;
   lblSurname.Caption := sSurname;
   lblSubject.Caption := sSubject;
   lblMark.Caption := strToFloat(rMarkAchieved);
   lblAvg.Caption := floattostr(rClassAvg);
   lblPlace.Caption := inttostr(iPosition);
end;
```



Did you know

```
cInitial := edtInitial.text[1];
```

The [1] is the command to Delphi to use only the first character of the values in edtInitial.text. It ensures that only ONE character is written to the variable. If there are more than 1, you will get an error message.

a. What data type was used to declare the sName variable?

b. What data type was used to declare the rClassAvg variable?

c. What data type was used to declare the cInitial variable?

d. Why would the line

```
lblMark.Caption := strToFloat(rMarkAchieved);
```

cause an error?

e. What is the purpose of 'StrToFloat' in the following code?

```
rClassAvg := StrToFloat(edtClassAverage.text);
```

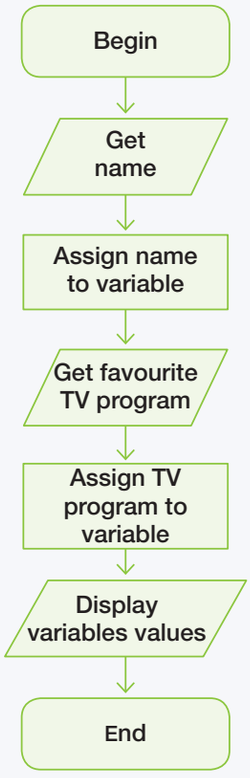
f. Why would the code below be incorrect?

```
iPosition := StrToInt(spdPlace.Value);
```

g. Copy and complete the table to show the value that is saved in each of the variables when the program is executed.

VARIABLE NAME	DATA TYPE	VALUE
sSurname		
sSubject		
cInitial		
rClassAvg		
iPosition		

10. Create a Delphi project, **Favourite_p**, to implement the following flowchart:



11. Create a Delphi program to do the following:
    - Enter a user name
    - Enter an e-mail address
    - Enter a 4-digit pin number
    - Display the user name
    - Display the e-mail address
    - Ask for confirmation that the e-mail address is correct (Y/N) – user must enter Y or N.

    a. Identify the components you need to use for the 'Input' and the 'Output'

    b. Declare the necessary variables and write the above code.

SOLVING BASIC MATHEMATICAL PROBLEMS USING DELPHI

CHAPTER 4

CHAPTER UNITS

Learning outcomes

At the end of this chapter you should be able to:

- identify, describe and apply different mathematical operators
- use the correct order of operations to solve mathematical problems
- explain and apply mathematical functions such as random, round and square root
- create Delphi programs to solve basic mathematical problems
- apply basic algorithms in programs
- use planning tools to plan solutions to problems before implementation in Delphi
- explain the difference between local and global variables and use them appropriately.

INTRODUCTION

Many mathematical problems can be solved using programming. For example, in game pictured below, the computer will have to:

- calculate the high score obtained by a player
- determine which level a player is on
- keep the score of the current game
- keep a record of the time of the current game.

For the computer program to perform the tasks mentioned above, the program must be able to perform mathematical calculations ranging from basic to more complex ones.

In this chapter, you will learn how to solve basic mathematical problems using Delphi. You will also learn how to use planning tools to assist you when making your applications.

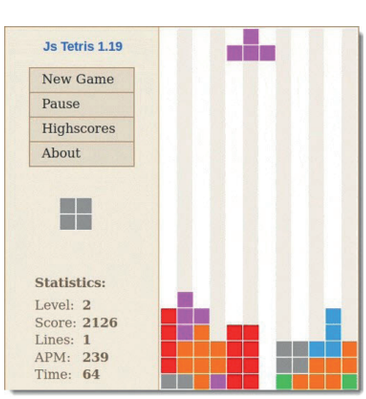


Figure 4.1: *The computer program uses mathematical calculations for even the most basic games (Source: Image by Cezary Tomczak)*

UNIT

4.1 Basic operators

Mathematical operators are the symbols used to tell Delphi to add, subtract, multiply or divide. In this unit you will learn how to use basic mathematical operators to perform calculations with integers and real numbers.

NAME	SYMBOL	DESCRIPTION
Addition	+	Adds two values, for example: `iTotal := 2 + 1; // iTotal = 3`
Subtraction	-	Subtracts the second value from the first. For example: `iTotal := 5 - 3; // iTotal = 2`
Multiplication	*	Multiplies two values. For example: `iTotal := 2 * 4; // iTotal = 8`
Real division	/	Divides the first value by the second. For example: `rTotal := 10 / 4; // rTotal = 2.5`

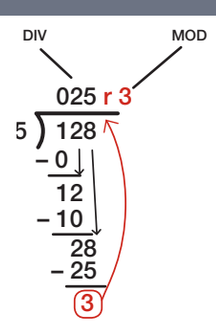
Integer division	div	Divides the first number by the second, then discards the remainder. For example:
		`iResult := 10 div 3; // iResult = 3`
		This can only be used with integer values and the result is always an integer value.

Modulus	mod	Divides the first number by the second, then keeps only the remainder. For example:
		`iRemainder := 10 mod 3; // iRemainder = 1`
		This can only be used with integer values and the result is always an integer value.

Take note

Long division results in a quotient and a remainder:

The variable used to store the result of real division is a real data type variable (rTotal) because the result from the real division operator (/) will return a value with a decimal value. Even if the numerator is exactly divisible by the denominator, the result will still be a real number. For example $\frac{10}{5}$, will result in 2.0. If you try to store this variable in an integer variable, your program will encounter a runtime error and crash.



Activity 4.1

Indicate whether the following expressions are valid or invalid. If valid, write down the answer to the calculation. If invalid, give a reason for why you say so.

4.1.1 8 + 7

4.1.2 56.7 – 23.987

4.1.3 9 MOD 5

4.1.4 2 * 9

4.1.5 4.3 DIV 6

4.1.6 $\frac{10}{4}$

4.1.7 9 DIV 2

4.1.8 7 DIV 3

4.1.9 6 MOD 9

4.1.10 6 DIV 9

4.1.11 8.5 MOD 4

Let's look at some of the rules used for calculations when we are working in Delphi.

ORDER OF PRECEDENCE

When we evaluate basic mathematical operators in mathematical expressions, they need to be evaluated using the BODMAS rule – just as you would do in Mathematics. We refer to this as the order of precedence. The table below lists the order of precedence used in Delphi:

OPERATOR	PRECEDENCE
Brackets ()	Highest level
* / DIV MOD	Second level – from left to right – whichever one comes first
+ –	Third level – from left to right – whichever one comes first

Example 4.1

Evaluate the expression below:

2 + 3 *26 / (16 – 3) – 5

= 2 + 3 * 26 / 13 – 5 (Level 1 – brackets)

= 2 + 78 / 13 – 5 (Level 2 – multiplication)

= 2 + 6 – 5 (Level 2 –division)

= 8 – 5 (Level 3 – addition)

= 3 (Level 3 – subtraction)

Activity 4.2

Evaluate the expressions below:

4.2.1 (12 + 4 * 4) DIV 2

4.2.2 12 + 4 * 4 DIV 2

4.2.3 10 – 4 / 2* 6 + 3

4.2.4 8 * 4 – 17 / 2 + 3

4.2.5 4 * (6/2 + 3)

4.2.6 4 * 6 / 2/4 + 3

4.2.7 17 DIV 2 * (4 * 5 +(10 – 1)) * 2.3

4.2.8 23 MOD 3 * (13 DIV 2 –5)

4.2.9 69 MOD (3 + 5) + 1.1 * 4.7

4.2.10 (32 MOD 7) * (26 DIV 8)

USING A TRACE TABLE

A trace table is a tool used to track how the value of variables change in a program after each line of code is executed. This tool is helpful for testing an algorithm because it helps you to determine if an algorithm gives the correct result. If the result is not correct, the trace table can help you to identify the logical error responsible for the incorrect result.

To create a trace table:

- identify all the variables
- create a table with a column for each line number, a separate column for each variable, and a column for the output
- follow the code line-by-line and write down the new value of the variable that changed.

Take note

Operators with the same importance are executed from left to right, in the order in which they appear.

LEARNING ABOUT ORDER OF OPERATIONS

https://www.youtube.com/watch?v=dAgfnK528RA

Example 4.2

A person buys two items and receives a 10% discount on the total price. The VAT of 15% is calculated on the discounted total. Calculate the total before the discount was applied, the VAT amount payable, and the total due after VAT was added on. Display the final total due.

LINE NUMBER	ALGORITHM
Line 1	Price1 = 19.40
Line 2	Price2 = 10.60
Line 3	Total = Price1 + Price2
Line 4	Total = Total * 0.90
Line 5	Vat = Total * 0.15
Line 6	Total = Total + VAT
Line 7	Display Total

Use the trace table below to trace through the algorithm above:

LINE NUMBER	PRICE1	PRICE2	TOTAL	VAT	OUTPUT
1	19.40				
2		10.60			
3			30.00		
4			27.00		
5				4.05	
6			31.05		
7					31.05

The Delphi Code for the algorithm:

```
…
Var
  rPrice1, rPrice2, rTotal, rVat : real;
Begin
    rPrice1 := 19.40;
    rPrice2 := 10.60;
    rTotal := rPrice1 + rPrice2;
    rTotal := rTotal * 0.90;
    rVat := rTotal * 0.15;
    rTotal := rTotal + rVat;
    lblTotal.Caption := FloatToStr(rTotal);
End;
….
```

USING AN INPUT-PROCESSING-OUTPUT (IPO) TABLE TO PLAN A PROGRAM

An Input-Processing-Output (IPO) table is a planning tool that can be used to record your inputs, processing and output. Usually, before you write a program, you need to determine:

- inputs
- processing
- outputs.

Example 4.3

Write a program that can be used to convert temperature from degree Fahrenheit to degree Celsius.

The formula to convert degree Fahrenheit to degree Celsius is: (Fahrenheit –32)*5/9

INPUT	PROCESSING	OUTPUT
DegreeFahrenheit	DegreeCelsius=(DegreeFahrenheit –32)*5/9	DegreeCelsius

The Delphi Code for the IPO:

```
…
Var
  rDegreeFahrenheit, rDegreeCelsius : real;
Begin
    rDegreeFahrenheit := StrToFloat(edtFahrenheit.Text);
    rDegreeCelsius := (DegreeFahrenheit-32) * 5/9;
    lblCelsius.Caption := FloatToStr(rDegreeCelsius);
End;
….
```

Example 4.4

Three friends Tom, Jerry and Andile need to share sweets in the ratio 3:4:5 respectively. The number of sweets is provided by the user. Part of a sweet will not be shared. Calculate and display how many sweets each friend will receive and how many sweets will remain after sharing takes place.

INPUT	PROCESSING	OUTPUT
NumSweets	Tom = NumSweets * 3 DIV 12 Jerry = NumSweets * 4 DIV 12 Andile = NumSweets * 5 DIV 12 LeftOver = NumSweets MOD 12 OR LeftOver = NumSweets-(Tom+Jerry+Andile)	'Tom's Share: ',Tom 'Jerry's Share: ',Jerry 'Andile's Share: ',Andile 'Number Remaining: ',LeftOver

Algorithm for the example:

LINE NUMBER	ALGORITHM
Line 1	Read NumSweets
Line 2	Tom = NumSweets * 3 DIV 12
Line 3	Jerry = NumSweets * 4 DIV 12
Line 4	Andile = NumSweets * 5 DIV 12
Line 5	LeftOver = NumSweets MOD 12
Line 6	'Tom's Share: ',Tom
Line 7	'Jerry's Share:',Jerry
Line 8	'Andile's Share:',Andile
Line 9	'Number Remaining: ',LeftOver

Example 4.4 *continued*

Use the trace table below to trace through the algorithm when the input for the number of sweets is 100.

Line Numbers	NumSweets	Tom	Jerry	Andile	LeftOver	Output
1	100					
2		25				
3			33			
4				41		
5					1	
6						Tom's share: 25
7						Jerry's share: 33
8						Andile's share: 41
9						Remaining sweets: 1
Stop						

Activity 4.3

4.3.1 Provide the answers for the following calculations:

a. $7 + 2 - 9 \times 1$

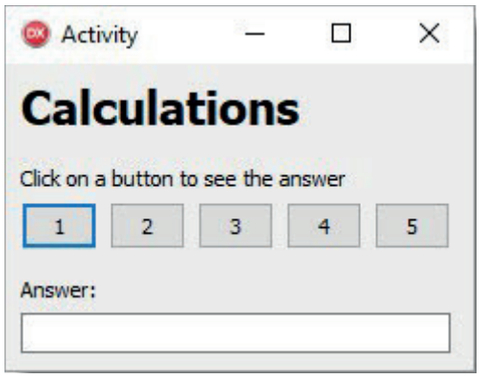
b. $7 + (2 - 9) \times 1$

c. $13 + 12 / (5 - 1)$

d. $(9 \times 5 \times 7) / 5 + 1$

e. $(12 \times 5) \times (3 + 2) / 3$

4.3.2 Create a program **Activities_p** that will calculate and display the answers of the above problems ((a) – (e)). The program can use the following user interface

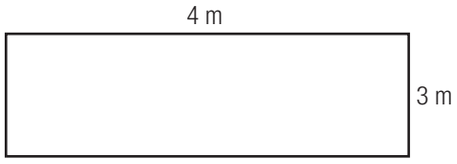


Button *1* refers to problem a above, Button *2* refers to problem b above, and so on.

Activity 4.4

The formula for calculating the area of a rectangle is the length of the rectangle multiplied by the width of the rectangle. With this in mind:

4.4.1 Create an algorithm for calculating the area of a rectangular room.

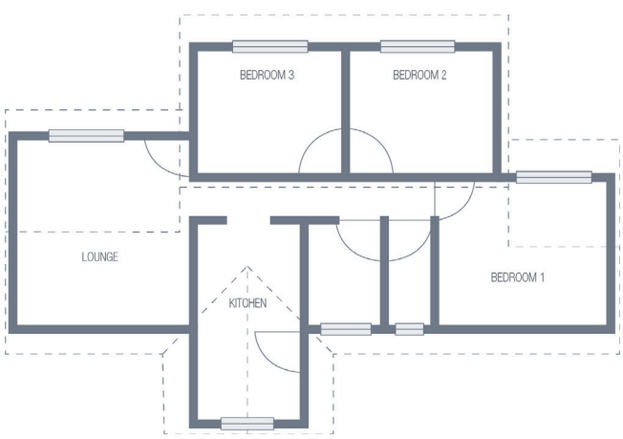


4.4.2 **a.** The owner of the house plan pictured here wants to tile the floors of the lounge andbedroom 2. Create an algorithm for calculating the area that needs to be tiled.

b. Create an IPO chart for tiling the lounge and bedroom 2.

Consider the following items when creating your algorithm and IPO chart.

- The input of your algorithm.
- The processing requirements and steps of your algorithm.
- The output of your algorithm.



Activity 4.5

The code below was written to assist a school principal, who wants to build a soccer field. The principal required the following calculations:

When the length and the width of the field is entered into the program, it should calculate the following:

- The area of the field – the formula for calculating the area is length × width
- The perimeter of the field – to calculate the perimeter you need to add each side of the field together
- The position of the half line – to determine the position of the half line, you need to find the middle of the longest side.

Study the code below to determine if there are any errors:

```
procedure TfrmSoccerPitch.btnCalculateClick(Sender: TObject);
var rHalfLine:integer;
    iArea, iPerimeter:integer;
    iLength, iWidth:integer;
    sSummary:string;
begin

Line 1  iLength.caption:= StrToInt(edtLength.text);
Line 2  iArea := iLength * iWidth;
Line 3  iPerimeter := (iLength*2)/(iwidth *2);
Line 4  iWidth := StrToInt(edtWidth.Text);
Line 5  rHalfline:= iLength/2;
Line 6  lblArea.Caption := StrToInt(iArea)+ ' m';
Line 7  lblPerimeter.Caption := inttostr(iPerimeter)+ ' m';
Line 8  lblHalfLine.Caption:= IntToStr(rHalfLine)+ ' m';
```

Activity 4.5 *continued*

A trace table will help you to identify line by line whether there are any errors. Use the table given below to trace through the code for: iLength = 90; and iWidth = 45. When an error is found, report and correct it in the correction table. The first error has been given to you as an example in the correction table.

TRACE TABLE

LINE NUMBER	iLENGTH	iWIDTH	iAREA (LENGTH * WIDTH)	iPERIMETER (ADD ALL FOUR SIDES)	rHALFLINE	OUTPUT

CORRECTION TABLE

LINE NUMBER	WHAT THE CODE SHOULD DO	ERROR DESCRIPTION	CORRECT THE CODE
1	Write the value from the edit box to the length variable	Error message. Variables do not have properties	iLength := StrToInt(edtLength.text)

UNIT

4.2 Formatting numbers

In Delphi, you can format real numbers to display in a specific format. In this unit we will use the FloatToStrF function to format real numbers. The FloatToStrF function converts a floating point number (real number) into a displayable string based on a given format.

The syntax of a FloatToStrF function is: **FloatToStrF(Value, Format, precision, digits)** where:

- *Value* refers to the real number that will be converted to string
- *Precision* refers to the total number of digits a number will display. If the number of digits in the *Value* exceeds the total number of digits in the *Precision*, then the *Value* is rounded to the total number of digits as per the *Precision*
- *Format* indicates how a *Value* will be formatted into a string. The formats that we will use this year are:

FORMATS ALLOWED	DESCRIPTION
ffCurrency	Formats the value with the currency included
ffFixed	Formats the value with the number of decimal places as specified

  Both Precision and Format control how a Value will be formatted when displayed as a string
- *Digits* indicates the number of places after the decimal comma

Here are some examples:

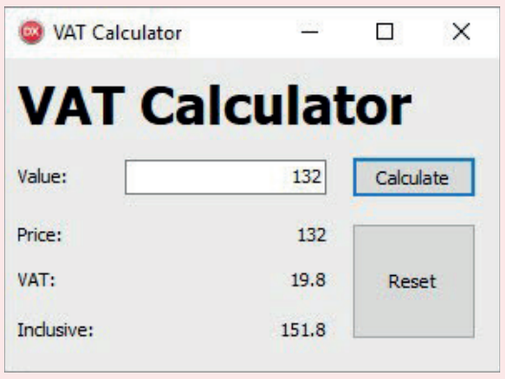
CODE	DISPLAYED STRING
rVal:=452.769; lblOut.Caption:=FloatToStrF(rVal,ffFixed,8,1);	452.8
rVal:=452.769; lblOut.Caption:=FloatToStrF(rVal,ffFixed,4,2);	452.80
rAmount:=78.12; lblOut.Caption:=FloatToStrF(rAmount,ffCurrency,8,2);	R78.12

Let's use this information to help us fix the output for a VAT Calculator.

Example 4.5

To format the output numbers:

1. Open your VAT Calculator application provided in the 04 – VAT Calculator folder.
2. Select the *Price* label.
3. Find the *Alignment* property in the *Object Inspector*.
4. Change it to "taRightJustify". This aligns the numbers on the right, making it easier to read.
5. Resize the label so that it is as large as the *Value* textbox.
6. Do the same for the *VAT* label and the *Inclusive* label.
7. Change the *Alignment* property of the *Value* textbox to be "taRightJustify".
8. Find the [Calculate] button's *OnClick* event in the code.



Example 4.5 *continued*

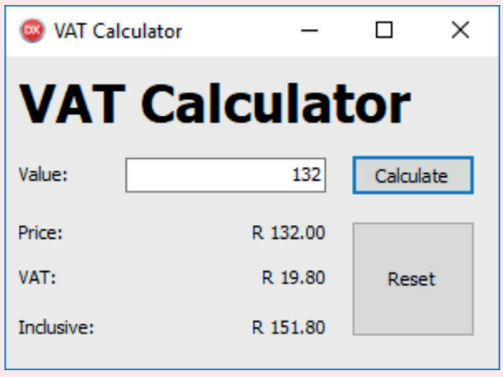
9. Change the code in the procedure so that the labels make use of the *FormatCurr* function as shown below:

Formatting output code

```
procedure TfrmVATCalculator.btnCalculateClick(Sender: TObject);
var
  rPrice, rVat, rInclusive : Real;
begin
  rPrice := StrToFloat(edtValue.Text);
  rVat := rPrice * 0.15;
  rInclusive := rPrice + rVat;

  lblPrice.Caption := FloatToStrF(rPrice, ffCurrency, 8,2);
  lblVat.Caption := FloatToStrF(rVat, ffCurrency, 8,2);
  lblInclusive.Caption := FloatToStrF(rInclusive, 8,2);
end;
```

It is important to note that the *FormatCurr* function returns a string value. As such, you do not need to convert the real number to a string. The FormatCurr function is an alternative to using FloatToStrF and ffCurrency.

10. Save and run the application. The output should now be much easier to read!



Congratulations! While this technique might seem like a small thing, it can make your programs a lot clearer and easier to use. It can also be used to create a format for numbers that are not currencies.

Activity 4.6

Write down format strings that will do the following:

4.6.1 Return a number that displays three digits and no decimal digits.

4.6.2 Return a number correct to two decimal digits.

4.6.3 Return a number as a South African currency (including cents).

Take note!

Surround your format string with single quotation marks. This shows that that it is a string.

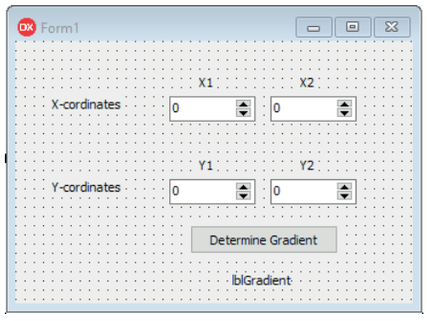
Activity 4.7

4.7.1 The formula to find the gradient of a straight line given two points A(x_1, y_1) and B(x_2, y_2) is:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

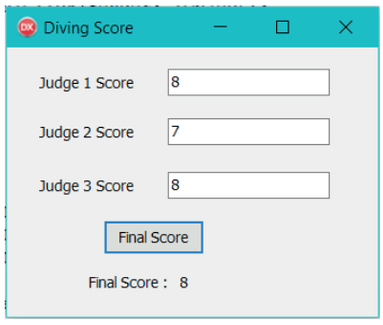
a. Open the **Gradient_p** project from the 04 – Gradient folder. Write a program that will determine the x and y-coordinates for Points A and B on a straight line using SpinEdit components. (**Hint:** Use the object inspector and set the min and max value of each SpinEdit component to –10 and 10 respectively.)

b. Calculate and display the gradient of the line correct to one decimal place.



4.7.2 In a diving competition, three judges score a participant. The final score of the participant is calculated by determining the average of the three scores and ignoring the decimal part.

Open the **Scoring_p** project from the 04 – Scoring folder. Write code for the [Final Score] button.

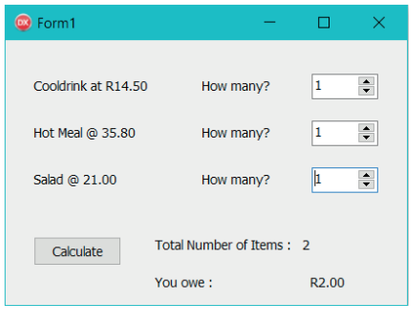


4.7.3 Open the **PreOrder_p** project from the 04 – PreOrder Calculator folder.

This program was created for learners who pre-order a healthy lunch from the tuckshop. The program should be able to give learners the value for how much they owe, but the output of the program is wrong.

a. Use a trace table to identify the errors and correct the errors in the code.

b. Once you have identified all the errors, correct them. Run the program again to determine whether the output is correct.



UNIT

4.3 Mathematical functions

Functions are written by the developers of Delphi. It is a segment of pre-written programming code that performs frequently performed tasks. So far you have worked with these functions: IntToStr, FloatToStr, StringToFloat and StringToInt (the data conversion functions); and FloatToStrF (formatting functions).

In this chapter, you will learn about the following mathematical functions:

- SQR
- SQRT
- ROUND
- TRUNC
- RANDOM

```
iNumber:=random
```

```
function Random(const ARange: Integer): Integer;
```

As you type the function name, Delphi provides a list that gives you the function syntax as follows:

- the function keyword
- the function name, for example, Random
- the data required by the function, for example, (const aRange: integer)
- the return type of the function, for example, :integer.

When you use any mathematical function you need to add the word *Math* to the *Uses* section of the code. This is shown in the example below:

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

When a function is used, we refer to it as being called. The data provided in the function call is called argument/s.

Take note!

If you don't add the word *Math* to the *Uses* section of the code, the Delphi compiler will return a syntax error.

SQR FUNCTION

The square (SQR) function calculates the square of a number (that is, the number multiplied by itself, for example, 2 multiplied by 2).

SYNTAX OF THE SQR FUNCTION

```
Sqr(Number)
```

The argument *number* can be either a positive or negative; integers or real numbers. The *value* returned by the SQR function is dependent on the data type of the argument – it can be a real number or an integer.

Examples

```
rSquare := Sqr(4); // rSquare = 16
rSquare := Sqr(1.6);  //rSquare = 2.56
```

iNum:=7;
lblMessage.Caption:=IntToStr(Sqr(INum)); // the number 49 will display in the label
The statement rSquare:=Sqr(4); is equivalent to rSquare:=4 * 4;

SQRT FUNCTION

The SQRT function calculates the square root of a number.

SYNTAX OF THE SQRT FUNCTION

```
Sqrt(Number)
```

The argument *number* can be either integer or real. Just like in Mathematics, you can only find the square root of a positive number. The SQRT function always returns a real number.

Examples

```
rRoot := Sqrt(16); // rRoot = 4.0
rVal := Sqrt(6.4009) ;  // rVal=2.53
```

ROUND

The ROUND function is used to round a real number to the nearest integer number.

SYNTAX OF THE ROUND FUNCTION

```
Round(Number);
```

The argument *number* is always a real number. It can be negative or positive. If the real number has a decimal value below 0.5, it will be rounded down. If the decimal value is above 0.5, it will be rounded up. If the decimal value is exactly 0.5, it will be rounded to the nearest even number (which could be up or down).

The table below shows how this works.

RVALUE	FUNCTION	RESULT
12.4	Round(rValue)	12
12.5	Round(rValue)	12
12.6	Round(rValue)	13
13.5	Round(rValue)	14
–12.4	Round(rValue)	–12
–12.5	Round(rValue)	–12
–12.6	Round(rValue)	–13
–13.5	Round(rValue)	–14

Examples

```
iAns := Round(2.4); // iAns = 2
iAns := Round(7.52); // iAns = 8
iAns := Round(8.52); // iAns = 9;
iAns := Round(8.5); // iAns = 8
```

Watch out!

A number with a decimal value of 0.5 being rounded to the nearest even number can produce unexpected results.

TRUNC FUNCTION

The TRUNC function truncates (removes or 'chops off') the decimal part of a real number. It returns an integer after the truncation.

SYNTAX OF THE TRUNC FUNCTION

```
Trunc(Number);
```

The argument *number* is always a real number. The result is an integer. Important: no rounding takes place.

Examples

```
iTrunc := Trunc(2.999); // iTrunc = 2
iTrunc := Trunc(2.4);   // iTrunc = 2
iTrunc := Trunc (7.5);  // iTrunc = 7
```

RANDOM FUNCTION

The RANDOM function is used to generate random numbers.

SYNTAX OF THE RANDOM FUNCTION

Random;
OR
Random(Number);

The RANDOM function has two different syntaxes:

- the RANDOM function without argument will generate a random decimal number from 0 to less than 1
- In the RANDOM(Number) syntax the argument is type integer. It generates a random number from 0 to **Number**-1, that is, if the argument is 6, then it will generate a number from 0 to 5
  To generate a number within a particular range a to b, we use the formula : Random(b-a+1)+a. For example, to generate a random number in the range 10 to 100, the statement Random(100–10+1)+10

Examples

```
rRan := random;      //the result is any random real number in the range 0 to less than 1
iRandom := Random(21)      //the result is an Integer number in the range 0 to 20
iRandom := Random(51)+5    //the result is an Integer number in the range 5 to 55
```

The random function is useful in any program where randomness is important. For example, in:

- board games like Tic Tac Toe, a RANDOM can be used to generate a starting move
- word games like Hangman, RANDOM can be used to select a word from a list
- card games, RANDOM can be used to shuffle the deck
- dice games, RANDOM can be used to roll the die.

USE OF RANDOMIZE COMMAND

Delphi will often generate the same random number every time you run the program unless you use the RANDOMIZE command before generating a random number. Here is an example of how to use the RANDOMIZE command:

```
Randomize;
iRandom := Random(100);
```

Let's work through the Guided activity below to see how these functions work in practice.

Guided Activity 4.1

Write down code that will do the following. Assign the result to a result variable of the appropriate data type.

4.1.1 Calculate the square root of 100.

4.1.2 Select a random number between 0 and 10 (excluding 10).

4.1.3 Round the number 13.45.

4.1.4 Calculate the square of 7.1.

4.1.5 Select a random real number between 0 and 1.

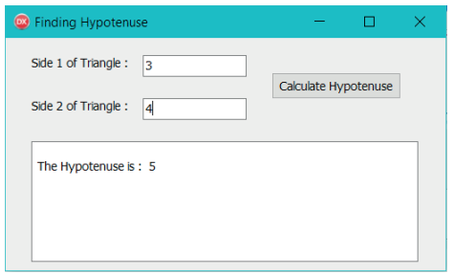
4.1.6 Round the number 1578.99.

4.1.7 Round the number 42.78 down to the nearest integer.

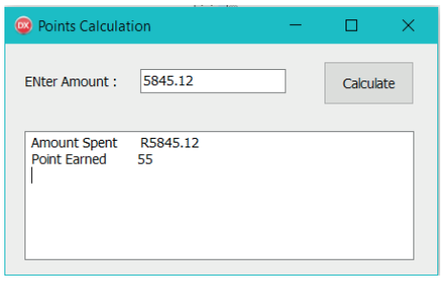
4.1.8 Select a random number between 1 and 6 (including 6).

Activity 4.8

4.8.1 Open the **Hypotenuse_p** project from the 04 – Hypotenuse folder. For a right-angled triangle two sides of the triangle are input via the keyboard. Write code for the [Calculate Hypotenuse] button to calculate and display the hypotenuse of the triangle.

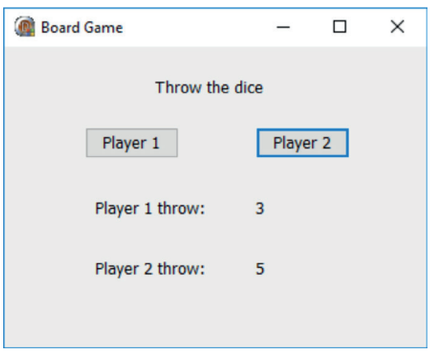


4.8.2 Open the **PointsCalculation_p** project from the 04 – Points Calculation folder provided. Tommy's bank allows him to earn points each month depending on the amount of money he spends. For every R500 he spends he get 5 points. Write code for the [Calculate] button to read the amount of money he spent in a particular month and calculate the points that he has earned for that month. Display the amount spent and the points earned.



Activity 4.8 *continued*

4.8.3 In a board game, two players throw a dice. The player that gets the highest number starts the game. For example, if Player 1 throws 3 and Player 2 throws 5, Player 2 will start.



Create a Delphi project **Boardgame_p** that will simulate the throw of the dice. Use the RANDOM function to determine each player's number.

Note: The program receives no input from the user.

Hint: A dice throw can only result in numbers 1 – 6.

4.8.4 A sphere is a perfectly round geometrical object in three-dimensional space. The formulae to calculate the surface area and volume of a sphere with radius, *r*, is:

$$\text{Surface area} = 4\pi r^2$$

$$\text{Volume} = \frac{4}{3}\pi r^3$$

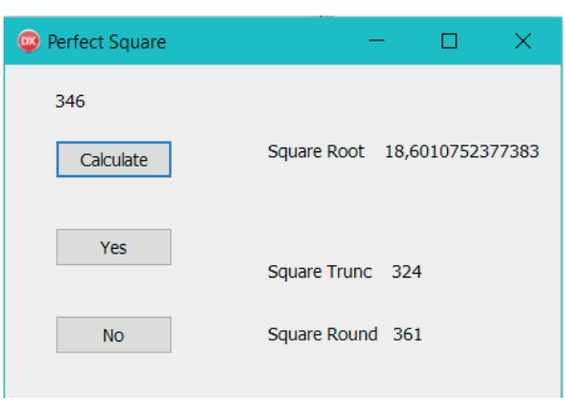
a. Open the **pSphereSurfaceAreaVolume_p** project from the 04 – SphereSurfaceAreaAndVolume folder provided.

b. Write code for a [Calculate] button that will read the radius of the sphere, then calculate and display the surface area and volume.

Activity 4.9

Open the **PerfectSq_p** project in the 04 – Act 4.9 Error folder provided.

This program should enable the user to determine whether a random number is a perfect square.



Take note

A perfect square is a number that produces an integer value for its square root. This means that the square root does not have a decimal value.

The program should do the following:

- Get a random number between 1 and 500.
- Determine the square root of this number.
- Determine the value of the square root if the decimal is removed.
- Determine the value of the square root if the square root is rounded.

Take note

If the square root, the trunc value and the round value are all the same; and if the square of the trunc value is equal to the square of the round value, then the user must press the [Yes] button. Otherwise the user must press the [No] button.

- Square the value of the square root after the decimal has been removed.
- Square the value of the square root after the square root value has been rounded off.

The program in your folder does not execute, nor does it give correct values.

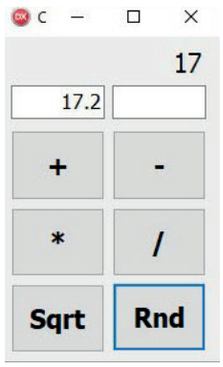
a. Correct the errors in the program.

b. Test your corrected program by running it and checking whether the values are correct according to the description of what the program should do.

c. If there are still errors, use any method that you have learnt in this chapter to find and fix the errors.

Activity 4.10

4.10.1 Open the **intermediateCalculator_p** project from the 04 – Intermediate Calculator folder provided. The program receives two numbers as input and displays the result of the operation, depending on the button (+, -, *, /, sqrt, sqr) clicked



a. Create event handler for the buttons to calculate the result of the operation.

b. Save and run the project.

4.10.2 Write down functions that will do the following and store the answers in an appropriate variable.

a. Round the real number 37.42.

b. Find the square root of 169.

c. Select a random number between 0 and 1000 (including 1000).

d. Round the product of 3.7 multiplied by 4.2.

e. Find the square root of a random number between 0 and 25.

UNIT

4.4 Variable scope

In Chapter 3, we learnt that a variable has a:

- name
- type
- value
- location.

In this unit you will learn about another property of a variable – its scope.

A variable must first be declared before it can be used in a program. However, where it is declared in a program, determines its scope. Variables declared in an event handler are only created in the computer's memory at the start of the event, and only exist as long as the event is being executed. These variables cease to exist once the event has terminated. We call these variables **local variables**, because they cannot be accessed from another event – that is, they have a *local scope*.

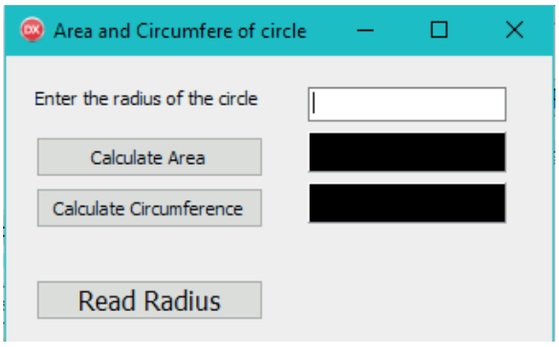
If a variable needs to be accessed from more than one event handler, it must be declared outside of the event handlers in the VAR section at the top of the program. We say that these variables have *global scope*. For example, the variable *iCount* in the code snippet below is declared globally. This means it can be accessed and changed from any event handler in the program.

```
….
type
  TForm1 = class(TForm)
    btnShow : TButton;
    lblMessage : TLabel;
    edtName : TEdit;
    lblShoutOut : TLabel;
    lblOut : TLabel;
    procedure btnShowClick(Sender : TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1 : TForm1;
  iCount :Integer;        //globsl variable
implementation
…
```

Guided Activity 4.2 Determining the area and circumference of a circle

4.2.1 Open the **Circle_p** project located in the 04 – Circle folder provided.

4.2.2 The design screen will display:



Radius is declared as a local variable.

4.2.3 The algorithm for calculating area and circumference of a circle using local variables reads as follows:

- OnClick Event for the [Calculate Area] button
  - The radius will be declared and read locally and used to calculate the area.
  - Area := Pi * radius * radius
  - Area will be a local variable
  - Display the area in lblArea
- OnClick Event for the [Calculate Circumference] button
  - The radius will be declared and read locally and used to calculate the area.
  - Circumference := 2 *Pi * radius
  - Circumference will be a local variable
  - Display the area In lblCircumference

4.2.4 Write the code using the algorithm in number 3 above:

Radius as a global variable

4.2.5 You may notice that the radius was read twice, once in the *Calculate Area* event handler and once in the *Calculate Circumference* event handler.

4.2.6 Remove the local declaration of radius from the *Calculate circumference* event handler. You will get an error even though radius is declared in the *Calculate Area* event handler. The scope of the radius declared in the *Calculate Area* event handler is only between the BEGIN and END of this event handler.

4.2.7 Since both events need to use the value of the radius, radius must be declared globally. So, remove the local declaration of radius from the Calculate Area event handler. You will get another syntax error.

4.2.8 Once you declare the radius variable globally, both syntax errors will disappear.

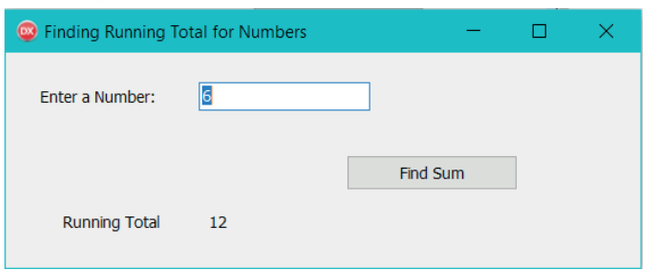
4.2.9 Finally, in the [Read Radius] button, read the radius from the *edtRadius* component and assign it to the global variable, radius.

Remember!

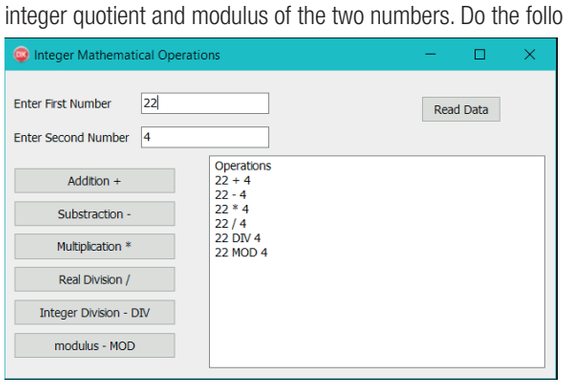
A variable has a:
- name
- type
- value
- location
- scope

Activity 4.11

4.11.1 Open the **RunningTotal_p** project from the 04 – Running Total folder provided. Write code for the [Find Sum] button that will read integer values entered in the edit box and determine a running total of the numbers entered, i.e. every time and integer value is entered, and the [Find Sum] button is clicked, the total is updated to include the number entered and the updated total is displayed. Note, that the answer will be displayed in label **lblRunningTotal**.



4.11.2 Open the **MathsOperations_p** project from the 04 – Mathematical Operations folder provided. The program reads two integer numbers (once only) and determines and displays the sum, difference, product, real quotient, integer quotient and modulus of the two numbers. Do the following:



a. Create an OnCreate event to clear the Memo box.

b. Write code for the [Read Data] button to read the two integer numbers from the edit boxes.

c. Write code for the **[Addition +]**, **[Subtraction –]**, **[Multiplication *]**, **[Real Division /]**, **[Integer Division-DIV]** and **[Modulus – MOD]** button to perform the operations for the two numbers that were read in the bullet above.

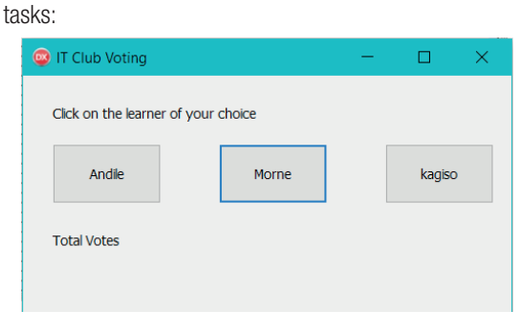
d. Display the answers of the mathematical operations in the Memo box.

4.11.3 An IT computer club was formed at ABC High School. Three learners have been selected to serve either as president, treasurer and secretary. The position each learner will be allocated will depend on the votes they receive from fellow club members.

a. Open the file **ITClubVoting_p** project from the 04 – IT Club Voting folder provided.

b. Correct the program so that it completes the following tasks:

- Each learner's total votes received will be displayed in the labels found below their names. Each learner will start with a total vote of zero when the program is initially executed.
- Club members vote by clicking on the button of the learner of their choice. Once a learner is selected, the total for that learner must increase by 1 and the increased total must be displayed.



BASIC MATHEMATICAL PROBLEMS USING DELPHI

Basic operators
Symbols that tell Delphi which mathematical operators to use

+ (Addition)	iTotal := 2 + 1;	// iTotal = 3
– (Subtraction)	iTotal := 5 – 3;	// iTotal = 2
* (Multiplication)	iTotal := 2 * 4;	// iTotal = 8
/ (Real division)	rTotal := 10 / 4;	// rTotal = 2.5
div (Integer division)	iResult := 10 div 3	// iResult = 3
mod (Modulus)	iRemain := 10 mod 3	// iRemain = 1

Plan your project

Using a trace table
To determine the values of variables throughout the program

Using an IPO table
Input-Processing-Output
Planning by keeping the end result in mind

BODMAS (Order of calculation operations)

1. Brackets (…)
2. Orders $\sqrt{x^2}\ x^2$
3. Division ÷
4. Multiplication ×
5. Addition +
6. Subtraction –

Formatting Numbers

FormatCurr('FormatString', Number);
E.g.
FloatToStrF(123.456, ffFixed, 8,2);
→ 123.46
FloatToStrF(123.456, ffCurrency, 8,2);
→ R123.46

The FloatToStrF function determines the way in which the number will be formatted

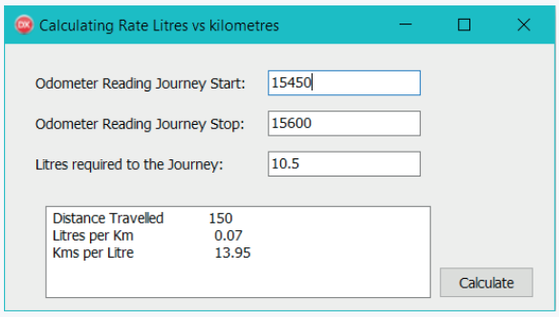
Mathematical Functions

Square	rSquare := Sqr(4);	// rSquare = 16
Square root	rRoot := Sqrt(16);	// rRoot = 4
Round	iRound := Round(2.4);	// iRound = 2
Truncate	iTrunc := Trunc(2.99);	// iTrunc = 2
Random	iRandom := Random(100);	// iRandom = any integer // value from 0 to 99

Variable Scope

- Local variable
  - Variable value only available inside the procedure where it is declared
- Global variable
  - Variable value available throughout the program

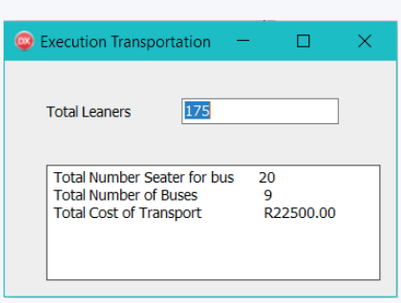
1. Explain the difference between the DIV and the MOD operators in Delphi.
2. Explain the difference between a local variable and a global variable.
3. What is the purpose of a trace table in programming?
4. Explain what an argument of a function is.
5. What is the purpose of the RANDOM function?
6. Open the **RateLitresKm_p** project from the 04 - RateOfLitresAndKilometres folder provided. At the start of Thandi's journey the odometer reading was *iStart* kilometres, and at the end of the journey her odometer reading was *iStop* kilometres. Her petrol tank was full at the start of the journey. At the end of her journey she filled *rLitres* of petrol so that her tank would be full again. Input values for *iStart*, *iStop* and *rLitres*.



   a. Write code for the [Calculate] button that will:
      - read in the odometer reading at the start and stop of the journey
      - calculate the number of litres required to fill up the tank at the end of journey.
   b. Calculate and display:
      - total kilometres travelled
      - rate of litres per kilometre
      - rate of kilometres per litre.

7. Open the **ExcursionTransportation_p** project located in the 04 – Excursion Transportation folder provided.

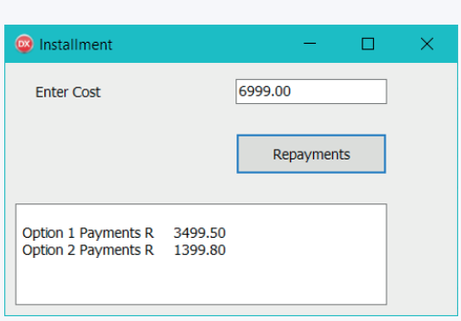
   ABC school is taking learners on an excursion. They approach a bus company for a quotation. They confirm that they have 20-seater buses and that the cost of hiring one bus is R2500.



   a. Write code for the [Quotation] button to enter the total number of learners that will be going on the excursion.
   b. Determine and print the number of buses required and the total cost of hiring the buses.

8. Open the **InstallmentCalculations_p** project located in the 04 – Installment Calculation folder provided.

   Nathi wants to buy a sofa from Furniture Galore Store. He has two options to make payment for the sofa:



   - **Option 1:** Two payments without being charged interest, or
   - **Option 2:** Six monthly payments. 20% interest will be added to the cost of the sofa before the six monthly payments are calculated.

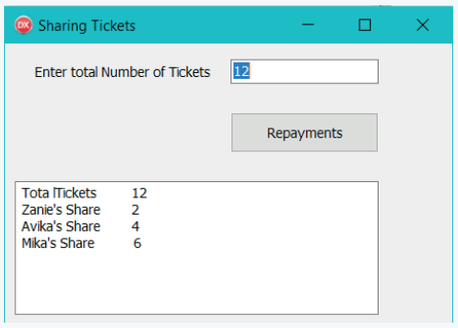
   Write code for the [Repayments] button. The button will have to read the cost of the sofa and then to determine and print the amount for the first instalment if Option 1 is chosen, and the amount for the first instalment if Option 2 is chosen.

9. Open the **SharingTickets_p** project located in the 04 – Sharing Tickets folder provided.

Three friends Zanie, Avika and Mike must share complimentary tickets to a concert in the following way:

- Avika will receive twice as much tickets as Zanie, and
- Mike will receive three times as much as Zanie.

Write code for the [Share] button. The button must read in the total number of tickets to be shared, then determine and display how many tickets each person will receive.



10. Open the **DistanceConversion_p** project located in the 04 - Distance Conversion folder provided.

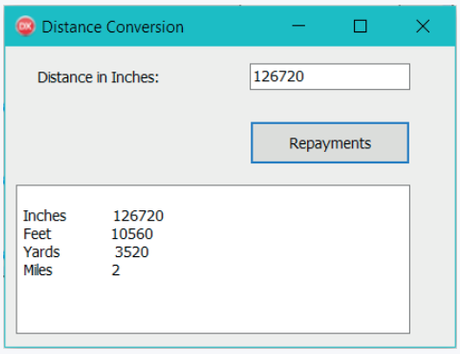
Distance in certain countries are measured in inches, feet, yards and miles. These are measured as follows:

12 inches = 1 foot

3 feet = 1 yard

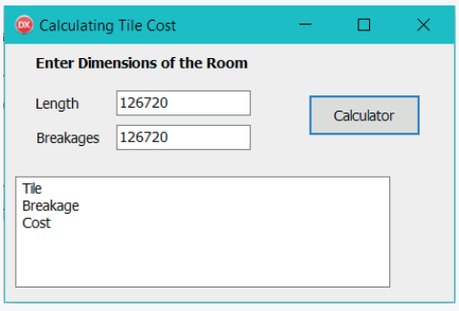
1760 yards = 1 mile

Write code for the [Convert] button that will read in the distance in inches and convert the distance to foot(feet), yards and miles.



11. Open the **TileCost_p** project located in the 04 – Tile Cost folder provided.

Andile wants to tile his room floor. He must calculate the m^2 that he requires for his room. In addition, he needs to add 10% additional m^2 to his requirements to cater for breakages. He likes a tile that costs R150 per m^2. The tiles are sold in boxes. Each box holds tiles with a measurement of 2.3 m^2.



Write code for the [Tile Calculator] to read in the length and breadth of Andile's room and determine and print the following:

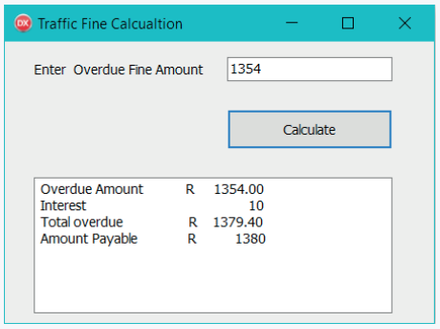
a. The total m^2 tiles needed (inclusive of breakages).

b. The number of boxes of tiles that must be bought.

c. The total amount that will be paid for the boxes of tiles.

12. Open the **TrafficFines_p** project located in the 04 – Traffic Fines folder provided.

A traffic accounts departments charges 10% interest on all overdue fines.



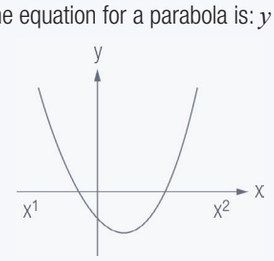
a. Write code for the [Calculate] button to read in the amount of the overdue fine, and then calculate the interest payable and the total amount of fine payable.

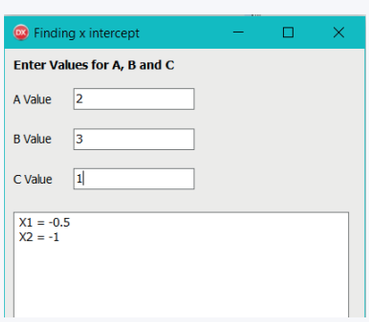
The traffic accounts department only accepts amounts converted to the nearest Rand, for example, if the amount for an overdue fine with interest is R456.67, then the amount payable is R457.

b. Display the amount of overdue fine, the interest on the overdue fine, total amount of fine due and total amount payable to the traffic department.

13. Open the **Parabola_p** project located in the 04 – Parabola folder provided.

The equation for a parabola is: $y = ax^2 + bx + c$





The following equations are used to find the two values for x where the graph cuts the x-axis.

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

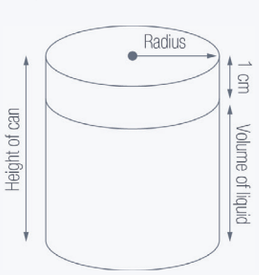
$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

a. Write code for the [Calculate] button that will allow the user to input values for a, b and c

b. Calculate and display the values for x_1 and x_2.

c. Test your program using the following values: a:=2, b:=3 and c:=1

14. A supplier of cooldrink cans must determine the volume of a can before it is filled with liquid. All cans are filled with liquid.



Open the **CanVolume_p** project located in the 04 – Can Volume folder provide.

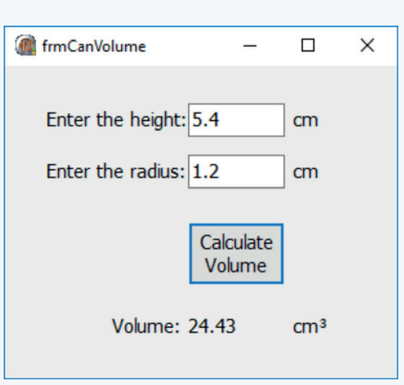
Write code to do the following when the [Calculate Volume] button is clicked:

a. declaring a variable for the height and radius of the can.

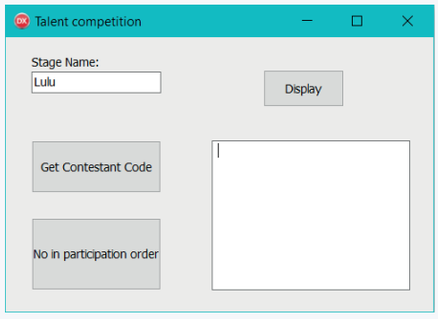
b. calculating the volume of the liquid required (according to the specifications) to fill a can if the height and radius of the can are provided as input from the user. The formula for the volume of a cylinder is: $V = \pi r^2 h$.

c. using a label to display the volume of the liquid in the can, calculated to ONE decimal place.

Look at this example of input and output if the height of a can is 5.4 cm and the radius is 1.2 cm:



15. Your school is hosting a talent contest.
You were asked to write a program to generate entry numbers for the contestants and to determine the order of the participation.



Open the **Talent_p** project located in the 04 – Contestant Code folder provided and complete the program.

a. Write code for *btnCode* to create a participant number as follows:

- Generate a random number between 10 and 200.
- Add this random number to the stage name the contestant entered in the edit box.
- Save this code in an appropriate variable.

b. Write code for the *btnOrder* to determine the order in which this contestant will participate by assigning a random order to the participant.

- Assume that there are 45 contestants in this competition.
- The order for participation starts at 1.

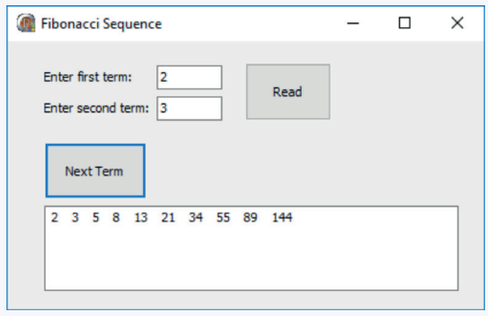
c. Write code for *btnDisplay* to display the following details of the contestant:

- The stage name
- The contestant code
- The order of participation

Did you know

You can build a longer string in a string variable by using + to join all the parts of the string.

16. Open the **FibonacciSequence_p** project located in the 04 – Fibonacci Sequence folder provided.



In a Fibonacci sequence, the first two terms are entered using the keyboard. Thereafter, every term (starting with the third term) is generated by adding the previous two terms. For example, if 2 and 3 are entered, then the Fibonacci sequence is:

2 3 5 8 13 21 34 55 …

a. Write code for the [Read] button that will read the first two terms from the two edit boxes and store them in two variables.

b. Write code for the [Next Term] button so that each time it is clicked, it will generate the next term in the sequence.

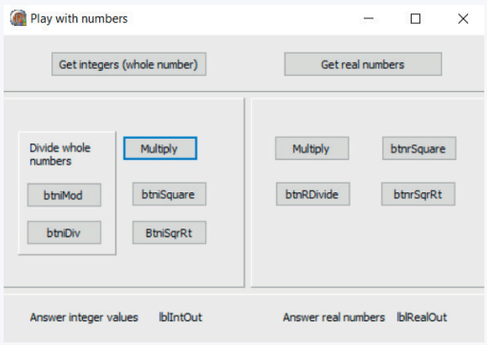
Did you know

To display the details of the contestant in the memo component use the code memOutput.lines.add(Stringvalues)

The description 'Stringvalues' mean the values that you would like to appear in the memo component

17. Write a program to play with numbers!

Open the **Numbers_p** project located in the 04 – Playing With Numbers folder provided. Complete the code for each button as follows:



a. Add code to *btnGetInteger* to save two random integer numbers to appropriate variables.

- Display the value of the first number in the *lblIntOut*.
- Name your variables *iNum1* and *iNum2*.

b. Study the code of the [btnGetReal] button below. Its purpose is to save two real random numbers to appropriate variables. These values must be between 1 and 100. Display the first number in *lblRealOut*, showing only two decimals.

- Name your variables *rNum1* and *rNum2*.
- There is an error in this code. Draw a trace table to find and correct the error.

Take note

Remember these values will be accessed from various buttons, so think carefully about where they should be declared.

```
procedure TfrmNumbers.btnGetRealClick(Sender: TObject);
begin
Randomize;
  rNum1 := random(10) ;
  rNum2 := random;
  rNum1 := rNum1*100;
  rNum2 := rNum2*10;
  lblRealOut.caption := floattostrF(iNum1,ffFixed,4,2);
  end;
```

c. Multiplication:

Write code for the [Multiply] buttons to multiply the two numbers saved in the variables. Remember to use the integer values for all the calculations on the left, and the real values to right. Display the answers in the labels for output. The decimal values should display three values.

d. Division:

- Add code to the *btnrDivide* to divide *rNum1* by a random number between 5 and 15. Display the number in the label for real output. Round the number.
- Add code to *btniMod* to find the remainder of *iNum1* divided by a random number between 0 and 9. Display the answer in the label reserved for integer output.
- Add code to *btniDiv* to find the integer value of *iNum2* divided by a random number between 1 and 10.

e. Square:

- Write code for *btniSquare* to display the squared value of the second number.
- Write code for the *btnrSquare* to display the squared value of the second number, displayed without any decimals. DO NOT round the number.

f. Square Root

- Determine the square root of *iNum1*. Display the value as an integer value.

  (**Hint:** The ROUND and TRUNC functions will remove the decimal values.)
- Determine the square root of *rNum2*. Display the value with one decimal value.

TERM 2

CHAPTER 5

DECISION MAKING

CHAPTER UNITS	
Unit 5.1	Decision making algorithms
Unit 5.2	Boolean expressions and the IF-THEN statement
Unit 5.3	Boolean operators
Unit 5.4	IF-THEN-ELSE statement
Unit 5.5	Nested IF-THEN statements
Unit 5.6	CASE statements

Learning outcomes

At the end of this chapter, you should be able to:

- implement decision making in algorithms, flowcharts and code
- explain what a condition is
- use Boolean operators to create conditions
- use complex conditions in decision making.

INTRODUCTION

Every day you must make choices in your life. Think about this morning:

- When your alarm went off, did you choose to wake up or did you snooze the alarm?
- What did you do after you got out of bed?
- Did you brush your teeth first or start by getting dressed?
- When getting dressed, did you choose to wear warm or cool clothes?

Just imagine … This is only the start of your day, and you may have had to make so many decisions already! People make hundreds of decisions every day. These can range from small, inconsequential decisions (such as what to wear), to large, life changing decisions (such as deciding what to study after school).
But how does this relate to computers?

So far you have learned that a computer program is a set of instructions that solves a problem. When solving a problem, choices have to be made. In Delphi, we can use decision making structures in the applications that we build – that is, you can tell your program how to make the decision that will generate the outcome you want.

In this chapter, you will learn about the decision-making process used in Delphi. These processes will include:

- Boolean expressions
- IF-THEN statements
- IF-THEN-ELSE statements
- Nested-IF statements
- CASE statements

You will also learn how Boolean operators can be used with these structures to create more complex conditional statements.

UNIT

5.1 Decisions in algorithms

When you are programming, you often need to make a decision which will help you to solve a problem. During the decision-making process, certain **conditions** (or criteria) are tested.

A condition evaluates one of the two Boolean values: TRUE or FALSE. The outcome of the condition determines which one of the two paths (the YES/TRUE path or the NO/FALSE path) will be followed.

In an algorithm, decision statements usually start with the word 'if', followed by the condition to be tested.

In a flowchart, a decision is represented by a diamond symbol. Decision making causes **branching** to occur in the normal sequential program flow.

Here are some examples showing these:

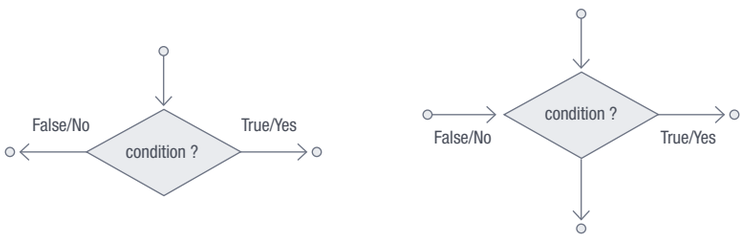


Figure 5.1: *Branching along TRUE or FALSE paths*

Do you remember the algorithm for making hot chocolate from Chapter 1? Here is what the algorithm looked like:

1. Add water to a kettle and turn the kettle on.
2. Add four teaspoons of hot chocolate to a cup.
3. Add 30 ml of milk to the cup.
4. Add one teaspoon of sugar to the cup.
5. Add boiling water from the kettle to the cup.
6. Stir the hot chocolate for 10 seconds. Your hot chocolate is now ready to drink!

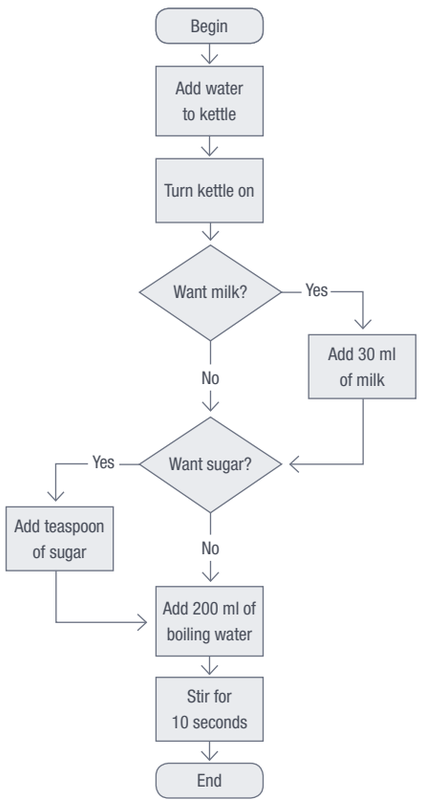
With this algorithm we found a problem: *not all people make their hot chocolate in the same way. Some people only add water, other people add milk and water. Some people drink their hot chocolate with sugar, other people drink it without sugar*.

If the goal of the algorithm is to make tasty hot chocolate, the original algorithm might have a lower quality since the hot chocolate would only be tasty to some people. So, to improve the algorithm, you could build a few decisions into it. These decisions will allow the algorithm to do one thing in some circumstances, and something else in other circumstances.

Take note

You were introduced to the Boolean data types in Chapter 3.

The hot chocolate algorithm could be improved as follows:

ALGORITHM	FLOW CHART
The following algorithm includes two conditional statements: 1. Add water to a kettle and turn the kettle on. 2. Add four teaspoons of hot chocolate to a cup. 3. If the user wants milk then: Add 30 ml of milk. 4. If the user wants sugar then: Add one teaspoon of sugar. 5. Add 200 ml boiling water from the kettle to the cup. 6. Stir the hot chocolate for 10 seconds. Your hot chocolate is now ready to drink!	

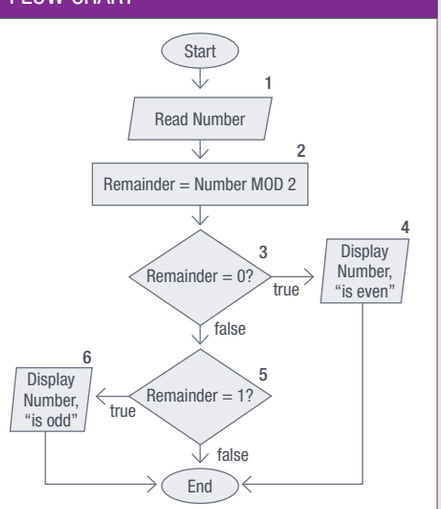
If you have a way (such as asking the user) of measuring whether the user wants milk or sugar, the decision statements allow you to create a hot chocolate algorithm that is more flexible.

Let's look at a different example.

Example 5.1 Determine whether a number is odd or even

When an even number is divided by 2, it does not have a remainder.

When an odd number is divided by 2, it has a remainder of 1.

ALGORITHM	FLOW CHART
1. Read Number 2. Remainder = Number mod 2 3. if remainder = 0 then 4. Display number, 'is Even' 5. if remainder = 1 then 6. Display number, 'is Odd'	

Take note

There are two decisions in the flowchart

- The condition of the first decision is: Is the Remainder = 0?
- The condition of the second decision is: Is the Remainder = 1?
- Conditions evaluate to TRUE or FALSE. Branching occurs according to the outcome of the test (whether the result of the test is TRUE or FALSE).

DECISION BOXES IN TRACE TABLES

When working with trace tables, each decision (condition) is represented with a question mark (?) in its own column. Depending on the result of the test, you then need to indicate TRUE or FALSE in the decision column.

Let's trace through the flowchart that was used to determine whether a number is odd or even if the input value for the number is 5:

BOX NUMBER	NUMBER	REMAINDER	REMAINDER = 0?	REMAINDER = 1?	OUTPUT
1	5				
2		1			
3			False		
5				True	
6					5 is Odd
Stop					

Activity 5.1

5.1.1 Use the skills that you have learned so far, then develop algorithms and flow charts for the following problem situations:

- **a.** A person must decide whether or not to take an umbrella when going to the shop.
- **b.** Read a person's gender in the format 'Male' or 'Female'. If the person is a female display 'F'; otherwise display 'M'.

5.1.2 Draw a trace table for the Even and Odd number flowchart using an input value of 6.

5.1.3 Draw a flow chart that receives a subject name and a mark out of 100 for the subject and then displays a message 'Pass' if the mark is 50 or more.

UNIT

5.2 Boolean expressions and the If-then statement

BOOLEAN EXPRESSIONS

A Boolean expression is an expression that evaluates to TRUE or FALSE. There are three categories of Boolean expressions. These are:

- Boolean variable
- Simple Boolean expressions
- Compound (complex) Boolean expressions – this will be discussed in Unit 5.3.

BOOLEAN VARIABLES

You can assign a Boolean value to a Boolean variable. For example:

- bFound := True;
- bValid := False;

SIMPLE BOOLEAN EXPRESSIONS

Boolean expressions are created by comparing variables/values of the same data types using comparison operators. This means that numeric data can only be compared with other numeric data (INTEGER and REAL); and text data can only be compared with other text data (CHAR and STRING).

The table below shows the comparison operators (relational operators) that you can use to create simple Boolean expressions:

Table 5.1: *Comparison operators and their meaning*

COMPARISON OPERATORS	MEANING
=	Equals
<>	Not equals to
<	Smaller than
<=	Smaller or equal to
>	Greater than
>=	Greater of equal to

Let's look at some examples:

Example 5.2

The computer needs to check if your name is "Sipho".

Name = "Sipho":

Name: Variable; =: Comparison Operator; "Sipho": Value

Example 5.3

The computer needs to check if the mark is more than 30

Mark > 30

Mark: Variable; >: Comparison Operator; 30: Value

Example 5.4

```
Var
  iA, iB : Integer
  sWord1 : string;
  cWord2 : char;
begin
  iA := 7;
  iB := 7;
  sWord1 := 'Seven';
  cWord2 := '7';
end;
```

EXPRESSION	IS THE EXPRESSION VALID?	RESULT	REASON
`iA = iB`	Yes	True	Both A and B have the value 7
`iB > 5`	Yes	True	The value of B is 7 which is greater than 5
`iA – iB = 5`	Yes	False	A – B equals 0 not 5
`iA = sWord1`	No	Error	A numeric value cannot be compared with a text value. This will give the error message "incompatible data types".
`sWord1 = cWord2`	Yes	False	'Seven' is not equal to '7' as a string of characters

Activity 5.2

Study the following code snippet and answer the questions that follow:

```
Var
  iA : 7;
  iB : 7;
  sWord1 : 'Seven';
  cWord2 : '7';
begin
  iA := 2;
  iB := 2;
  sWord1 := 'Joly';
  cWord2 := 'A';
end;
```

5.2.1 Determine whether the following statements will return TRUE or FALSE. Give a reason for you answer.

5.2.2 Copy and complete the table below.

EXAMPLE	RESULT	MOTIVATION
`iA > 2`		
`iA = iB`		
`iA >= 2`		
`iA + iB = 0`		
`iA div 2 = iA mod 2`		
`sWord1 <> cWord2`		
`sWord1 > cWord2`		
`cWord2 = 'A'`		

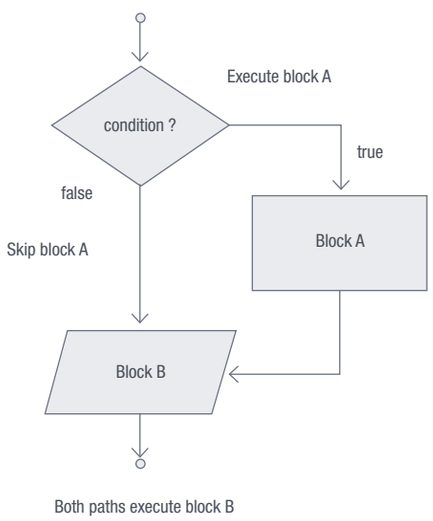
THE IF-THEN STATEMENT

IF and THEN are keywords in Delphi. To make a decision in your programming, you can use an IF-THEN statement in your code. The IF-THEN statement executes the statement/s following the THEN-keyword if the condition is TRUE and skips the execution of these statement/s when the condition is FALSE.

Take note

Remember that you cannot use IF and THEN as variable names. If you do, you will get an error 'Declaration expected but IF found'.

In the flow chart below, block A represents statement following the THEN-keyword. Pay careful attention to how block A is skipped if the condition is false.



SYNTAX OF IF-THEN STATEMENT

Below is an example of the Delphi syntax of an IF-THEN statement, if one statement follows the THEN-keyword:

```
If <condition> then
    <statement1>;
```

Note that the THEN-keyword it not followed by a semicolon because it is not the end of the statement. However, the <statement1> is followed by a semicolon, and ends the IF-THEN statement.

To help you to understand how to implement the IF-THEN statement using Delphi code, work through the following algorithm that determines if one number is a factor of another number.

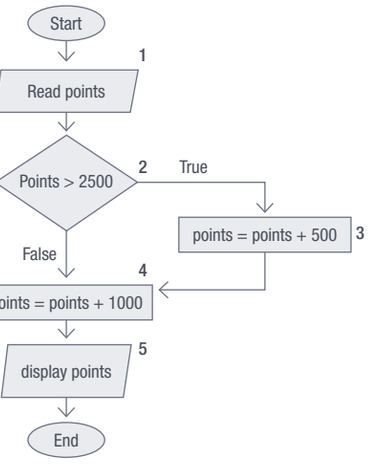
Example 5.5 Determine points

A customer is awarded points on a store card. The store decides to award 1000 bonus points to all its customers. Those customers who have points greater than 2500 before the bonus points are added, will be awarded an additional 500 points.

Create a flowchart that you can use to read a customer's current points on the store card, calculate the final points after the bonus points have been allocated, then displays the customer's final points.

Example 5.5 Determine points *continued*

```
Read points
If points > 2500 then
        points = points + 500;
points = points + 1000;
Display points
```



If points equal 1200, trace through the owchart using the trace table below:

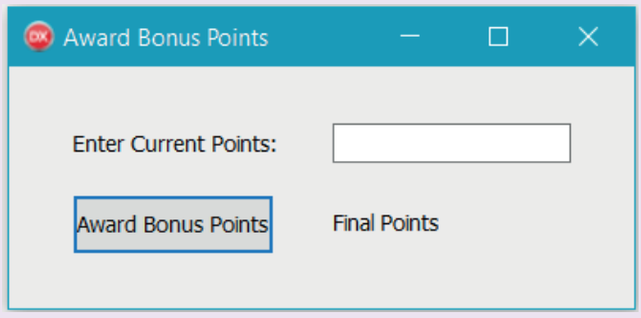
BOX NUMBER	POINTS	POINTS > 2500?	OUTPUT
1	1200		
2		False	
4	2200		
5			2200
Stop			

If points equal 3000, trace through the flowchart using the trace table below:

BOX NUMBER	POINTS	POINTS > 2500?	OUTPUT
1	3000		
2		True	
3	3500		
4	4500		
5			4500
Stop			

Now let's code the algorithm into Delphi.

1. Open the **AwardBonusPoints_p** project located in the 05 – Bonus Points folder.



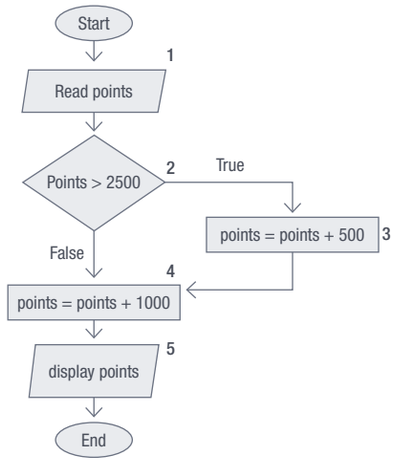
Example 5.5 Determine points *continued*

2. Create an OnClick event on [Award Bonus points] button to determine the final bonus points.

```
procedure TForm1.btnAwardPointsClick(Sender: TObject);
var
  iPoints : Integer;
begin
  iPoints := StrToInt(edtCurrentPoints.Text);
  if iPoints > 2500 then
    iPoints := iPoints + 500;  //end of IF statement
  iPoints := iPoints + 1000;
  lblFinalPoints.Caption := IntToStr(iPoints);
end;
```

Activity 5.3

Study the flowchart below and then answer the questions that follow:



5.3.1 If Box 4 is moved immediately below Box 1, will the algorithm work? Explain.

5.3.2 What will happen if Box 3 and Box 4 are interchanged?

5.3.3 Use trace tables to verify your answer to questions 1 and 2 above.

5.3.4 Amend the flowchart to display the original points, as well as the final points.

Activity 5.4

Write down Delphi conditional statements to do the following:

5.4.1 Set variable *sName* to 'John' if the value in variable *sSurname* equals to 'Karabo'.

5.4.2 Set variable *sEvenOrOdd* to 'Even' if the value in variable *iRemainder* equals 0.

5.4.3 Double the value in variable *iValue* if the value in variable *iInput* does not equal to 10.

5.4.4 Set variable *sGender* to 'Female' if the Boolean value in variable *bGender* is True.

5.4.5 Increase the value of variable *iTotal* by 10 if the value of variable *iTotal* is greater than or equal to 100.

Sometimes, if a condition is true, we need to execute more than one statement. Multiple statements are grouped together within the keywords Begin and End as shown below. It is seen as one group of statements to be executed in the THEN part.

Here is the syntax of an IF-THEN statement if more than one statement follows the THEN-keyword:

```
If <condition> then
begin
    <statement1>;
    <statement2>;
…
end;
```

Remember that the THEN and BEGIN keywords are not followed by a semicolon. However, the END statement marking the end of the IF-THEN statement is followed by a semicolon.

Example 5.6

```
If Gender = 'F' then
begin
  iNumGirls := iNumGirls + 1;
  lblNumGirls.caption := IntToStr(iNumGirls);
end;
```

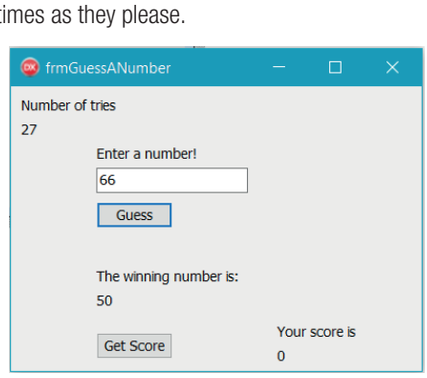
Activity 5.5

5.5.1 ABC stores have a promotion where a customer can win stars based on their spending. A customer will win one star for every R150 spent. If the customer spends more than R4000, they receive four extra stars. Customers cannot win part stars.
Open the **Reward_p** project located in the 05 – Stars Promo folder and complete the program by adding code to *btnCalculate* to determine how many stars a client must receive.
Test yourself: If a client enters 4150 your answer must be 31.

Watch out!

As a client cannot win part stars there must not be any decimal numbers. Refer to Chapter 4 to look at the difference between TRUNC and ROUND!

5.5.2 In this guessing game, a player guesses a number between 50 and 100.
The computer then displays a winning number. If the number is correct, a message with the words, 'Great Stuff' is displayed to congratulate the player AND 10 marks is added to the score.
If the number is 5 less than the winning number, 1 mark is added to the player's score and a message with the words, 'Keep going' is displayed.
A player may try as many times as they please.



Activity 5.5 *continued*

a. Open the **Guess_p** project located in the 05 – Guess a Number folder.

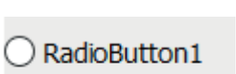
b. Complete the code for *btnGuess* as follows:
The winning number is generated randomly between 50 and 100. If the number guessed by the player is equal to the winning number, 10 is added to the score and a message is displayed in *lblMessage*.
If the number is 5 less than the winning number, 1 is added to the score and a message is displayed in *lblMessage*.

c. Complete the code for *btnGetScore* by displaying the score in *lblScore*.

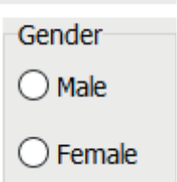
d. Run your program.
Hint: Use the variables provided.

THE RADIOBUTTON COMPONENT

A *RadioButton* component or option button that allows a user to choose only ONE button from a group of multiple buttons. To place a *RadioButton* on a form, you need to select the *TRadioButton* component from the *Standard Palette*.

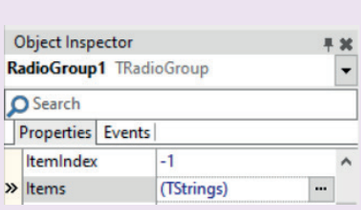


If you want to use radio buttons in groups of two or more – as shown in the figure alongside – the *TRadioGroup* component allows you to do this in a neat and dynamic way. When radio buttons are placed on a *TRadioGroup*, only ONE radio button can be selected at a time. If radio buttons are not placed in a *TRadioGroup*, then the radio buttons can be selected independently.

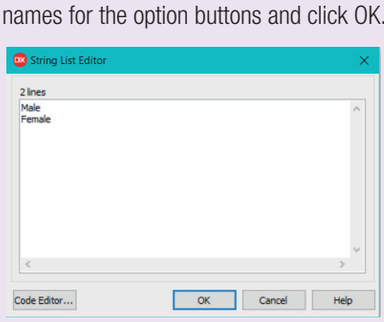


Example 5.7 Creating a group of RadioButtons

1. Select the *TRadioGroup* from the *Standard Palette* and place it on the form. The *TRadioGroup* serves as a container for the radio buttons.
2. Set the *Name* property of the *TRadioGroup* to *rgpGender*
3. Change the *TRadioGroup* caption to an appropriate name for the group of buttons, for example, 'Gender'.
4. Select the *TRadioGroup* component on the form. In the *Items* property in the *Object Inspector*, click on the ellipse (…).



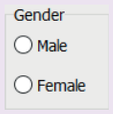
5. In the *StringList Editor*, enter the names for the option buttons and click OK.



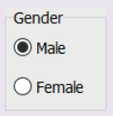
Example 5.7 Creating a group of RadioButtons *continued*

6. Setting the *ItemIndex* property of the *TRadioGroup*:

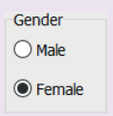
   a. If the *ItemIndex* is set to -1, then none of the radio buttons will be selected at runtime.



   b. If the *ItemIndex* is set to 0, then the first button is already selected at runtime.

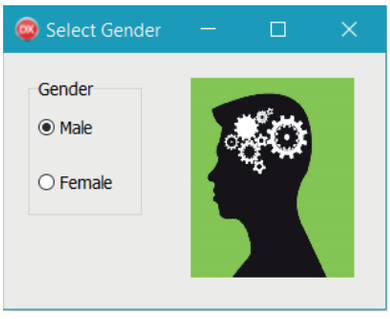


   c. If the *ItemIndex* is set to 1, then the second button is already selected at runtime.

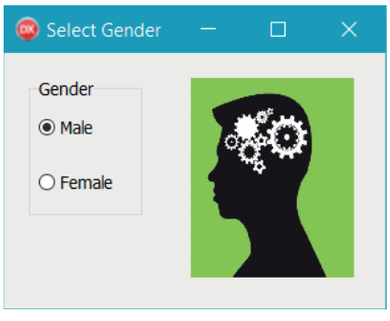


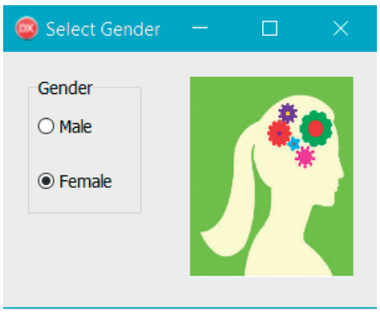
Guided Activity 5.1 Creating a group of RadioButtons

5.1.1 Open the **Gender_p** project from the 05 – Select Gender folder.



5.1.2 Create an *Onclick* event for the *RadioButton* group *rgpGender* so that when the Male option is selected, the boy image is displayed. When the female option is selected, then the girl image is displayed.





Guided Activity 5.1 Creating a group of RadioButtons *continued*

5.1.3 To create the *OnClick* event on the *rgpGender* group, double click on the *rgpGender* component. In the *rgpGender* event handler write the code (to load a picture into the image component during run time) for each option.

```
procedure TfrmGender.rgpGenderClick(Sender: TObject);
  begin
    if rgpGender.ItemIndex = 0 then
      imgGender.Picture.LoadFromFile('Male.bmp');

    if rgpGender.ItemIndex = 1 then
      imgGender.Picture.LoadFromFile('Female.bmp');
  end;
```

Image component — Property — Picture file name. Add path name if file resides in different folder

```
imgGender.Picture.LoadFromFile('File name')
```

5.1.4 Save and run your project.

Did you know

The *LoadFromFile* method allows you to load the contents of a file (the image) into the image component during run time.

Take note

- Previously, you set the ItemIndex property manually
- Now we are going to check the value of the ItemIndex property. The value of the ItemIndex property indicates which option is selected by the user.
- If the user selects the Male option, then ItemIndex will have the value 0 and if the Female option is selected, then the ItemIndex has a value 1
- We can test which option is selected by comparing ItemIndex against 0 or 1. Example: in the first IF-THEN statement the condition is: rgpGender.ItemIndex = 0. This tests if the current value of ItemIndex is equal to 0.
- If the Male option is selected, then the test rgpGender.ItemIndex = 0 evaluates to true and the Male image is loaded to the imgGender image component.
- Similarly, the female image is loaded when the Female option is selected

Activity 5.6

5.6.1 All library users pay a fee of R35.00 per month for the library services. To encourage reading amongst learners, they are offered the following discounts.

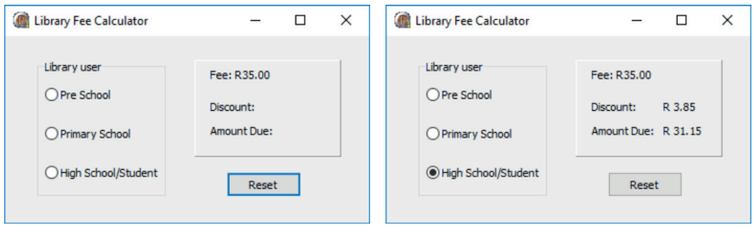
- Pre-schoolers get a discount of 15%
- Primary school learners get a discount of 13%
- High school learners and students get a discount of 11%.

Activity 5.6 *continued*

a. Create a project, **Discount_p** to calculate the fees payable as follows:

- When the RadioButton in the RadioGroup is clicked, calculate the discount. Display the discount and amount due formatted as currency.
- Write code for a [Reset] button that will reset the RadioGroup so that no option is selected, and clear the labels so that nothing displays in the labels.

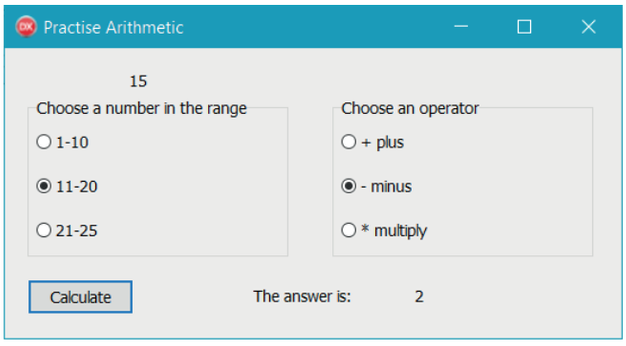
Use the interface shown below as a guide.



5.6.2 Open the **Arithmetic_p** project located in the 05 – Practise Arithmetic folder.

This program allows a user to practice their mental arithmetic. The RadioGroups allow a user to choose the number range and operator that they would like to practice.

Run the program and correct any errors you may encounter when the program executes.



Take note

Use the code

```
<labelname>.
caption := '';
```

This places an empty string in the caption of the label – and displays nothing!

UNIT

5.3 Boolean operators

To create compound Boolean expressions, you can use logical operators. These are:

- AND Logical Operator
- OR Logical Operator
- NOT Logical Operator

AND LOGICAL OPERATOR

The AND operator is used to test two conditions using the format:

(condition1) AND (condition2)

Both conditions must evaluate to TRUE for the result to be true. If either condition1 or condition2 is FALSE, then the result will be false.

Example 5.8

If A = 5 and B = 7 then:

BOOLEAN EXPRESSION	CONDITION1	CONDITION2	RESULT
(A > 2) AND (B <= 7)	True	True	True
(A > 2) AND (B > 7)	True	False	False
(A = 3) AND (B > 5)	False	True	False
(A = 3) AND (B > 7)	False	False	False

OR LOGICAL OPERATOR

The OR operator is used to test two conditions in the format:

(condition1) OR (condition2)

Both conditions must evaluate to FALSE for the result to be false. If either condition1 or condition2 is TRUE, then the result will be true.

Example 5.9

If A = 5 and B = 7 then:

BOOLEAN EXPRESSION	CONDITION1	CONDITION2	RESULT
(A > 2) OR (B <= 7)	True	True	True
(A > 2) OR (B > 7)	True	False	True
(A = 3) OR (B > 5)	False	True	True
(A = 3) OR (B > 7)	False	False	False

NOT LOGICAL OPERATOR

The NOT operator negates the result of the condition in the format:

NOT(condition)

If the condition evaluates to TRUE, then the NOT(condition) evaluates to false. If the condition evaluates to FALSE, then the NOT(condition) evaluates to true.

Example 5.10

If A = 5 and B = 7 then:

BOOLEAN EXPRESSION	CONDITION	RESULT
NOT(A > 2)	True	False
NOT (B > 7)	False	True

THE ORDER OF PRECEDENCE

In Chapter 4 you learned about the order of precedence of mathematical operators. Similarly, Boolean operators also have order of precedence. The order of precedence from highest to lowest precedence is as follows:

OPERATOR/S	LEVEL
NOT	1 (highest level of precedence)
*, /, DIV, MOD, AND	2
+, -, OR	3
=, <>, <, >, <=, >=	4

Example 5.11 The order of precedence

Assume the value of the variables is as follows:

Gender = 'F', Age = 15 and Sport is 'Netball'.

Evaluate the following Boolean expressions:

(Gender = 'F') OR (Age > 10) AND (Sport = 'Cricket')

= TRUE OR TRUE AND FALSE

= TRUE OR FALSE

= TRUE

Activity 5.7

5.7.1 Suppose the values of the variables are as follows:

A = 1; B = 2; C = 5.5; D = 8.1; Letter = 'S'; Test = True;

Evaluate the result of each of the following statements:

a. (B > 0) **AND** (B <5)

b. (C < D) **OR** (Letter = 's')

c. (B mod 2 = 0) **AND NOT** (Test)

d. Test **AND** (Letter = 'S')

Activity 5.7 *continued*

5.7.2 Study the code snippet below and then answer the question below. For each of the Boolean expressions (a to e), indicate whether the expression is valid or invalid. If the expression is valid, indicate the result. If the expression is invalid, correct the expression.

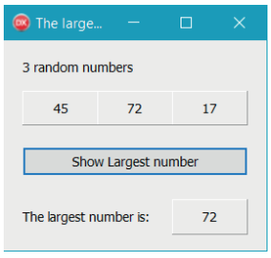
```
VAR
   rValReal : REAL;
   iValInt : INTEGER
   cCharacter : Char;
   sString : String;
   bTest : Boolean;
```

a. (rValReal = 0) **OR** (iVallnt < 100)
b. (rValReal > 0 **AND** < 100)
c. **NOT**(bTest)
d. sString = 50.1
e. Test **OR NOT** (cCharacter = J)

5.7.3 Write down the output of the following Boolean combinations.

a. True AND True
b. True OR False
c. False OR NOT False
d. True OR False OR False
e. (True OR False) AND True
f. NOT (False OR True) AND True
g. (5 > 4) AND ('3' = 3)
h. (Sqrt(16) = 4) OR (15 < 10 + 5)
i. NOT ('Hello' = 'hello') AND (4 >= Round(3.5))

Guided Activity 5.2 Generate three random numbers and determine the largest of three numbers



5.2.1 Open the project **Largest_p** located in the 05 – Largest Number folder.

5.2.2 Create an *OnClick* event for the [Show Largest number] button.

- Generate three random numbers iNum1, iNum2 and iNum3 in the range 34 to 86.
- Display the iNum1, iNum2 and iNum3 in lblNum1, lblNum2 and lblNum3 respectively
- Determine the largest of the three numbers and display the largest number in lblLargest.

```
procedure TForm1.btnShowLargestClick(Sender: TObject);
var
  iNum1, iNum2, iNum3 : Integer;
begin
  Randomize;
  iNum1 := random(86 - 34 + 1) + 34;
  iNum2 := random(86 - 34 + 1) + 34;
  iNum3 := random(86 - 34 + 1) + 34;
  lblNum1.Caption := IntToStr(iNum1);
  lblNum2.Caption := IntToStr(iNum2);
  lblNum3.Caption := IntToStr(iNum3);
  if (iNum1 >= iNum2) AND (iNum1 >= iNum3)then
    lblLargest.Caption := IntToStr(iNum1);
  if (iNum2 >= iNum1) AND (iNum2 >= iNum3)then
    lblLargest.Caption := IntToStr(iNum2);
  if (iNum3 >= iNum1) AND (iNum3 >= iNum2)then
    lblLargest.Caption := IntToStr(iNum3);
end;
```

5.2.3 Save and run your program.

Activity 5.8

5.8.1 Run the program **BooleanPractise_p.exe** located in the 05 – IT Awards folder.

This app allows learners to enter their marks for IT Practical and Theory exams. The program then displays the IT award based on the marks entered. Run the program and test it with various combinations of marks. Copy and complete the table below. Fill in the output and the Boolean expressions that led to the results you observe.

SPECIAL AWARD	OUTPUT	BOOLEAN EXPRESSION
Gold medal		
Silver medal		
Bronze medal		
Other		

5.8.2 Write a program, **DiceGuess_p** where the computer rolls two die without showing the results, then allows the user to guess the value of the two die. Inform the user if:

- neither guess was correct
- one guess was correct
- both guesses were correct.

5.8.3 You must develop an application, **BMICalculator_p** that can be used to calculate a user's Body Mass Index (BMI). The formula for BMI is the person's weight (or mass) in kilogram, divided by the square of their height (in metres).

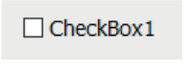
$$\text{BMI} = \frac{\text{mass (kg)}}{\text{height}^2 \text{ (m)}}$$

- Based on the BMI, the application should state whether the user is underweight (BMI under 18), at optimal weight (BMI between 18 and 25), or overweight (BMI over 25).
- Develop an algorithm for the application and draw a flowchart to represent the algorithm.
- Write and run the application.

DECISION MAKING

CHECK BOXES

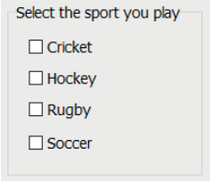
Check boxes are used to give a user a YES/NO choice, for example, whether a person is an athlete or not.



To create a single check box:

- select *TCheckBox* component from *Standard Palette* and place the component on the form
- change the *Caption* property to indicate the purpose of the checkbox
- change the *Name* property to a meaningful name starting with the prefix cbx, for example, cbxSoccer
- if you want the checkbox to be ticked as default, set the Checked property to true.

Unlike RadioButtons, you can select as many check boxes as you want. In the example alongside, a person can select all the sporting codes that he or she participates in.



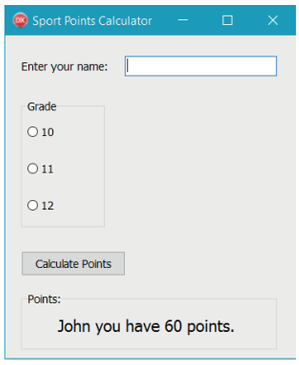
The *TGroupBox* component allows you to place check boxes in a neat way. If you need to move the check boxes to another location on the form, you simply need to move the *TGroupBox*.

To create a group of checkboxes as shown on the previous page:

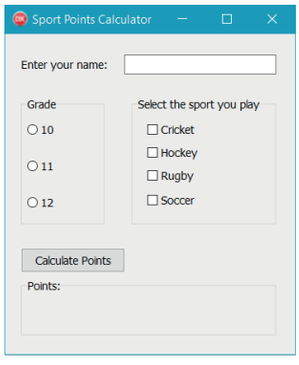
- Select the *TGroupBox* from the *Standard Palette* and place on the form. Set the *Caption* property of the *TGroupBox* to prompt the user to make a selection from the checkboxes
- Select *TCheckBox* component from *Standard Palette* and add to the *TGroupBox* component. Change the *Caption* property of the checkbox to indicate it purpose. Add as many checkboxes as you need.

Guided Activity 5.3

5.3.1 Open the **SportPoint_p** project located in the 05 – Sport Points folder.



5.3.2 Modify the form to display as shown:



- Add a TGroupBox component from the Standard Palette
- Set the Caption property of the TGroupBox as 'Select the sport you play'
- Add four checkboxes on the TGroupBox
- Change the Name property and Caption of the checkboxes as follows:

NAME	CAPTION
chbCricket	Cricket
chbHockey	Hockey
chbRugby	Rugby
chbSoccer	Soccer

5.3.3 Create an *OnClick* event for [Calculate Points] button to calculate points as follows:

- if the learner plays cricket, he or she earns 10 points
- if the learner is in Grade 10 and plays rugby or soccer, he or she earns 50 points.

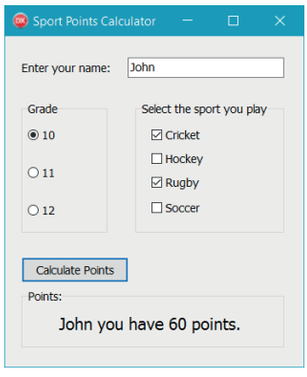
Guided Activity 5.3 *continued*

5.3.4 The code for the event handler *Calculate Points* is:

```
procedure TForm1.btnCalculatePointsClick(Sender: TObject);
var
  sName : String;
  iPoints : Integer;
Begin
  iPoints := 0;
  //get inputs
  sName := edtName.Text;

  if chbCricket.Checked then
    iPoints := iPoints + 10;

  if (rgpGrade.ItemIndex = 0) and (chbRugby.Checked or chbSoccer.Checked) then
    iPoints := iPoints + 50;
  //insert your code to questions ... here

  lblPoints.Caption := sName + ' you have ' + IntToStr(iPoints) + ' points.';
end;
```

- use the *Checked* property of the checkbox to determine whether the chbCricket is checked(selected) or not: chbCricket.Checked
- chbCricket.Checked returns a value true if the checkbox is selected, otherwise it returns false
- the statement to award points if the learner plays cricket:
  - if chbCricket.Checked then // This is the same as if chbCricket.Checked = true then
    iPoints := iPoints + 10;
- the statement to award points for the learner who is in Grade 10 and plays rugby or soccer is:
  - if (rgpGrade.ItemIndex = 0) and (chbRugby.Checked or chbSoccer.Checked) then
    iPoints := iPoints + 50;

5.3.5 Save and run your program.



Activity 5.9

5.9.1 Open the **SportPoint_p** project used on the previous page and add code to the *Calculate Points* event handler to calculate points using the following criteria:

- if a learner plays any sport he or she earns 10 points.
- if a learner is in Grade 11 or Grade 12 and plays hockey, he or she earns 5 points.

5.9.2 Save and run the project.

IF STATEMENT AND THE SHOW MESSAGE DIALOG BOX

Until now you have mostly used the *Label* component to display information. You can also use the *ShowMessage* Dialog Box to display information.

The syntax of a *ShowMessage* statement is as follows:

```
ShowMessage(sMessage);
```

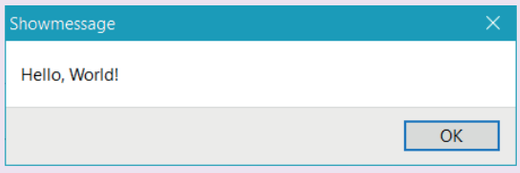
It is important to note the following:

- the *ShowMessage* statement is a stand-alone statement
- It displays *sMessage*, which is of string data type
- Upon execution of the ShowMessage statement, a ShowMessage Dialog Box (a popup window) appears and displays the string sMessage.
- If you want to display strings with multiple lines, then you need to use the #13 escape character to move the cursor to the next line.

Example 5.12

```
ShowMessage ('Hello, World!');
```

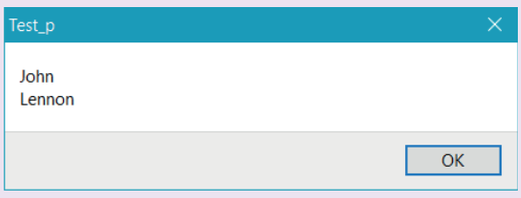
This statement will result in a *ShowMessage* Dialog Box displaying the text 'Hello, World!'



Example 5.13

```
ShowMessage('John'+#13+'Lennon');
```

This statement will result in a *ShowMessage* Dialog Box displaying the text 'John Lennon' on two separate lines.



Note that in both examples the title of the ShowMessage Dialog box refers to the project name.

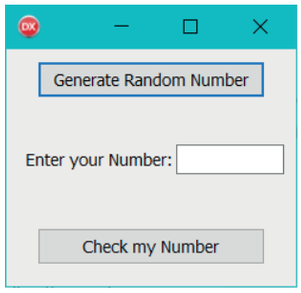
Guided Activity 5.4

In the *I Can Guess Game*, a random number is generated. The user must guess what the number is.

Display the following messages in *ShowMessage* Dialog boxes based on the user's guess:

- 'Number Too Low' if the guess is less than the random generated number
- 'Number Too High' if the guess is greater than the random generated number
- 'Correct' and the random number if the guess is equal to the random generated number

5.4.1 Open the project **GuessNumber_p.dproj** located in the 05 – I Can Guess folder.



5.4.2 Since the random number must be accessed from both the [Generate Random Number] button event handler and the [Check my Number] button event handler, the random number iNum must be declared globally:

```
var iNum : integer;
```

5.4.3 Add *OnClick* events for:

- [Generate Random Number] button: A random number in the range 0 to 100 is generated. The code to generate the random number is:

```
randomize;
iNum := random(101);
```

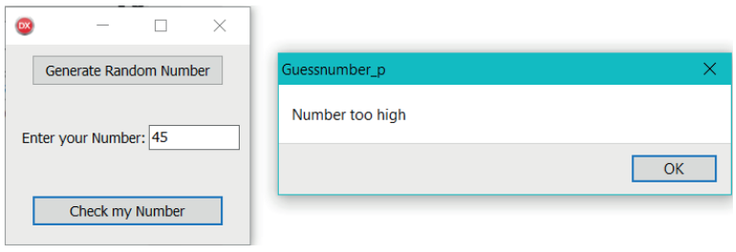
- [Check my Number] button: Read the user's guess from the edtGuess component and store the value in iGuess. Compare the iGuess number to the iNum value. Depending on the outcome of the comparison, show appropriate messages in a *ShowMessage* Dialog box.

```
iGuess := StrToInt(edtGuess.text);
if iGuess > iNum then
  ShowMessage('Number too high');
If iGuess < iNum then
  ShowMessage('Number too low');
If iGuess = iNum then
  ShowMessage('Correct'+' '+IntToStr(iNum)');
```

Guided Activity 5.4 *continued*

5.4.4 Save and run the project.

Here is an example of a sample run.



Activity 5.10

5.10.1 Open the application you wrote for the Guided activity above.

5.10.2 Add code that will allow you to keep track of the number of guesses.

If the number of guesses is more than 10 then the program must terminate.

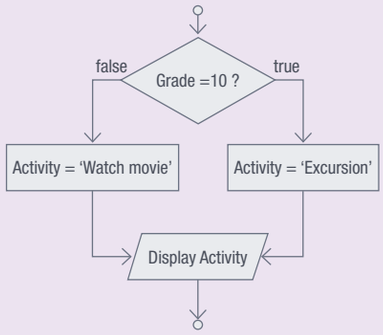
UNIT

5.4 If-then-else statement

Sometimes you may want something to happen if a condition is met, and something else to happen if the condition is not met. Let's look at an example:

Example 5.14

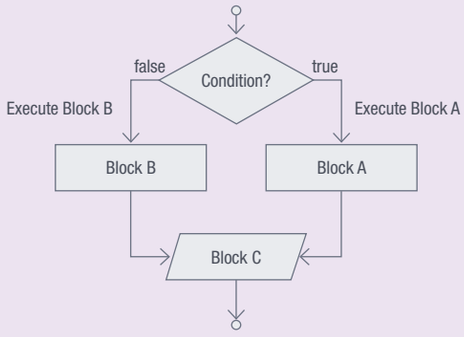
All Grade 10 learners are invited to go on an excursion. Learners who are not in Grade 10 will stay at school and watch a movie. The condition stipulates that if a learners is in Grade 10, then the learner will go on an excursion, else the learner will watch a movie.



The IF-THEN-ELSE statement executes a statement/s following the THEN-keyword when the condition is true; and executes a statement/s following the ELSE-keyword when the condition is false.

Example 5.15

In the fl ow chart below, block A represents the statement/s following the THEN keyword. Block B represents the statement/s following the ELSE keyword. Both paths execute Block C.



SYNTAX OF IF-THEN-ELSE STATEMENT

Here is the Delphi syntax of an IF-THEN-ELSE statement if one statement follows the THEN- and ELSE-keywords:

```
If <condition> THEN
    <statement1>
ELSE
     <statement2>;
```

- the THEN and ELSE keywords are not followed by a semicolon
- the statement before the ELSE statement does not have a semi-colon
- if more than one statement appears in the THEN or ELSE part, then the statements must appear in a BEGIN…END block. For example:

```
If <condition> THEN
begin
    <statement1>;
     …
   <statement4>;
end
ELSE
begin
   <statement5>;
   <statement6>;
…
end;
```

- a single statement does not need to appear in a BEGIN…END block

Guided Activity 5.5 Read the name and mark of a learner

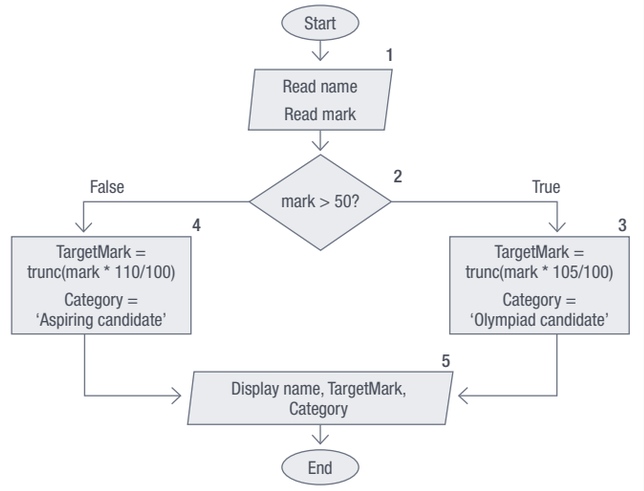
Create a program that will read the name and mark of a learner for different tests.

- If the mark is greater or equal to 50 then:
  - calculate a target mark for the learner's next test by increasing the mark by 5%
  - set the category to 'Olympiad candidate'
- If the mark is less than 50 then:
  - calculate the target mark for the learner's next test by increasing the mark by 10%
  - set the category to 'Aspiring candidate'.

Guided Activity 5.6 Read the name and mark of a learner *continued*

5.6.1 Here is the algorithm and flowchart for this program:

```
Read name,mark
If mark >= 50 then
 begin
   TargetMark = trunc(mark * 105/100)
   Category = 'Olympiad candidate'
 end
else
 begin
   TargetMark = trunc(mark * 110/100)
  Category = 'Aspiring candidate'
 end
Display (Name,TargetMark, Category)
```



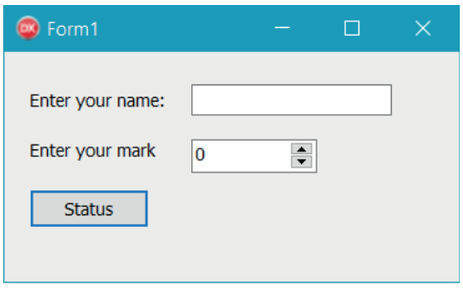
5.6.2 Trace through the flowchart using the inputs John for name and 44 for the mark.

BOX NUMBER	NAME	MARK	TARGETMARK	CATEGORY	MARK >= 50?	OUTPUT
1	John	44				
2					False	
4			48	Aspiring candidate		John 48 Aspiring candidate
5						
Stop						

5.6.3 Complete the trace table below for the flowchart using the given inputs: Mary and 57 for the mark.

BOX NUMBER	NAME	MARK	TARGETMARK	CATEGORY	MARK >= 50?	OUTPUT

5.6.4 We are now ready to code. Open the **Olympiad_p** project.



Guided Activity 5.6 Read the name and mark of a learner *continued*

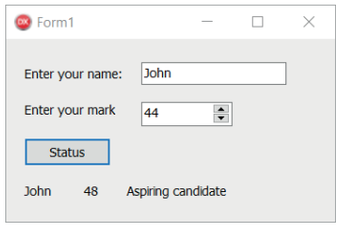
5.6.5 Create an *OnClick* event for the [Status] button to determine the target mark and category and to display the name, target mark and category of a learner. The code in the event handler for the [Status] button is:

```
procedure TForm1.btnStatusClick(Sender: TObject);
var
  sName : String;
  iMark : integer;
  sCategory : String;
  iTargetMark : integer;
begin
   sName := edtName.Text;
   iMark := sedMark.Value;

   if iMark > 50  then
   begin
      sCategory := 'Olympiad candidate';
      iTargetMark := trunc(iMark * 105/100);
   end
   else
   begin
   sCategory := 'Aspiring candidate';
      iTargetMark := trunc(iMark * 110/100);
   end;
  lblStatus.Caption := sName + #9 + IntToStr(iTargetMark) + #9 + sCategory;
end;
```

Take note

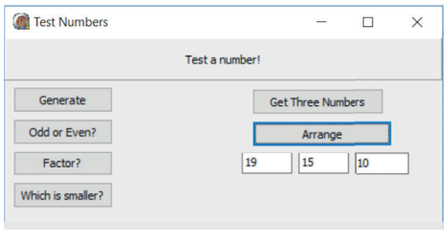
As with the #13 escape character that you used to place text on separate lines, you can use the #9 tab character to place text in columns.

5.6.6 Save and run the project.



Activity 5.11

5.11.1 Open the incomplete **TestNumbers_p** project located in the 05 – Test Numbers folder and complete the program as follows:



- *BtnGenerate*: Complete the code for *btnGenerate* to save two random numbers between 1 and 35 in appropriate variables.
- *BtnOddEven*: Write code for this button to test whether a number is an odd or even number. Display a message with your answer.
- *BtnFactor*: Test if the first number is a factor of the second number. Display a message with your answer.
- *BtnSmaller*: Compare the two numbers and display the smaller one.
- *BtnThreeNum*: Write code to place three random numbers between 1 and 20 in edtANum, edtBNum and edtCnum respectively.
- *btnArrange*: Write code to place the three numbers in descending order in the three edit boxes.

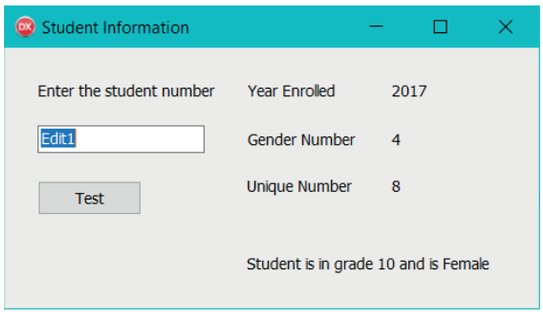
Activity 5.16 *continued*

5.11.2 ABC school assigns every student that enrols with a unique four digit number, for example, 1748.

This number is made up as follows:

- the first two numbers represent the year in which the learner enrolled in any school in Grade 8.
- the third number represents the gender of the learner, 4 for female and 8 for males.
- The last number is a random number between 0 and 9.

a. Create a new Delphi program with an interface similar to the image to the right.



The user will enter his student number in the *Edit* box.

b. Use the **StudentInformation_p** project located in the 05 – Student Information folder. Write the code for *btnTest* to do the following:

- use the given variables
- use div and mod functions to separate the number into the unique number, gender number and the year enrolled
- display these values as shown in the example above
- use an IF statement to determine the gender of the student and save either 'Male' or 'Female' in a string variable.
- the code to determine the grade is given. Read the code comments to understand what this code does
- display all the values in a single string (as shown in the example above).

c. Test your program using the values 1748 and 1853.

5.11.3 Write an applicaton, **Schoolbag_p** that will calculate the approximate weight of your school bag and categorise it according to three levels: extra load (more than 6 kg), manageable (between 3 and 6 kg, inclusive) and lightweight (less than 3 kg). For each book in the bag:

- The user clicks on a RadioButton, in a RadioGroup, to select the book size.
- Click on [Add Book] button to record the book weight according to the size as follows:
  - large books weigh 900 g on average
  - medium books weigh 400 g on average
  - small books weigh 150 g on average.
- Write code for *BtnGo* to display the following Information:
  - the total number of books recorded
  - the weight of the bag
  - the category of the bag.

5.11.4 Open the Diving competition activity you completed in Activity 4.7 (use the **Scoring_p** project from the 04 - Scoring folder) and expand the program as follows:

a. Add two more *Edit* boxes with appropriate labels to record the scores of two more judges, so that five judge's scores are recorded.

b. Change the code for the [Final Score] button to exclude the highest and the lowest score and calculate the average of the other three scores

Activity 5.12

Create a program that generates three random numbers between 1 and 100. Display all three numbers to the user. If the user clicks the [Largest] button, display the value of the largest random number. If the user clicks the [Smallest] button, display the value of the smallest number.

For this application:

5.12.1 Create an algorithm for the [Smallest] button.

5.12.2 Create a flow chart for the [Largest] button.

5.12.3 Create a trace table for the [Smallest] button if the random numbers are 39, 41 and 15.

5.12.4 Create a trace table for the [Largest] button if the random numbers are 72, 14, and 11.

5.12.5 Create the program in Delphi and run it.

UNIT

5.5 Nested if-then statements

A nested IF-THEN statement occurs when one conditional statement is placed inside another conditional statement. By doing this your program first checks if the outer condition has been met before looking at the inner conditional statement.

DELPHI SYNTAX OF THE NESTED IF STATEMENT

The syntax for the nested IF statement is:

```
IF <condition1> THEN
    IF <condition2> THEN
    <statement1>
     ELSE
        begin
            <statement5>;
            <statement6>;
         …
        end;
```

The code snippet below shows an example of a nested-if statement.

Nested-IF example

```
if iValue > 0 then
begin
  if iValue < 100 then
     ShowMessage('Number is between 0 and 100')
  else
    ShowMessage('Number is 100 or above');
end;
```

Take note of the following for a nested IF statement:

- the outer conditional statement (iValue > 0) is tested first. If this condition is true, the inner conditional statement (iValue < 100) is tested next
- if the conditional statement (iValue < 100) is true, then the ShowMessage('Number is between 0 and 100') statement is executed, else the ShowMessage('Number is 100 or above'); is executed.

Work through the following example to help you understand.

Example 5.16

A company is handing out bags using the following criteria:

- if a person is a male and drinks coke then he qualifies for a bag; otherwise no bag
- if a person is female and drinks fanta then she qualifies for a bag; otherwise no bag.

Nested-if example

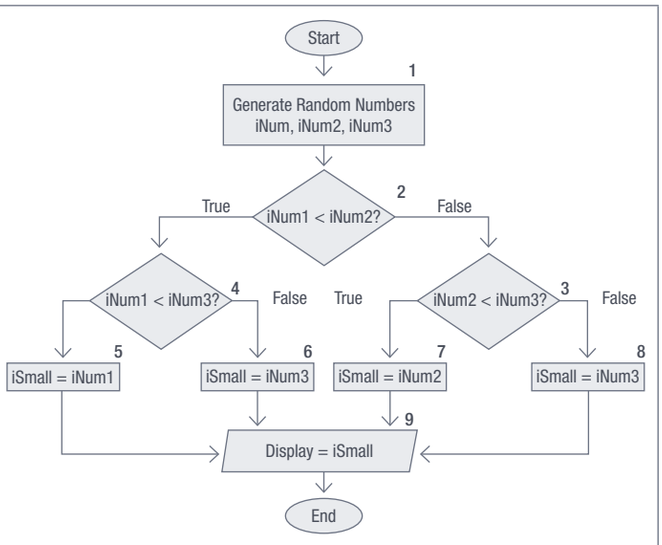
```
if Gender = 'Male' then
begin
  if Drink = 'Coke' then
     ShowMessage('You get a bag')
  else
    ShowMessage('No bag');
end;
if Gender = 'Female' then
begin
   if Drink = 'Fanta' then
     ShowMessage('You get a bag')
  else
    ShowMessage('No bag');
end;
```

Guided Activity 5.7

Generate three random integers Num1, Num2 and Num3 in the range 0 to 100 and display the smallest of the three numbers. You may assume that the three numbers are different.

5.7.1 The algorithm and flowchart:

```
Generate three random number a, b, c
in the range 0 to 100
Display Num1, Num2 and Num3
If Num1 < Num2 then
  If Num1 < Num3 then
     small = Num1
  else
     small = Num3
Else
  If Num2 < Num3 then
     small = Num2
  else
     small = Num3
Display small
```



5.7.2 Assume that the values 10, 25 and 12 were randomly generated for Num1, Num2 and Num3 respectively. Trace through the flowchart using these values.

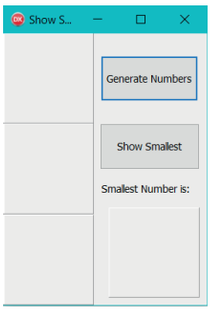
BOX NUMBER	NUM1	NUM2	NUM3	SMALL	NUM1 < NUM2?	NUM1 < NUM3?	NUM2 < NUM3?	OUTPUT
1	10	25	12					
2					True			
4						True		
5				10				
9								10
Stop								

5.7.3 Trace through the flowchart using the randomly 25, 12 and 32 for Num1, Num2 and Num3 respectively.

BOX NUMBER	NUM1	NUM2	NUM3	SMALL	NUM1 < NUM2?	NUM1 < NUM3?	NUM2 < NUM3?	OUTPUT
1	25	12	32					
2					False			
4						True		
5				12				
9								12
Stop								

Guided Activity 5.7 *continued*

5.7.4 Open the **SmallestOfThree_p** project located in the 05 – Smallest of Three folder.

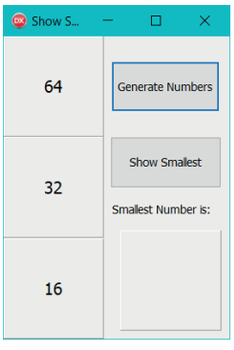


a. Because Num1, Num2 and Num3 will be used by both the [Generate Numbers] button and the [Show Smallest] buttons, they have to be declared globally.

```
var
  Form1: TForm1;
  iNum1, iNum2, iNum3: integer;
```

b. Create an *OnClick* event for the [Generate Numbers] button to generate the three random numbers Num1, Num2 and Num3. Display the randomly generated numbers in lblNum1, lblNum2 and lblNum3.

Here is the event handler code for the [Generate Numbers] button.



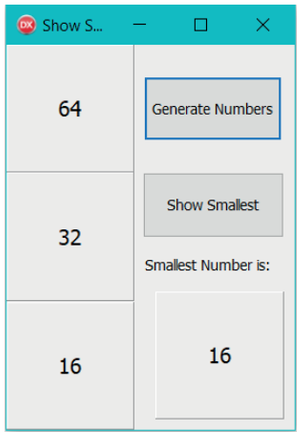

```
procedure TForm1.btnGenerateNumbersClick(Sender: TObject);
begin
  Randomize;
  iNum1 := random(101);
  iNum2 := random(101);
  iNum3 := random(101);
  lblNum1.Caption := IntToStr(iNum1);
  lblNum2.Caption := IntToStr(iNum2);
  lblNum3.Caption := IntToStr(iNum3);
end;
```

Guided Activity 5.7 *continued*

c. Create an *OnClick* event for the [Show Smallest] button to determine the smallest number of the three numbers and display the smallest number in the *lblSmall*.

```
procedure TForm1.btnShowSmallestClick(Sender: TObject);
var iSmall: integer;
begin
    if iNum1 < iNum2 then
      if iNum1 < iNum3 then
        iSmall := iNum1
        else
        iSmall := iNum3
    else
      if iNum2 < iNum3 then
        iSmall := iNum2
        else
        iSmall := iNum3;
    lblSmall.Caption := IntToStr(iSmall);
end;
```

d. Save and run the program.



Activity 5.13 More number games!

5.13.1 Create a new Delphi program that will do the following:

a. Generate a random number between 1 and 100.

b. If the number is larger than 50, determine whether the number is an odd or even number.

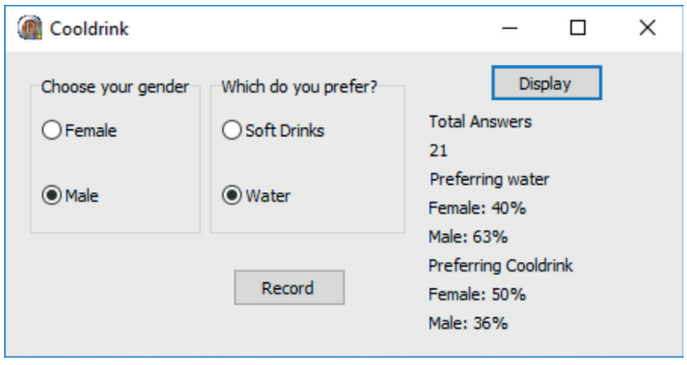
c. If the number is smaller than 50, determine whether the number is an odd or even number.

5.13.2 Display the number of even numbers larger than 50 and the number of odd numbers smaller than 50 in a *ShowMessage* dialog box.

Activity 5.14

Complete the application to help a tuck shop decide how to order the drinks they sell. They want to know what the learners prefer – cooldrink or cold water – and also whether there is a difference between what the girls and boys prefer.

Open the **Prefer_p** project from the 05 – Prefer Cooldrink folder and study the interface. The application should do the following:



- Determine whether a learner is male or female.
- Determine whether that learner prefers water or cooldrink.
- Keep track of the total number of learners that answer.
- Calculate the percentage of females who prefer water.
- Calculate the percentage of females who prefer cooldrink.
- Calculate the percentage of males who prefer water.
- Calculate the percentage of males who prefer cooldrink.
- Display the percentages in the labels provided as shown.

a. Draw a flowchart to help you complete this App.
b. Complete the code according to your flowchart and run the App.

UNIT

5.6 Case statements

Another type of decision-making structure in Delphi is the CASE statement. Instead of using a sequence of cascading IF-THEN-ELSE-IF statements for decision making, the CASE statement provides a tidy way of dealing with decision-making. The cascading IF-THEN-ELSE-IF statement allows you to execute a block code amongst different alternatives. If you are checking on a value of a **single** variable of the IF-THEN-ELSE-IF statement, it is better to use the CASE statement.

The CASE statement uses the following syntax:

CASE statement syntax

```
CASE <variable> OF
  value1 : statement1;
  value2 : statement2;
  value3 : statement3;
 ELSE
     Statement4;
end;
```

Take note of the following:

- start with the keyword CASE followed by a <variable> followed by the keyword OF. There is no semicolon after the OF keyword
- the CASE statement does not have a begin but has an END
- the <variable> can be the name of the any variable of type integer or character
- different cases: Value1, Value2 and Value3 refer to the cases against which <variable> will be compared and must be of the same data type as <variable>
- each case is followed by a colon
- the statements following the colon indicates what code must be executed for that case. Example for case Value2, Statement2 must be executed.
- if more than one statement needs to be executed in a case, it must be in a Begin... End block
- a case can be represented by:
  - an integer : 5
  - a character : 'A'
  - range : 3..5
- if the same action is required for more than one case then cases can be grouped as follows:

```
Case cLetter Of
   'a','e','i','o','u':   ShowMessage ('Vowel');          // grouped cases are separated by comma
    Else
                          ShowMessage ('Not a vowel')
End;
```

- when a match is found during the comparison, control of the program passes to that case and the code of that case is executed and the CASE statement is exited
- the ELSE statement is optional. It is used to provide a default if none of the cases match the <variable>.

Guided Activity 5.8 Cooldrink selection

A user makes a choice from a RadioButton group, and depending upon the choice he or she makes, the following is displayed:

RADIOBUTTON	MESSAGE
Coco Cola	You selected Coca Cola
Creme Soda	You selected Creme Soda
Fanta Grape	You selected Fanta Grape

5.8.1 Open the **CooldrinkSelection_p** project from the 05 – Cooldrink Selector folder.

5.8.2 The code below uses nested IF-THEN-ELSE to determine the cool selected. Note that the nesting is always in the ELSE part.

Nested IF-THEN-ELSE statement

```
if iSelectedItem = 0 then
  ShowMessage('You selected a Coca Cola')
else
  if iSelectedItem = 1 then
     ShowMessage('You selected a Creme Soda')
  else
     if iSelectedItem = 2 then
        ShowMessage('You selected a Fanta Grape')
     else
    ShowMessage('You have not selected a cooldrink');
```

5.8.3 Now we are going to write the same code using a CASE statement. Create an *OnClick* event for the [Select] button. Here is the code in the event handler:

CASE statement

```
procedure TForm1.btnSelectClick(Sender: TObject);
var
  iSelectedIndex : integer;
begin
   iSelectedIndex := rgpCooldrinks.ItemIndex;
case iSelectedIndex of
    0 : Showmessage('You selected Coca Cola');
    1 : Showmessage('You selected Creme Soda');
    2 : Showmessage('You selected Fanta Grape');
    else
    ShowMessage('You have not selected a cooldrink');
   end;
end;
```

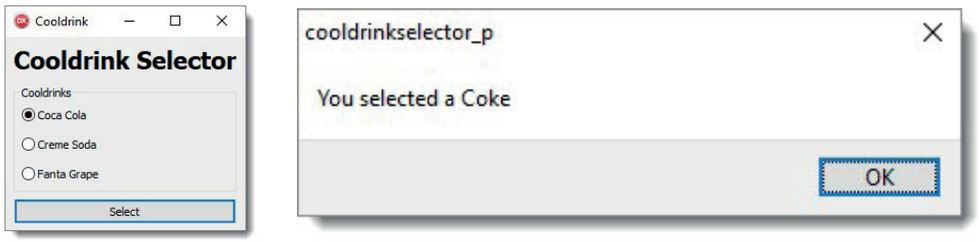
Did you know

The CASE statement can use integers and character variables, but not string or double variables.

The CASE statement is a lot easier to read and use. It is also easier to expand or add additional values to.

Guided Activity 5.8 Cooldrink selection *continued*

5.8.4 Save and run the project.



The CASE statement below shows:

- Checking if a variable falls within a range of values.
- Checking if a variable has one of many individual values.
- Combining a range of values with individual values.

The code snippet below shows an example of each of these techniques.

CASE complex comparisons

```
CASE iNumber OF
  1..99 : ShowMessage('Number is between 1 and 100');
  iMin, iMax : ShowMessage('Number is the minimum or maximum');
  -1, 101..9999 : ShowMessage('ERROR: Number is illegal');
ELSE
  ShowMessage(‘Number does not meet any requirements);
end;
```

As this example shows, the CASE statement can also contain an ELSE condition, which is activated when the variable does not match any of the values.

To try out the CASE statement, work through the following example:

Example 5.17 Percentage to symbol converter

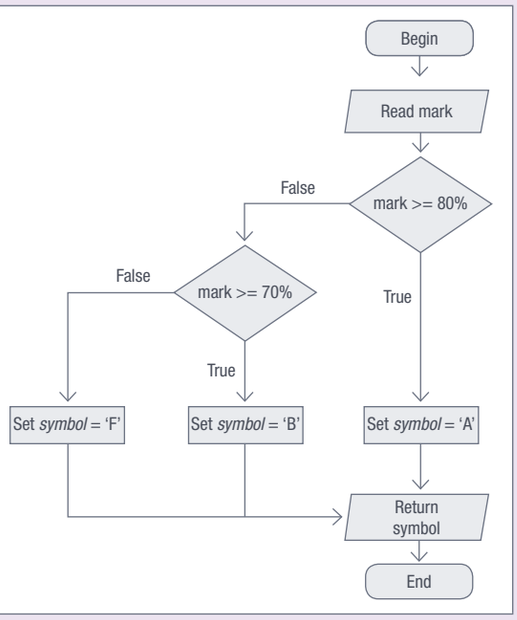
Learners receive a percentage mark, as well as a symbol for certain educational courses in South Africa. These symbols might be allocated as follows:

MARK	SYMBOL
80% – 100%	A
70% – 79%	B
0% – 69%	F

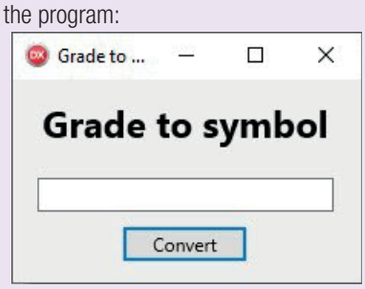
The following pseudo code and flow chart were created to plan this program that converts the mark into a symbol:

Example 5.17 Percentage to symbol converter *continued*

```
BEGIN
SET mark ← user input
IF mark >= 80%
   SET symbol ← 'A'
ELSE
   IF mark >= 70%
      SET symbol ← 'B'
      ELSE
         SET symbol ← 'F'
RETURN symbol
END
```



Use the pseudo code and flow chart, plus the instructions below to create the program:



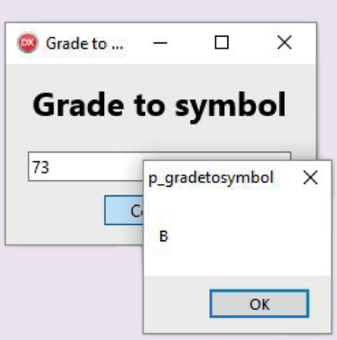
1. Create the following user interface.
2. Create an *OnClick* event for the [Convert] button.
3. Create a local integer variable called *iPercentage*.
4. Obtain the value of *iPercentage* from the text box. Remember to convert the text in the edit box to an integer.
5. Create the following CASE statement to display the symbol.

Grade to symbol CASE statement

```
case iPercentage of
  0..69 : ShowMessage('F');
  70..79 : ShowMessage('B');
  80..100 : ShowMessage('A');
else
  ShowMessage('Invalid mark');
end;
```

This CASE statement reads the value of iPercentage and then compares it with the ranges from the CASE statement. If it fi nds a match, it uses the *ShowMessage* dialog box to create a pop-up box with the symbol. If no match is found, the ELSE condition activates, which creates a message with the words, Invalid mark.

6. Save and test the application.

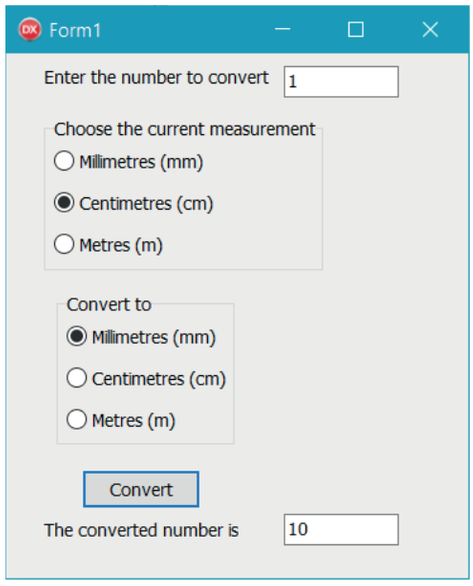


Activity 5.15

Write a program to convert different length units according to the table shown below:

MILLIMETRES (MM)	CENTIMETRES (CM)	METRES (M)
1	0.1	0.001
10	1	0.01
1000	100	1

Your interface may resemble the example below:



Activity 5.16

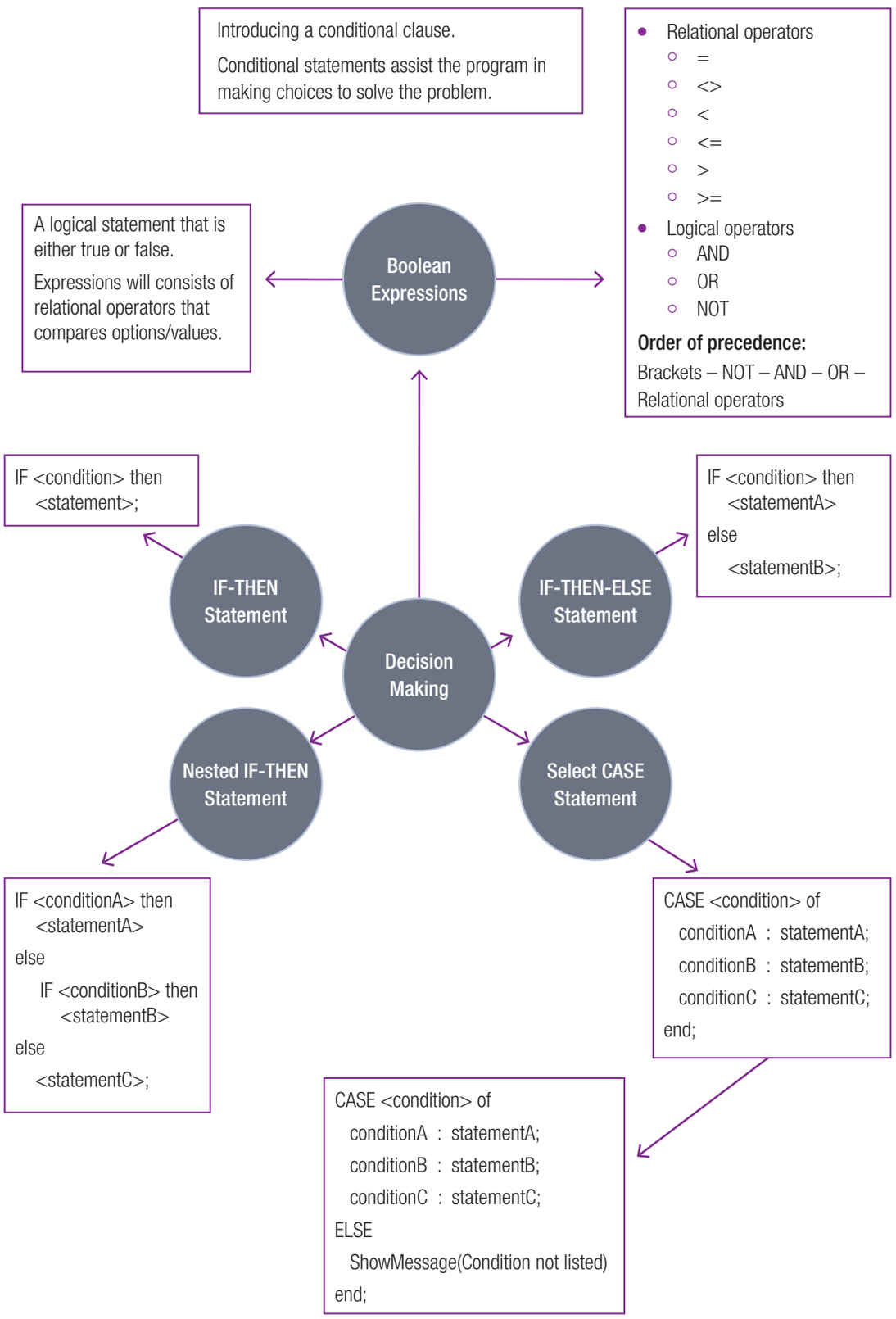
Laurens runs a gardening service, and his rates are displayed in table below:

HOURS PER WEEK	RATE PER HOUR
1 – 2	R 250
3 – 4	R 200
5 – 6	R 175
7 – 8	R 150

5.16.1 Write a nested IF-statement to determine the rate per hour given the hours worked per week.

5.16.2 Determine the rate per hour using a CASE-statement.

DECISION MAKING



1. The following variables are declared in the procedure.

```
Var
    iQuarters : Integer;
    sName : string;
```

Check if the following-statements are valid. If not, correct the statement .

a. (iQuarters > 0)

b. (1 <= iQuarters <= 10)

c. iQuarters = 10 AND sName = 'Rametswe';

d. (iQuaters = '10') OR (sName = '1');

2. In each of the following activities you must decide what structure is the best to use in every situation. Remember to do thorough planning for every activity.

a. Write a program to determine grades in a course with three tests. No test may count more than 10 marks. If a mark entered is more than 10, a message must be displayed to inform the user that the mark is more than 10, and the user must be given an opportunity to enter the correct mark. The grades are determined on the average (rounded) of the three tests. Grades are determined according to the following rule:

- Grade A: an average of 9 or better
- Grade B: an average between 8 and 9
- Grade C: an average between 7 and 8
- Grade D: an average between 6 and 7
- Grade E: an average between 5 and 6
- Grade F: an average below 5

Display the three tests, the average for each test and the grade obtained in a ShowMessage dialog box.

b. You want to sell newspapers to raise extra money. Each paper is sold for R5. The newpaper agency offered you a choice of wage package. Use the information below to create an app to help you decide which wage package you would like to accept:

- Straight wage of R300 per week
- R3.50 per hour for 40 hours plus a 10% commission
- A straight 15% commission on the papers sold per week with no other wage

The program takes your expected weekly sales as input and outputs the weekly wage under each plan. Save and run your application.

c. Write a program to score the rock-paper-scissors game. Each of the two players type in 'P','R', or 'S', and the program announces the winner, as well as the reason that choice won. The following rules apply:

- paper wins over rock because 'paper covers rock'
- rock wins over scissors because 'rock breaks scissors'
- scissors win over paper because 'scissors cut paper'

Each win scores a point for its player. If both players choose the same play, no score is added to either player.

3. There are 13 cards in one suite of cards (such as hearts). They consist of 10 numbered cards, as well as the jack, queen and king.

Create an app to simulate Blackjack by following the guidelines below:

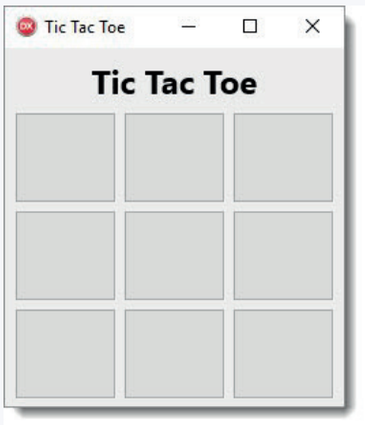
- The application randomly selects two cards from a suite of cards.
- Load the images of the card corresponding with the random cards chosen.
- Calculate the value of the two cards that were chosen.

Remember each card's value is its face value (the three of hearts is worth three); the king, queen and jack are worth 10 each, and the ace is worth 11.

- Add the values of the two cards.
- If the value is 21, display the message, 'Blackjack!', otherwise display 'No luck!'.

5. Optional activity

Follow the instructions below to create a Tic Tac Toe game.



a. Create an interface as shown alongside:

Hint: Each block represents a button.

b. Create a global Boolean variable called *blsCross*.

c. In the variable declaration, set the value of *blsCross* to True, as shown in the code below.

```
Global variable starting value
bIsCross : Boolean = True;
```

It is possible to assign starting values to global variables as the code above shows. This is very useful if you need one of your variables to have a value from the start of the application.

d. Create an OnClick event for the first button by double clicking the button.

e. In the OnClick event, create an IF-THEN-ELSE statement that checks if *blsCross* is true.

f. If *blsCross* is True, set the caption of the first button to 'X' and set the value of *blsCross* to False.

g. Otherwise (using the ELSE statement), set the caption of the first button to 'O' and the value of *blsCross* to True.

The code for the first button is shown below to help you.

```
First OnClick event
if bIsCross = True then
begin
  btnOne.Caption := 'X';
  bIsCross := False;
end
else
begin
  btnOne.Caption := 'O';
  bIsCross := True;
end;
```

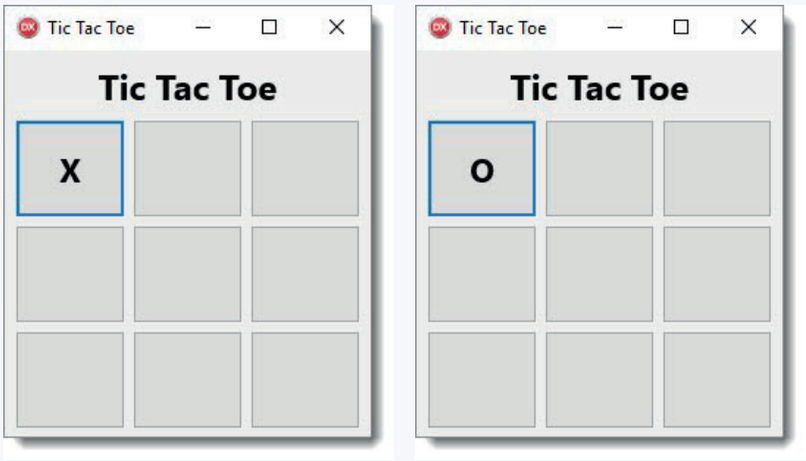
Looking at the code, you will see that the first condition checks the value of *blsCross*. The Boolean variable *blsCross* is used to indicate if the next move should be a cross or a naught.

If the *blsCross* is True, it sets the text of *btnOne* to "X" (a cross) and changes the value of *blsCross* to False. If it is not True, then the ELSE statement activates which sets the caption of the button to "O" and sets the value of *blsCross* back to True.

Did you know

In the conditional statement, you could simply check the value of *blsCross*. Depending on the value of *blsCross*, this will either be True or False. As such, the conditional result of "blsCross = True" will always be the same as Boolean value *blsCross*.

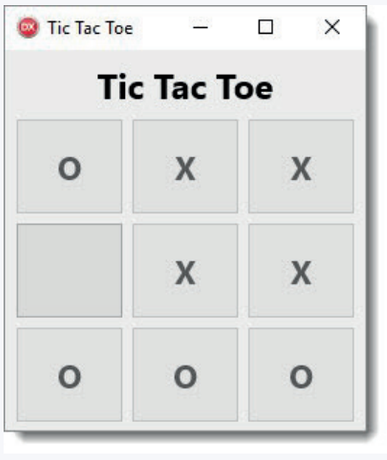
h. Save and test your application. If you created the code correctly, your first button should swap between "X" and "O" each time you press it.



i. In the *OnClick* event handler, set the caption and disable the button to prevent players from pressing it again.

j. Copy the code to create OnClick events for all the other buttons. Make sure you change the code for each event so that it changes the caption of the correct button!

k. Save and test your application. You should now be able to play a game of Tic Tac Toe!



CHAPTER 6

VALIDATING DATA

CHAPTER UNITS	
Unit 6.1	String comparison
Unit 6.2	Validating data
Unit 6.3	IN operator

Learning outcomes

At the end of this chapter you should be able to:

- calculate the length of a string
- compare two strings based on their single character ASCII values
- check (validate) the information that users enter into a program before processing it
- use the IN operator

INTRODUCTION

In spy movies, there are often situations where one spy needs to meet another spy and they need to make sure the person they are meeting is the correct person. To solve this problem, the spies usually have a secret phrase that they say at the start of the conversation, such as, "It always rains in February". If the first spy gives the correct greeting, and the second spy gives the correct response, then the two spies have confirmed their identities and can have a conversation. In computer terms, what these spies are doing is comparing responses. If the response spoken by one spy does not match the string stored in the second spy's memory, then the first spy must be an imposter!

Figure 6.1: *Spies in movies often use string comparisons*

Without thinking about it, you make string comparisons every day. There are several different reasons for using string comparisons in day to day life, including:

- finding the correct item from a list of items
- verifying that an item is correct
- making decisions based on the value of a string.

Similar comparisons can be made in Delphi using conditional statements. These comparisons allow you to improve your application by building decisions into your application. Examples of string comparisons in applications include:

- finding the right person to send a text message to
- confirming that the user knows the correct username and password.

You can use a string comparison in any situation where you need to find an item, confirm that text is correct, or create a condition based on changing values.

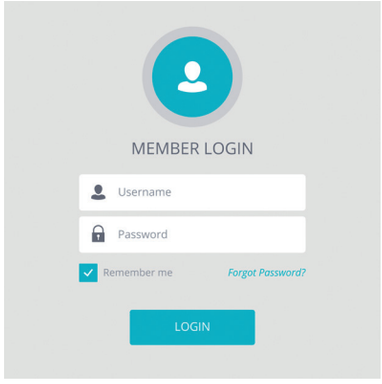


Figure 6.2: *String comparisons can be used for usernames and passwords*

In this chapter, you will learn about string comparisons. You will, more specifically, learn how to determine if one string is larger or smaller than another string, how to validate data and how to find a value in a list of values.

UNIT

6.1 String comparison

In Chapter 5 you learnt about relational operators that can be used to compare integer, real, character and string data. In this chapter you will learn how string comparisons take place. Each character has an American Standard Code for Information Interchange (ASCII) value. The ASCII code for a character is a numerical value. For example, 'A' has the ASCII value 65, 'B' is 66, …, 'Z' is 90.

The table below shows the ASCII code for capital letter 'A' to 'Z', lower case letter 'a' to 'z' and digits 0 to 9.

Table 6.1: *The ASCII code for some characters*

CHARACTERS	ASCII CODE
'A'…'Z'	65…90
'a'…'z'	97…122
0…9	48…57

You will find a full ASCII table in Annexure B at the end of this book.

ALGORITHM FOR COMPARING STRINGS

Delphi compares two strings character by character, starting with the first character of each string using ASCII values as follows:

1. Read the first character of string1 and store it in char1
2. Read the first character of string2 and store it in char2
3. Compare char1 and char2
   - If char1 > char2, then string1 > string2
   - If char1 < char2, then string1 < string2
   - If char1 = char2, then store the next character from each string in char1 and char2
4. Repeat Step 3 until you find out that one string is larger than another or until reach the last character of both strings.
5. If you reach the last character without finding that one string is larger than another, then the two strings are equal.

Example 6.1

If you must compare two strings 'Cat' and 'Cot':

Determine if 'Cat' > 'Cot'

'CAT'		'COT'	
CHAR1	ASCII VALUE	CHAR2	ASCII VALUE
C	67	C	67
a	97	o	111
t	116	t	116

Using the algorithm above:

- Characters in position one in both strings are equal
- Characters in position two in both strings are not equal: Char1 < Char2, therefore string1 < string2.
  Hence 'Cat' < 'Cot' and the condition 'Cat' > 'Cot' is false

Guided activity 6.1 String comparisons

Considering the explanation above, arrange the following strings from the smallest to the largest.

6.1.1 'Benjamin'

6.1.2 'Ben'

6.1.3 'Patience'

6.1.4 'Peter'

6.1.5 'm'

6.1.6 '999'

6.1.7 '1000'

VALIDATING LOGIN INFORMATION

In this section, you will use string comparisons to:

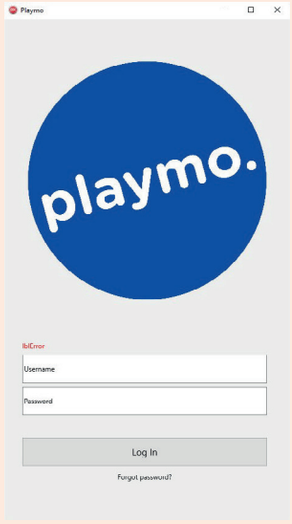
- create a smartphone application's login screen
- create a relational string comparison application.

To do this, you will create a user interface for an application called Playmo.

Example 6.2 Smartphone login page

Use the logo provided to create your smartphone login screen. To do this.

1. Open the **SmartphoneLogin_p** project located in the 06 – Smartphone Login folder. You should see the following user interface.



2. Remove the caption from *lblError*, so that the label is empty. This label will only show a message if the user makes a mistake.
3. Create an *OnClick* event for the [Log in] button.
4. Create two local string variables called *sUsername* and *sPassword*.
5. Assign any username or password to these strings in your event. Do not use your real username and password!
6. Create two new string variables called *sInputUsername* and *sInputPassword*.
7. Set the value of these two variables equal to the text from the Username and Password textboxes.
8. Create a conditional statement that checks if the username and the password is correct.
9. If both are correct, show the user a message stating that they have been logged in successfully.

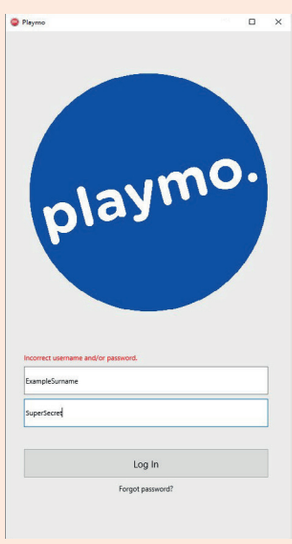
Example 6.2 Smartphone login page *continued*

10. Otherwise, update *lblError* so that it says, 'Incorrect username and/or password.' The code for this conditional statement is shown below.

Condition text comparison

```
if (sInputUsername = sUsername) and (sInputPassword = sPassword) then
  ShowMessage('You have been logged in successfully!')
else
  lblError.Caption := 'Incorrect username and/or password.';
```

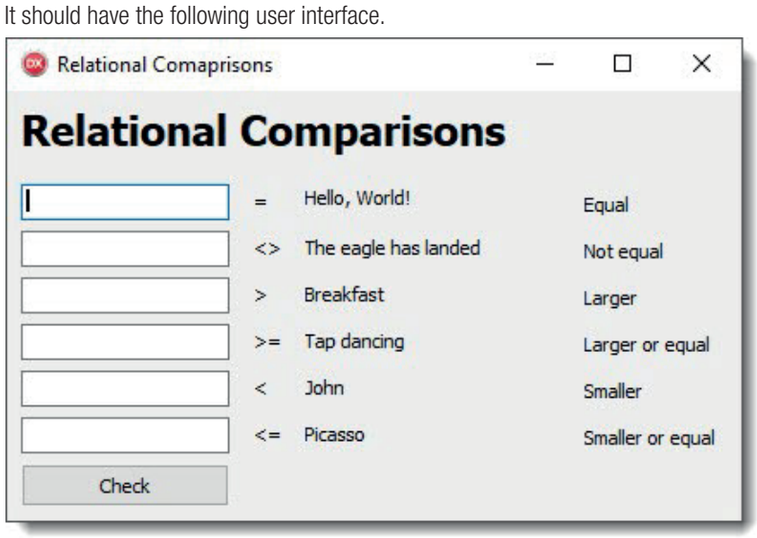
11. Save and test your application. Enter different combinations of correct and incorrect usernames to make sure it works.



Congratulations, you just created a username and password application.

Guided activity 6.2 Relational comparisons

For this guided activity, you need to create an application that will use all six relational operators to compare existing strings with strings you enter. You can start by opening the application stored in the 06 – Relational Comparisons folder. It should have the following user interface.



Guided activity 6.2 Relational comparisons *continued*

The application should already have an *OnClick* event for the [Check] button with the following code.

OnClick event

```
if edtEqual.Text = lblEqualInput.Caption then
  lblEqual.Caption := 'True'
else
  lblEqual.Caption := 'False';
```

This conditional statement checks if the text entered into the first text box is equal to the text in the lblEqualInput caption (that is, 'Hello, World!'). If it is, it updates the lblEqual caption to say TRUE. If not, the lblEqual caption says FALSE.

Using a similar structure to this code:

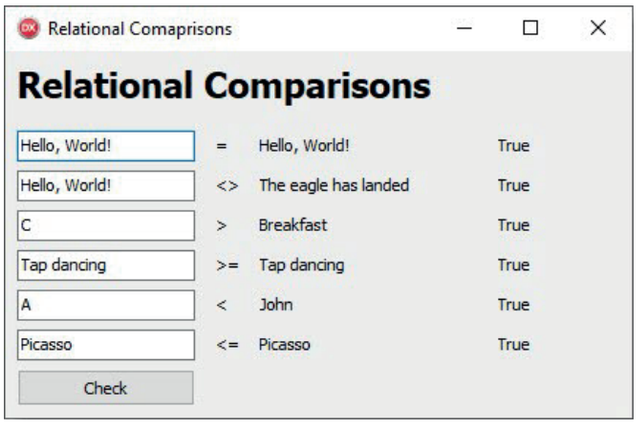
6.2.1 Create similar conditional statements for the remaining relational operators.

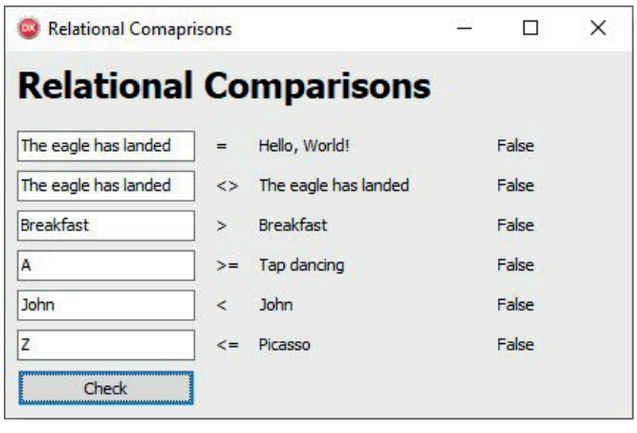
6.2.2 Write down a value for each line that will result in the text TRUE being shown.

6.2.3 Write down a value for each line that will result in the text FALSE being shown.

6.2.4 Write down the text that is shown for each line if you leave the textboxes empty.

For question 2 and 3, you can use the application to test your own answers. The images below show two possible answers.





Activity 6.1

Make the following changes to the Playmo login screen:

6.1.1 Create an *OnClick* event for the *Forgot password* label that displays a password clue.

6.1.2 If the label is pressed a third time, display the whole password.

UNIT

6.2 Validating data

Data validation is a technique used by programmers to check (or **validate**) the information that users enter before processing it. This allows programmers to prevent common errors from occurring by making sure that the information entered is correct before it is used. The goal of **input validation** is therefore to prevent users from accidentally or purposefully entering incorrect data into your program.

If your program automatically generates the data it will use, you can test the data before using it. You can also improve the algorithm generating the data to ensure that only the correct types of data are generated for your program.

In contrast, when a user is asked to supply data for your program, many unpredictable things can happen. The user may have:

- misunderstood what data is expected
- clicked the next button without entering data
- entered the correct data in an incorrect format
- entered the correct type of data, but an incorrect value.

This incorrect data can cause your application to crash, or even worse, provide incorrect output.

DIFFERENT TYPE OF INPUT VALIDATION

You can use different types of input validation in your program, including:

- **Required input validation** prevents the processing until certain required inputs are given. When you must read a value and perform a calculation from an *Edit* component, you may want to test whether the *Edit* component has a value before proceeding with calculations. For example:

```
    ….
if edtAmount.text = '' then
       ShowMessage('Enter a value')
    Else
      rAmount := StrToFloat(edtAmount.text);
    ….
```

Take note

'' refers to a null/empty string

- **Type validation** ensures that the data entered is the correct data type. In, for example, a calculator application, you could prevent a user from entering any values that are not numbers. Alternatively, you could inform the user that an invalid input was entered if he or she tries to do a calculation with letters and request them to enter the correct data type. This will be covered in Grade 11 and 12.
- **Length validation** ensures that the data entered is the correct length. In the Smartphone Login application created above, you could use length validation to ensure that all passwords are at least eight characters long. For example:

```
    ….
    sPassword := edtPassword.text;
if length (sPassword) >= 8 then
       ….
    Else
     ShowMessage('Password requires 8 or more characters');
    ….
```

Take note

The *Length* function determines the length of a string.

- **Range validation** is used to ensure that number or date falls within a specific range. For example, in a form asking for age, you might use range validation to ensure the age is between 0 and 120. For example:

```
iAge:=StrToInt(edtAge.text);
if (iAge>0) AND (iAge<120) then
  ….
Else
  ShowMessage('Enter an age in the range 1 to 119');
….
```

- **Pattern matching validation** ensures that the data entered matches a specific pattern. For example, all email addresses would match the pattern that they contain a bunch of letters or numbers, followed by an '@' sign, followed by more letters or numbers, followed by two or more groups of characters separated by full stops. You will learn about pattern matching validation in Grades10 and 11.

Did you know

In general, it is better to prevent a user from making a mistake than to inform the user that they have made a mistake afterwards.

IMPLEMENTATION OF INPUT VALIDATION

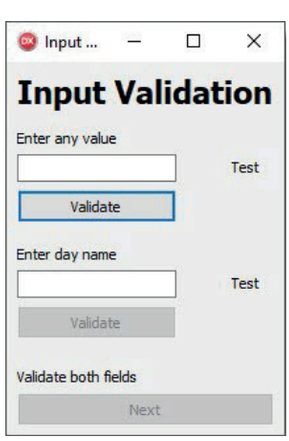
Input validation can be implemented in several different ways:

- One way to implement input validation is to inform the user of the problem before they try to process the data. This could be in the form of an error message or a disabled button with an error message.
- A second way of implementing input validation is to check the data before it is processed. With this implementation, you build certain checks or conditional statements into your program to ensure that you do not process incorrect data. When these statements identify incorrect data, you send a message to inform the user of the problem.

To see how input validation can be used in Delphi, work through the following guided activities.

Guided activity 6.3 Input validation

Open the project saved in the 06 – Input Validation folder.
You should see the following user interface.



In this application, complete the following tasks:

6.3.1 Create an *OnClick* event for the first [Validate] button.

6.3.2 When the button is pressed, use a conditional statement to make sure the edit box's value is not blank (that is, it is not equal to '').

a. If it is not blank, set the value of the first *Test* label to 'Correct' and enable the second [Validate] button.

b. If it is blank, set the value of the *Test* label to 'Incorrect'.

c. Add code to enable the second [Validate] button.

Guided activity 6.3 Input validation *continued*

6.3.3 Create an *OnClick* event for the second [Validate] button.

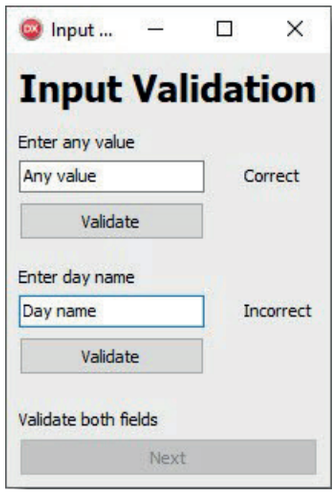
6.3.4 Create a conditional statement to check if the text entered into the second edit box is equal to the name of one of the days of the week.

- **a.** If it is, set the label's caption to 'Correct'.
- **b.** If it is not, set the label's caption to 'Incorrect'.
- **c.** Add code to enable the [Next] button.

6.3.5 Create an OnClick event for the [Next] button.

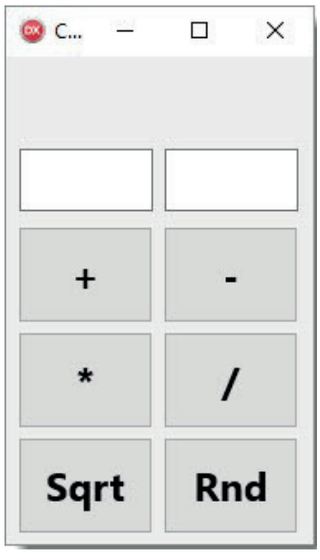
6.3.6 Show a message congratulating the user on completing data validation if the [Next] button is clicked.

6.3.7 Save and test your application.



Guided activity 6.4 Intermediate calculator validation

Open the project stored in your 06 – Intermediate Calculator folder.



Using this application, add the following data validation:

6.4.1 For the [Sqrt] button, add a conditional statement to make sure the number entered is equal to or larger than 0. This prevents the program from trying to find the square root of a negative number.

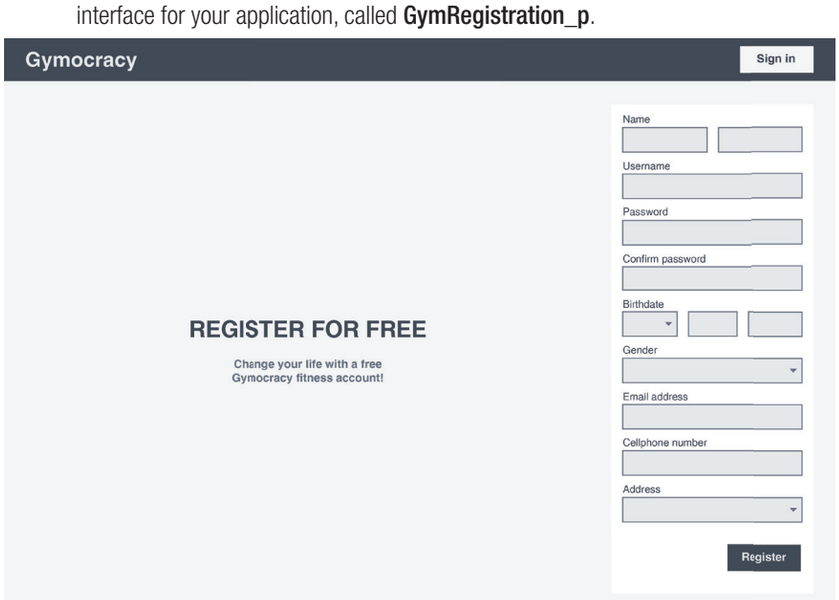
Guided activity 6.4 Intermediate calculator validation *continued*

6.4.2 For the [/] button (that is, the division button), add a conditional statement to make sure the value entered into the second edit box is not 0. This prevents the program from dividing by 0.

In both cases, you can use the *ShowMessage* function to inform the user if they have entered an illegal number into the edit boxes. Once done, save the project in the 06 – Intermediate Calculator folder.

Activity 6.2 Individual activity

6.2.1 Your friend is a young graphic designer who has designed the following user interface for your application, called **GymRegistration_p**.



a. Use this design as a basis to create a Delphi sign-up form that asks the user's name, surname, gender, birthdate, ID, email, cellphone number, residential address, weight, and length.

b. Create an IPO chart for this application. In the *Input* section, describe five useful input validation techniques that can be added to the form. In the *Processing* section, describe how you would implement each of these input validation techniques. Finally, in the *Output* section, list the error messages that you would give to the user if the input validation is not successful.

c. Implement the input validation techniques in your application to prevent invalid inputs.

6.2.2 Choose a program that you have created this year.

a. Describe three input validation techniques that could be added to the application.

b. Implement one of the input validation techniques.

UNIT

6.3 IN operator

In Chapter 5 you learned how to use the CASE statement to check if a variable falls within a specific range. Specifically, you checked if a percentage value occurred in a specific range and then used the result to display a message to the user.

Look at the following code:

CASE range comparisons

```
case iPercentage of
  0..69 : ShowMessage('F');       // 0..69 represents all numbers from 0 to 69
  70..79 : ShowMessage('B');
  80..100 : ShowMessage('A');
else
  ShowMessage('Invalid mark');
end;
```

This CASE statement checks if a specific value occurs within a set of values. Note that a range of number is shown in the format: n..m where n is the start number and m is the end of the range. Two dots(..) represents all numbers between n to m.

Did you know

You can only use integers or characters in a set, not string or real values.

IN OPERATOR

You can test if an element is included in a set of values using the IN operator. The test returns TRUE if the element is found in the set of values; otherwise the test returns FALSE.

SYNTAX OF THE IN OPERATOR:

- The syntax used for the IN operator is: Element **IN** [set of Values]

Take note of the following:

- *Element* is the variable that is being tested against the set of values
- *Element* variable/value can only be an ordinal data type (integer, char & Boolean). The values in the [*set of values*] must match the data type of element
- The IN operator checks whether the *Element* is found in the set [*set of Values*]. The set of values appear within square brackets []

For example:

```
Var cLetter:char;
          ….
bFound := cLetter IN ['a','b','c'];
if bFound then
  ShowMessage('Letter Valid')
else
  ShowMessage('Letter Invalid');
          …
```

Notes

- If cLetter has the value ‘a’ then bFound will be TRUE and ‘Letter Valid’ will display. Remember that ‘a’ is not the same as ‘A’
- If cLetter has any other value than ‘a’, ‘b’ or ‘c’, then *bFound* will be FALSE and ‘Letter Invalid’ will display.

When using the IN operator, the set of values can be:

- a range of values (with the minimum and maximum values separated by two full stops)
- individual values (with the values separated by commas)
- a combination of a range of values and individual values.

Here are some examples of statements using an IN operator:

- if iMonth in [1,2,3,4,5] then .. Checks whether iMonth is one of the first 5 months. The set of number can also be written as a range [1..5] because the numbers are inclusive
- if iNum in [1..5,8, 50..53] … Checks whether iNum is in the set 1 to 5, 8 and 50 to 53
- if cLetter IN ['A'..'Z','a'..'z'] then … 'A'..'Z' and 'a'..'z' indicate a range of uppercase and lowercase letters

When creating a conditional statement, the IN operator is used in place of the equals operator because you are using the conditional statement to see if your value can be found inside the set.
To see how a set can be used in an application, work through the following example.

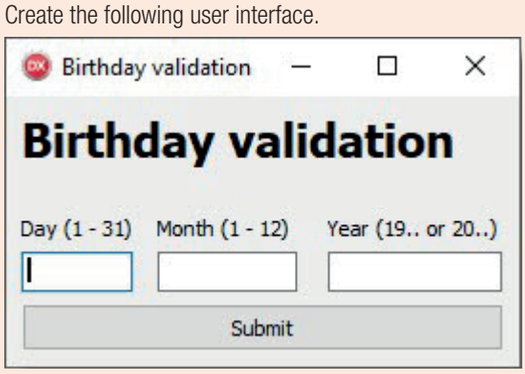
Example 6.3 Birthday validation

You created an application that asks users to enter the day, month and year for their birthday. However, you noticed that a small percentage of users either make typing errors or enter the correct value in the wrong place.

To fix this problem, you want to add input validation to your birthday application that will ensure the day entered is between 1 and 31, the month is between 1 and 12, and the year is in the range 1923..2019.

To do this:

1. Create a new application and save it in the folder 06 – Birthday validation.
2. Create the following user interface.



3. Create an *OnClick* event for the [Submit] button.
4. Declare three local integers called iDay, iMonth and iYear.
5. The variable declaration should now look as follows:

Variable declaration

```
var
  iDay, iMonth, iYear : Integer;
```

Example 6.3 Birthday validation *continued*

6. Read the values from the *edtDay*, *edtMonth* and *edtYear* component and store the values in iDay, iMonth and iYear respectively.

Setting the integers' values

```
iDay := StrToInt(edtDay.Text);
iMonth := StrToInt(edtMonth.Text);
iYear := StrToInt(edtYear.Text);
```

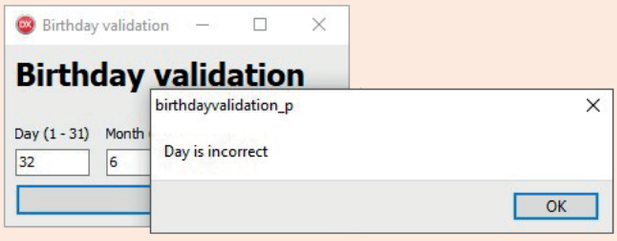
7. Use conditional statements to check whether:
   - iDay is in the range 1 to 31
   - iMonth is in the range 1 to 12
   - iYear is in the range 1923..2019

   Use conditional statements to show a message if the integers are not found in the set of valid values. Remember to surround the condition with brackets and place the NOT operator before the condition. The conditional statements are shown below.

Conditional statements

```
if not (iDay IN [1..31]) then
  ShowMessage('Day is incorrect');

if not (iMonth IN [1..12]) then
  ShowMessage('Month is incorrect');

if not (iYear IN [1923..2019]) then
  ShowMessage('Year is incorrect');
```

8. Save and test your application. Try using both valid and invalid inputs to see how the program reacts.



Congratulations! You have created an application that used sets and the IN operator to do data validation.

Activity 6.3

Using a pen and paper, write down statements that do the following.

6.3.1 Give a set that could contain any letter.

6.3.2 Give a set that could contain all the special characters that could be used as part of a password.

6.3.3 Give a set that could contain any number from 0 to 50.

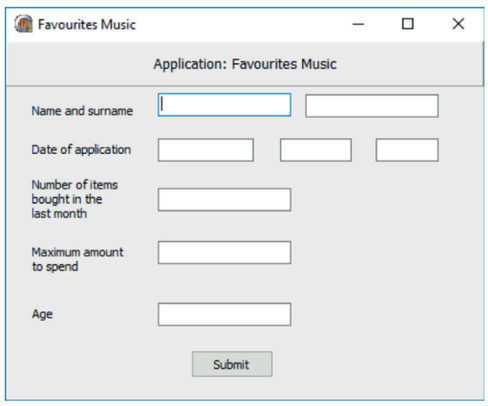
6.3.4 Give a set that contains the capital letters A, B, C, D, E and F.

6.3.5 Give a set that contains the numbers 7, 12 and 25.

6.3.6 Check if the letter "c" is found in a set.

Activity 6.4

Write an application that will allow a user to apply for membership at a music store. Use data validation techniques to prevent accidental incorrect input.



Input must follow the following rules to be valid:

- The name and surname:
  - The field is compulsory – it cannot be left empty.
- The date of application must correspond with the actual date:
  - The date must be entered in the format yyyy/mm/dd.
- The number of items bought in the month must:
  - Be an integer value.
  - Must be more than 10 and less than 25.
- Maximum amount to spend must:
  - Be a real value.
  - Not exceed R500.
  - May not be smaller than 0.
- Age:
  - The applicant must be older than 18.

6.4.1 Complete the table below to record the possible errors and how it will be addressed. The first instance has been done for you:

1. Name and surname	Possible error	Data validation technique
May not be left empty	Submit without having filled in a name and surname	Test for length of string. Error message: 'Please enter a value'
2. Date of application	Incorrect date	
	Date in incorrect format	
3. Number of items	Not an integer value	
	Not in the range 10 to 25	
4. Maximum amount	Not a real value	
	Value exceeds R500	
	Value less than 0	
5. Age	Not Integer value	
	Applicant younger than 18.	

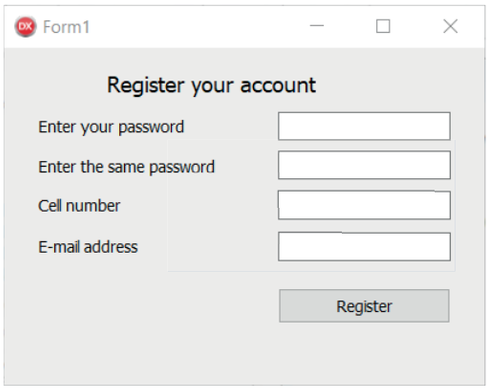
6.4.2 Use the solutions suggested in your table to write code to validate data for all input components.

Activity 6.5

Open the **RegisterAccount_p** project from the 06 – Register Account folder.

A user must enter a password to register an account with their Favourite Music Store. They are then asked to verify the password by retyping it.

Other information required include the user's cell phone number and email address.

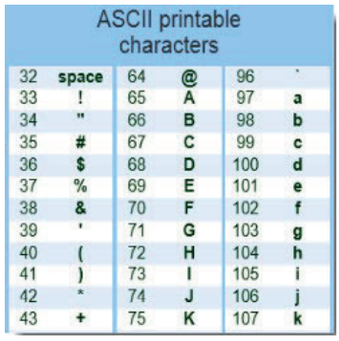


6.5.1 Add code to *btnRegister* to verify the data. Data is valid when:

- all Information is entered
- the two passwords match
- the cell number Is of the correct data type
- the cell number has the correct number of digits

6.5.2 Code an error message for each type of data validation done.

VALIDATING DATA



ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k

String comparison (According to ASCII values)

The length of a string = the number of characters

- Indexing of characters in a string:
- myWord[1] returns 1st character of myWord
- Most known characters have ASCII values
- For comparison the ASCII value is used
- Produce special char with: Alt + ext. ASCII
  E.g.: √ (Alt + 251) ë (Alt + 137)

- Ascending order: **A → Z**
- Descending order: **Z → A**
- For words compare matching characters
- ASCII table is the foundation for char / string comparison

True ← ('H' > 'A')
True ← ('a' > 'H')
True ← (65 < 75)
True ← ('2' < 'A')
True ← ('A' > '❤')

Validate data

Types of input validation

- Required input
- Data type validation
- Length validation
- Range validation
- Pattern matching validation

If a ***user*** is asked to supply data for input, many unpredictable things may happen. The user may have:

- Misunderstood what data is expected
- Ignore your prompt and hit ENTER (no data was entered)
- Typed in a wrong piece of data (letters instead of a number)
- Supplies correct data, but outside the acceptable range.

We as application builders need to control all input data and test it if necessary.

The IN-operator

The IN operator is used to see whether one specific item forms part of a collection of unique ordinal items, not necessarily in a specific order.

All items must be of the same data type to belong to one set.

E.g.
Test for "long" months (31 days) in a year:

```
IF (iMonthNum IN [1,3,5,7,8,10,12]) then
   ShowMessage('Selected month has 31 days.');
```

Test for gender in an ID-number:

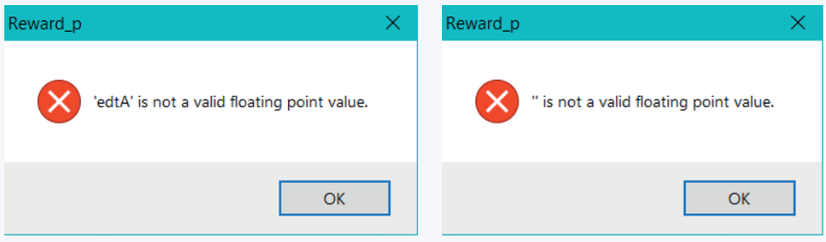
```
IF (iRSAID[7] IN [1..4]) then
   ShowMessage('ID comes from a female.');
```

Test for a capital letter:

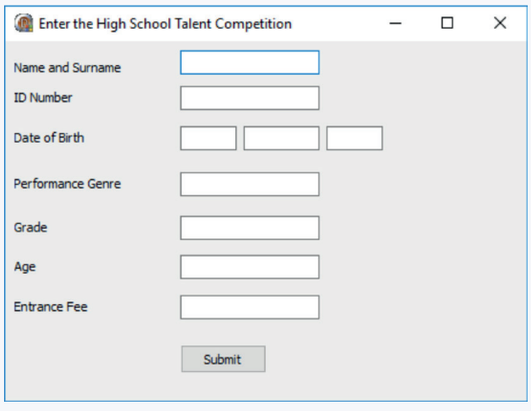
```
IF (sFirstName[1] IN ['A'..'Z']) then
   ShowMessage('The first name starts with a
   capital letter.');
```

Consolidation questions — Chapter 6: Validating data

1. One way in which to make sure that correct data is entered by the user is through using String comparison.
   a. What type of data is stored in a string variable?
   b. If you want to verify a user's password, which conditional operator would you use in your string comparison?
2. Explain how it is possible for Delphi to interpret a condition, such as, If 'Cat' > 'Dog'.
3. Compare the error messages below:



   a. Both errors were caused by a user error. Explain the user actions that led to these errors.
   b. Suggest validation techniques that you could apply to prevent this error in future.
4. Give two reasons why it is important to validate input data.
5. A user is asked to enter their cell number.
   a. Name two types of validation that you would apply to ensure that the data is valid.
   b. Write down pseudo code for the validation described above.
6. Describe one instance where you would use range validation.
7. Carefully consider the following statement: Valid data is not always correct.
   Explain why this statement is TRUE, by giving an example.
8. Study the interface and different input components of the interface below carefully. Use the interface to create a new program that will allow learners to enter a talent competition at your school. Add the necessary data validation to ensure that all inputs are valid.



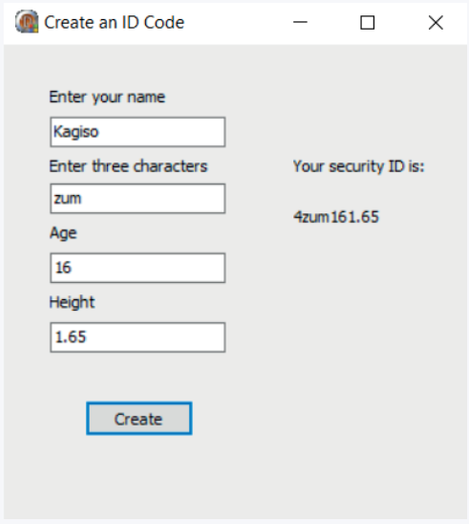
   Take note of the following:

   - The genre must include the word 'Dance' OR 'Music'. Genres could be Contemporary Dance, Jazz Music, etc.
   - The name and surname must appear in capital letters.

9. You need to generate secure IDs for your teacher to use as identification for network profiles for your Grade 10 class.

To generate the secure ID the string is built up by:

- Adding a number that is determined by the first letter of the user's name according to the following pattern:
  - A to C are represented by 1
  - D to F are represented by 2
  - G to H are represented by 3
  - K to L are represented by 4
  - M to O are represented by 5
  - P to S are represented by 6
  - S to V are represented by 7
  - W to Z are represented by 8
- Adding three characters entered by the user:
- Adding the age of the user
- Adding the height of the user, displaying at least one decimal value



Open the **Secure_p** project from the 06 – Secure ID folder and complete the code as follows.

a. Check that all data entered is valid.

Take note of the following:

- The first letter of the name is a capital letter.
- The user must enter any 3 characters.
- The age must be entered as an integer value and be appropriate for Grade 10. Allow for learners two years older and two years younger than the usual age for Grade 10.
- The height must be entered as a real number.

b. Complete the code for *btnCreate* by creating a string following the rules explained above.

c. Display the *SecureID* in the label, *lblSecureID*.

d. Save and close your program.

TERM 3

CHAPTER 7

REPETITION

CHAPTER UNITS	
Unit 7.1	Using the ListBox and ComboBox components
Unit 7.2	Looping algorithms and flowchart
Unit 7.3	FOR loop
Unit 7.4	Using loops with components
Unit 7.5	Using the Input Box
Unit 7.6	REPEAT loop
Unit 7.7	WHILE loop
Unit 7.8	Applying loop structures
Unit 7.9	Initialising variables using the OnShow event
Unit 7.10	Timers

Learning outcomes

At the end of this chapter you should be able to:

- create and use a ListBox
- explain and apply FOR loops in your programming
- explain and apply WHILE loops in your programming
- explain and apply REPEAT loops in your programming
- initialise variables using the OnShow event
- apply loops with different components
- use the timer component to create code that runs when a timer event triggers.

INTRODUCTION

There are many situations in real life where you need to repeat the same action until a specific goal is reached. When you practice for a sport or practice an instrument, you repeat the same steps until you master the skill. In fact, most people repeat the same routines: every morning they wake up and every night they go to sleep, with only the decisions made during the day differing slightly! Think about your morning routine. Do you need to create a new morning routine every morning or can you simply repeat the same basic tasks each morning without thinking?

Figure 7.1: *Most people follow the same morning routine every day*

The advantage of repeating the same tasks is that you only need to decide how to do the task once.

So far you learned to read in data in a variable. If you need to read ten values and keep a running total, then you could have done this in the following way:

- initialise the running total
- read a value
- update running total with this value
- read a value
- update running total with this value
- repeat until all values are read and added to the running total.

For this, you will need to write about 21 lines of code! This will make your program long and cumbersome. Imagine how many lines of code would be required to add 100 numbers!

In Delphi, you can use looping constructs that allows you to write a set of repetitive tasks once to achieve the same purpose. This chapter will teach you how to create programming **loops**.

New words

loops – loops repeat certain lines of code until a specific condition is met

UNIT

7.1 Using the listbox and combobox components

The *ListBox* component is used to display and manage a scrollable list of items (that is, a list of items that you can scroll up and down in). If you can see a fully populated (top to bottom) *ListBox*, then a vertical scroll bar automatically appears. The same happens with the horizontal scroll bar.

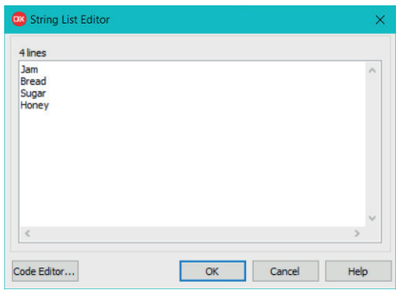
To place a *ListBox* component on the form:

- select the *TlistBox* component from the *Standard* palette and place it on the form
- the prefix, *lst*, is used in naming the component

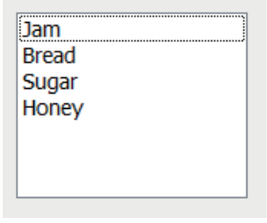
Each line inside the *ListBox* is called an item. In simple terms, we can think of one item as being a *string* and *many* items as data type *TStrings*. *TStrings* represents a list of strings.

POPULATING A LISTBOX

- To populate a *ListBox* with items during design time from the *Object Inspector*:
  - select the *ListBox*
  - in the *Items* property click the ellipse (…)
  - a *String List Editor* dialog box will appear
  - type the list of items on separate lines in the *String List Editor*
  - click OK.



- the items will now be displayed in the *ListBox* component.



- To add an item to a *ListBox* during run time simply execute this code *per* item:

```
lstData.Items.Add('Butter');
```

This value will be added to the *Items* property programmatically.

RETRIEVING INFORMATION FROM A *LISTBOX*

All captured lines, which are added to the ListBox, are put in sequence and are given an index number called, *ItemIndex*. If you need to read a line from a *ListBox*, you can access it using the ItemIndex. So, if you click on a line inside the *ListBox*, it will be highlighted and the *ItemIndex* value can be read.

You will also get the sequence number for the line (Item) starting from 0 (we say it is zero-based). For example:

```
iIndex := lstData.ItemIndex;      // If first line was
                                  // highlighted it
                                  // returned 0
```

To access the content/text of the line you selected, use the following code to retrieve the value of that specific item:

```
sData := lstData.Items[iIndex];   // Retrieve the selected
                                  // item's contents
sName := lstNames.items[0] ;      // Reads the first value
                                  // in the list
sName := lstNames.Items[3];       // Reads the fourth
                                  // value in the list
```

New words

readable – you can access the value

writable – you can add information to a value

The property *ItemIndex* is **readable** and **writable**. This means you can assign a value to it. If you execute the following line, you will see that line 2 will be highlighted:

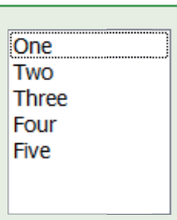
```
lstData.ItemIndex := 1;          // Line 2 will be selected
                                 // during run time
```

To test whether an item exists inside a *ListBox* you can use the method *IndexOf*(). *IndexOf*() takes one argument, the string to search for and it returns the index of the item where the string was found.

Example 7.1

Compare the given code with the *ListBox* items shown:

```
iIndex := lstData.Items.IndexOf('Three');
```



The integer *iIndex* will be assigned the value 2.

If we executed the following line:

```
iIndex := lstData.Items.IndexOf('Six');
```

The integer *iIndex* will be assigned the value –1, because no *Item* could be found to match 'six'.

You can also test if an item is in the list by determining the index position of the item. If the index position is –1, then the item is not in the list.

Example 7.2

```
If lstData.Items.IndexOf('Six')<>-1 then
    ...
Else
    ShowMessage('Item not in the list');
```

Let's have a look at some more useful properties of the *ListBox*.

COUNTING THE NUMBER OF ITEMS IN THE LISTBOX

You can find out how many items there are in a *ListBox* by using the following code:

```
iCount := lstListBox1.Items.Count   // will return the
                                    // number of items in
                                    // the ListBox
```

IMPORTANT LISTBOX PROPERTIES

- A *ListBox* has a property *Sorted*, which is set to FALSE by default. If it is set to *TRUE*, all lines will be sorted in ascending order (that is, A to Z). Once sorted, the process cannot be reversed. An *Unsort* property does not exist.
- A tab-stop size can be assigned to the property *TabWidth*. The number assigned sets equal-sized tab-stops across the *ListBox* in **dialog units**. Each character is approximately four dialog units wide.

Example 7.3

View the three screenshots showing different *ListBoxes*.

All *ListBoxes* are assigned the font 'Courier New' with a `TabWidth := 20`.

```
Courier New (size 8)
123456789012345678901
     |    |    |    |
```

See how the tab distance grows bigger as the font size increases. However, the first tab-stop is always positioned just after the '5', that is, 20 (TabWidth) / 5 (characters) = 4 (dialog units per character).

```
Courier New (size 10)
123456789012345678901
     |    |    |    |
```

```
Courier New (size 12)
123456789012345678901
     |    |    |    |
```

To use the defined tab-stops (TabWidth) of a *ListBox*, the code must explicitly contain the tab control character (#9), which is defined at position 9 as HT (Horizontal Tab) in the ASCII table.

To produce the four vertical bars (as shown in the example) on the third line, you must execute the following code:

```
lstData.Items.Add(#9 + '|' + #9 + '|' + #9 + '|'+ #9
+ '|');
```

Take note

- To activate a **tabulator effect**, you must set the *TabWidth* and include tabs (#9) in your text.
- To adjust the correct tab width for a tabular neat output with data columns is a "trial and error" exercise.

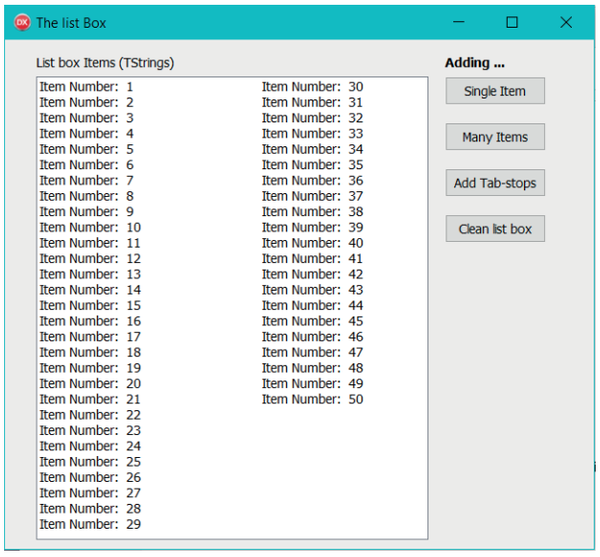
New words

dialog units – the average width and height of characters in the system font

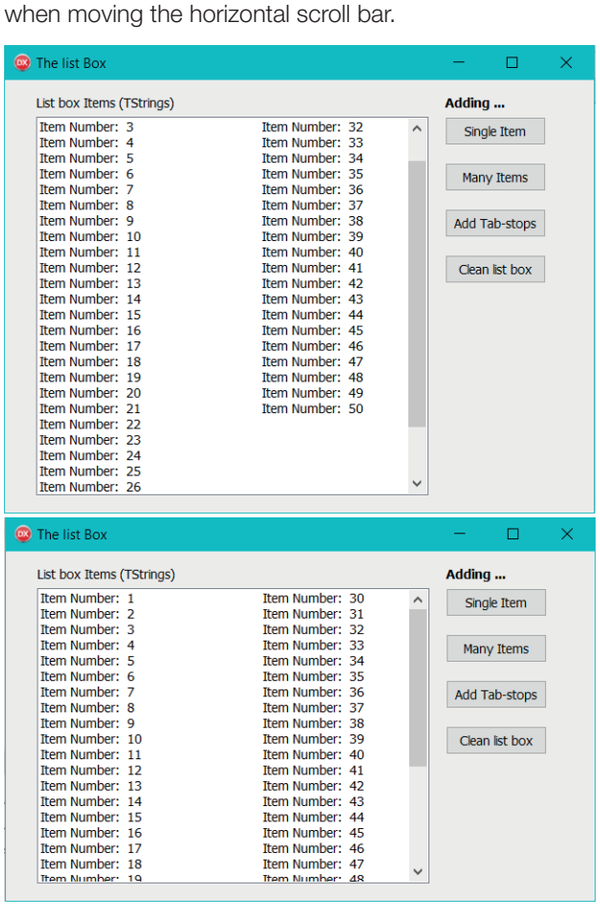
A COMPUTER SCIENCE QUESTION

https://www.youtube.com/watch?v=k4RRi_ntQc8

A *ListBox* can display bulky data in a more compact format by adding *Columns* to the *ListBox*. The property *Columns* works similarly to the columns in MS-Word. The columns are evenly spaced across the width of the *ListBox*.
For example, in the screenshot below, the columns are set to 2 and the *ListBox* is anchored to the right and bottom.



Even when the *ListBox* is resized, it keeps the data in two columns, but rearranges the data. The data that does not fit well is expanded to the right and can be seen when moving the horizontal scroll bar.



The table below provides a list of other useful ListBox properties.

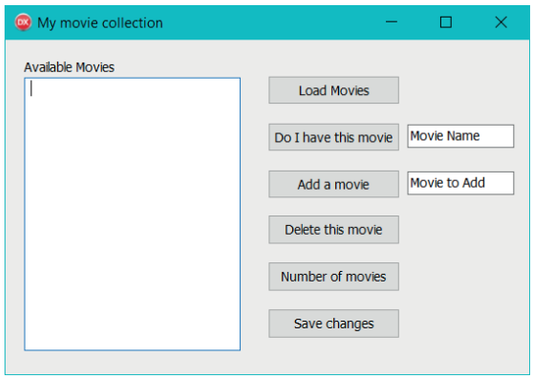
Table 7.1: *Other useful ListBox properties/methods*

PROPERTIES /METHODS	DESCRIPTION
lstListBox1.DeleteSelected;	To delete the selected item from the ListBox
lstListBox1.Items.Delete(lstListBox1.ItemIndex);	To delete an item from a specific ItemIndex position
lstListBox.Items.Insert(lstListBox1.ItemIndex,edtValue.Text)	To insert edtValue.text to position ItemIndex
lstListBox1.Items.Clear;	Clears the ListBox
lstListBox1.Items.LoadFromFile('Data.txt');	Data from the text file Data.txt is used to populate the ListBox
lstListBox1.Items.SaveToFile('Data.txt');	Data from the ListBox is saved to the text file Data.txt
iCount:=lstListBox1.Items.Count	Will return an integer value containing the number of items in the ListBox
showMessage(lstListBox1.Items[4]);	Will display the item at ItemIndex 4, that is the 5th item in the ListBox

Activity 7.1

You have some movies on an external hard drive. You want an application that will help you to manage your movies.

Study the interface shown on the right, then open the **Movies_p** project from the 07 – Movies folder file and complete the application as follows:



7.1.1 Write code for *btnLoadMovies* to load the movies from the file Movies.txt in your folder into the *ListBox*.

7.1.2 **a.** Write the code for *btnFind* to check whether the movie you are looking for is in the *ListBox*. The name of the movie should be entered into *edtLookFor*.

b. Use a *ShowMessage* dialog box to indicate whether or not the movie is found.

HINT: If the item is not in the list the index number will be -1.

7.1.3 Add data validation to *btnAdd* to prevent a user from adding the text 'Movie to Add' to the list.

7.1.4 Write code for *btnAdd* to add a new movie to the list. The name of the movie should be entered in *edtAdd*.

7.1.5 Add code to the *btnDelete* event to delete a movie that a user clicked on in the *ListBox*.

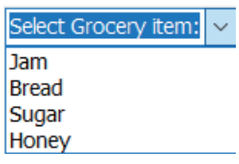
7.1.6 Write the code for *btnNumMovies* to display the number of movies in the *ListBox* in a *ShowMessage* dialog box.

7.1.7 Write code for *btnSaveChanges* to save the content of the *ListBox* as it is displayed in *lstMovies*.

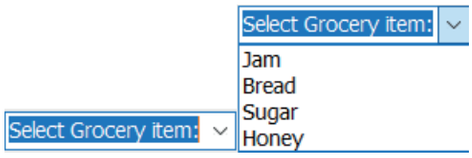
7.1.8 Save and run your app.

THE COMBOBOX

A *ComboBox* is a scrollable drop-down list from which a user is able to select an item. The *ComboBox* is very similar to the *ListBox*, except that the items are not instantly visible nor displayed. You must click on a drop-down arrowhead to display a limited number of items. It is a component optimised to serve as a **selection tool**. Both the *ListBox* and the *ComboBox* allow the user to select only one choice at a time.



- To place a *ComboBox* component on the form:
  - select *TComboBox* component from the *Standard palette* and place on it the form
  - the prefix, cmb, is used in naming the component
- To fill the *ComboBox* with items from the *Object Inspector*:
  - select the *ComboBox*
  - in the Items property click the ellipse (…). A *String List Editor Dialog box* will appear
  - type the item list in the *Editor* on separate lines and click OK.
- To set the default prompt text for the ComboBox:
  - select the *ComboBox*
  - in the *Text property* enter some meaningful text (for example, Select grocery items) about the selections in the *ComboBox*. This text will appear only once before a selection is made. Once a selection has been made from the items in the *ComboBox*, the selected item displays:



- To add an item, for example, butter, to a *ComboBox* during run time, use the code found below:

```
cmbComboBox1.Items.Add('Butter');  //will add butter to the list
```

- To retrieve the index of the selected item execute the code as shown below:

```
iIndex := cmbComboBox1.ItemIndex;
```

Just like the ListBox, if an item is not selected, *ItemIndex* returns -1 and the item position starts from 0.

- To retrieve an item from the *ComboBox* execute the code as shown below:

```
//will return an item from the list at the selected index position
sGroceryItem:=cmbComboBox1.Items[cmbComboBox1.ItemIndex];
```

or

```
sGroceryItem = cmbComboBox1.text; // will return the text at the selected
                                  // index position
```

or

```
sGroceryItem := cmbChoice.Items[0];      // will retrieve the first item in
                                         // the Combo Box
```

- To count the number of items in a *ComboBox* execute the code as shown below:

```
iCount := cmbComboBox1.Items.Count;      // returns the number of items in
                                         // the ComboBox
```

The table below lists other useful *ComboBox* properties.

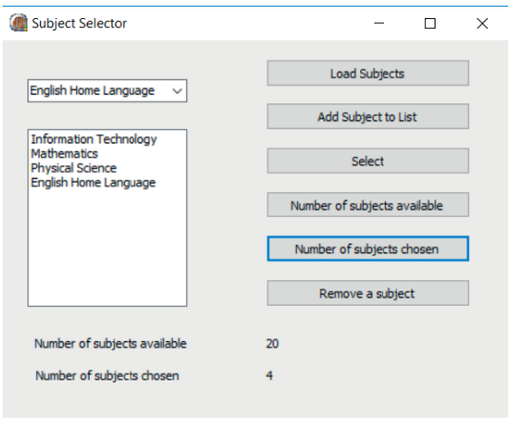
Table 7.2: *Other useful Combo Box properties/methods*

PROPERTIES /METHODS	DESCRIPTION
cmbComboBox1.DeleteSelected;	To delete the selected item from the combo box
cmbComboBox1.Items.Delete(cmbComboBox1.ItemIndex);	To delete an item from a specific ItemIndex position
cmbComboBox1.Items.Insert(cmbComboBox1.ItemIndex,edtValue.Text)	To insert edtValue.text to position ItemIndex
cmbComboBox1.Clear;	Clears the combo box
cmbComboBox1.Items.LoadFromFile('Data.txt');	Data from the text file Data.txt is used to populate the combo box
cmbComboBox1.Items.SaveToFile('Data.txt');	Data from the combo box is saved to the text file Data.txt
iCount:=cmbComboBox1.Items.Count	Will return an integer value containing the number of items in the combo box
showMessage(cmbComboBox1.Items[4]);	Will display the item at ItemIndex 4, that is the 5th item in the combo box

Activity 7.2

The Subject selector application allows a user to choose subjects and display the chosen subjects in a *ListBox*.

Study the interface of the application shown on the right.



7.1.1 Recreate the interface of this application.

Note: the following components are used on this interface:

- *ComboBox*
- *ListBox*
- six buttons
- four labels

7.1.2 Write the code for each button as follows:

a. [Load subject] button loads the list of subjects from the subjectlist.txt file (provided) in the *ComboBox*.

b. [Add subject to list] button adds the subject 'Life Orientation' to the *ComboBox*.

c. The [Select] button writes the subject selected from the *ComboBox* to the *ListBox*.

d. The [Number of Subjects available] button counts the number of items in the *ComboBox* and displays it in a label.

e. The [Number of Subject chosen] button counts the number of subjects in the *ListBox*.

f. The [Remove a subject] button deletes the Item clicked on In the *ListBox*.

Expand your program as follows:

- Add data validation to the [Number of subjects chosen] button to warn a user if more than seven subjects have been selected.
- Add a button that will remove a subject from the list in the *ComboBox*.

UNIT

7.2 Repetition concepts

Loops are not limited for input and output purposes only. Imagine that you want to create an algorithm in which you want to increase a counter by one until you reach 6. To do this, you could create the following algorithm:

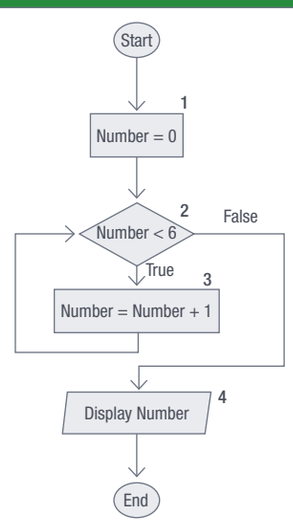
1. N = 0
2. N = N + 1
3. N = N + 1
4. N = N + 1
5. N = N + 1
6. N = N + 1
7. N = N + 1
8. Display N

While this algorithm would technically provide you with the correct answer, it would be a very clumsy way of obtaining an answer. Imagine if you need to do this 1 000 times to reach a total of 1 000!

A much more efficient method of finding the answer is to create a loop in your algorithm. This loop tells your program to repeat certain lines of code, until a condition is met.

With a loop, your program changes to:

ALGORITHM	FLOWCHART

`Number = 0`
`while number < 6`
`  Number = Number + 1`
`Display Number`
 	
Notes:
• number starts from an initial value of 0
• in this case we know that the statement Number = Number + 1 must be executed six times
• the statement Number = Number + 1 is placed inside the loop structure
• all statement/s that appear in a loop are indented
• the loop structure determines how many times it will execute
• the statement Number = Number + 1; is therefore executed six times
	

Let's trace through the flowchart:

BOX NUMBER	N	N<6	OUTPUT
1	0		
2		True	
3	1		
2		True	
3	2		
2		True	
3	3		
2		True	
3	4		
2		True	
3	5		
2		True	
3	6		
2		False	
4			6
Stop			

Take note

- Loops adhere to the ITC principle:

ITC PRINCIPLE		
Initialise	Variable/s used in loop conditions must be assigned value/s before the condition is evaluated	Number = 0
Test	Variable/s is/are tested in the loop condition	Number < 6
Change	Statements inside the body of the loop, should change the value of variable/s used in the loop condition; otherwise condition/s will always be TRUE, causing the loop to run infinitely (an infinite loop)	Number = Number + 1

- Box 2 is executed seven times
- Boxes 2 and 3 form the loop of the flowchart
- While Number<6, the condition (test) evaluates to TRUE and the body of the loop is executed. When the condition (test) evaluates to FALSE, box 4 is executed
- The condition becomes FALSE when Number becomes 6.

There are a number of different conditions that can be used to stop a loop:

- once it has run the required number of times
- once it reaches a certain value
- based on user input.

Example 7.4

Write an algorithm to find the sum of five integer values input by the user and display the sum of these values.

ALGORITHM

```
Sum = 0
loop c = 1 to 5
    Get X
    Sum = Sum + X
Display Sum
```

Sum is set to an initial value of 0

The statement loop c = 1 to 5, indicates that this is a pre-determined loop. We know that the statements in the body of the loop must be executed five times:

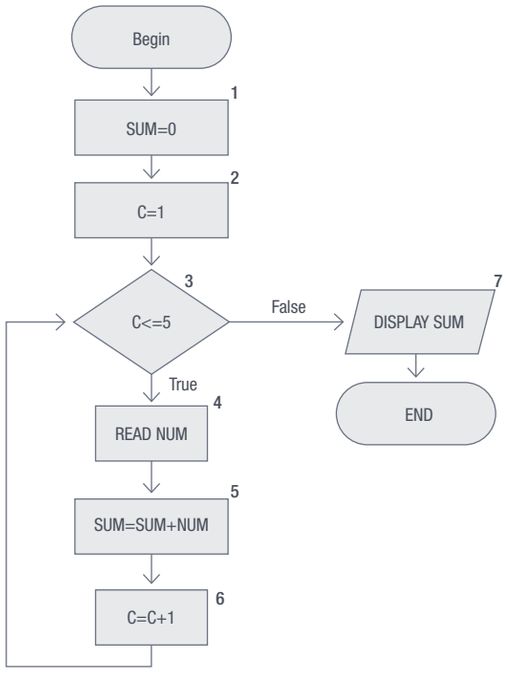
```
    Get X
    Sum = Sum + X
```

The loop variable, C, is a counter that starts with an initial value of 1. Each time the body of the loop is executed, the loop counter is incremented by 1 until C = 6

Once the counter has reached its last value, which is 6, the loop is exited and the statement that follows the loop, is executed. In this case, it is the *Display Sum* statement

The flowchart representation of the algorithm is shown on the right:

FLOWCHART



Example 7.4 *continued*

Trace through the fl owchart on the previous page using the following inputs: 8, 4, 2, 11 and 7

BOX NUMBER	SUM	C	NUM	COUNT < 5	OUTPUT
1	0				
2		1			
3				True	
4			8		
5	8				
6		2			
3				True	
4			4		
5	12				
6		3			
3				True	
4			2		
5	14				
6		4			
3				True	
4			11		
5	25				
6		5			
3				True	
4			7		
5	32				
6		6			
3				False	
7					32
Stop					

Using a loop in your algorithm allows it to be more flexible. For example, in the algorithm above, a statement could be added at beginning of the algorithm to prompt the user to enter how many numbers he or she wants to add. The integer variable, *iTimes*, storing this number, can then replace the end value, 5, in the loop and the condition (test) C <= 5, will change to C <= iTimes.

Activity 7.3

The output of this activity is a Christmas tree built up of lines of x's as shown in the figure alongside. Follow the steps below to complete this activity:

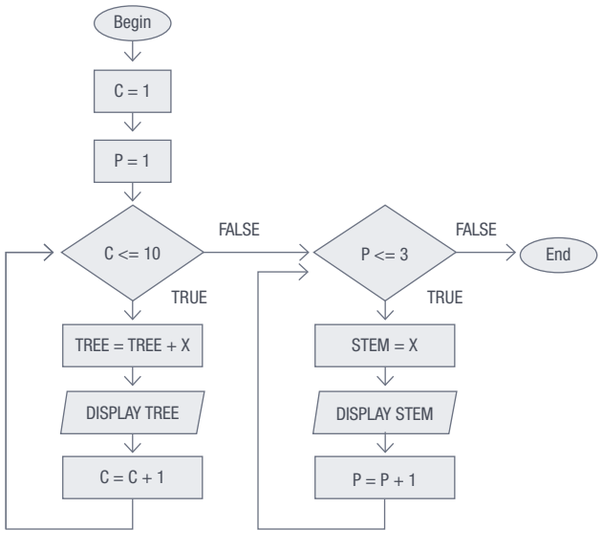
```
x
xx
xxx
xxxx
xxxxx
xxxxxx
xxxxxxx
xxxxxxxx
xxxxxxxxx
xxxxxxxxxx
x
x
x
```

7.3.1 Understand the problem:

- There are two loops, one to make the 'tree' and one to make the 'stem' of the tree.
- For the tree you need to add an x to the existing string.
- For the stem you need to add one x at a time.

7.3.2 Create a flow chart for the creation of this Christmas tree.

CHRISTMAS TREE FLOW CHART



7.3.3 Write the algorithm based on the flow chart.

ALGORITHM

```
loop C = 1 to 10
    tree = tree + x
    display tree
loop P = 1 to 3
    stem = x
    display stem
```

7.3.4 Draw a trace table for the algorithm above by completing the table below:

BOX NUMBER	C	C <= 10	TREE	P	P <= 3	STEM	OUTPUT

UNIT

7.3 FOR…Do loop

A LOOP WITH A FIXED NUMBER OF ITERATIONS

Sometimes in programming, we need to repeat the same action over and over again. At times we are aware of the number of repetitions in advance, whilst at other times, the number of repetitions depend on a specific condition.

In this unit we will focus on a known number of loop iterations/repetitions first.

THE FOR LOOP

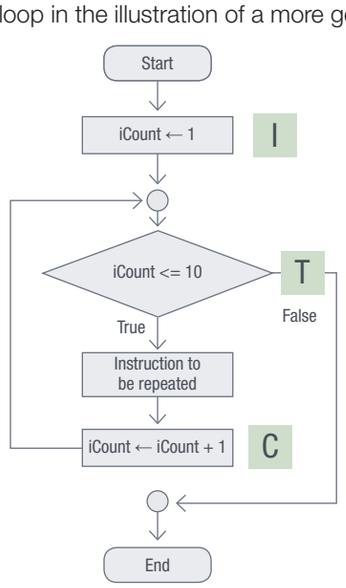
The FOR loop is represented in an algorithm below:

```
FOR iCount ← Minimum to Maximum do
BEGIN
        // 1 or more instruction(s)
END     // iCount increment
```

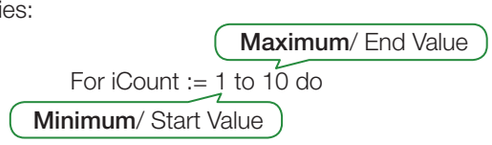
- The FOR loop needs a counter. We will call it *iCount* in the next example.
- The loop starts with the reserved/keyword word, FOR, and the same line ends with a DO.
- After the loop definition line, we have a BEGIN-END block of code, that will be executed several times.
- Counting from a minimum value to a maximum. Each iteration of the loop increases the counter *iCount* by one. If one more than the maximum value is reached by the counter, the iteration comes to an end and the statement after the loop is executed.

The FOR loop statement in an algorithm does not differ much from the Delphi syntax on the next page.

Let's follow the logic flow of the FOR loop in the illustration of a more general flowchart below:



The flowchart looks more complex than the algorithm or the Delphi code. That is because one line of code hides several steps of activities:



This line hides *three* important steps.

Let's follow the sequence of these logic steps. Take note of the summarised I-T-C steps below:

- **I for Initialize**: This step initialises the counter (*iCount*) to the value 1.
- **T for Testing**: The counter is compared against the maximum or end-value (*iCount* <= 10) to confirm that the loop still operates in the provided range. If the expression results to FALSE, the loop quits.
- **C for Change**: The third step changes the value of the counter (*iCount* ← *iCount* + 1).

In Delphi we do get two types of for-loops:

- The *incremental* FOR-loop can be identified by the command to: For *iCount* := 1 to 10 do … .
  It typically starts with a small value, which is then increased by one each time to reach the maximum or the end-value.

The Delphi source code for an incremental FOR-loop:

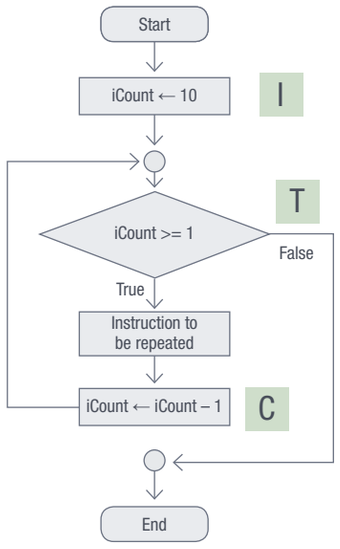
```
  Var
  iCount : Integer;
begin
  for iCount := 1 to 10 do
  begin
    // execute instruction(s)
  end;              // increment by 1 (always 1)
end;
```

- The **decremental** FOR loop can be identified by the command **downto**: For *iCount* := 10 downto 1 do … . It starts with a large (maximum) value to be decreased by one each time to reach the minimum or the end-value.

The Delphi source code for a decremental FOR loop is:

```
Var
  iCount : Integer;
begin
  for iCount := 10 downto 1 do
  begin
    // execute instruction(s)
  end;                    // decrement by 1 (always -1)
end;
```

In the decremental FOR loop, the flow of the algorithm does not change, but the *values* for the ITC steps do. Take note of the position of the arrows in the illustration below:



A Delphi FOR loop:

- loop counter can only be of **ordinal data type (Integer, Char, Boolean)**.

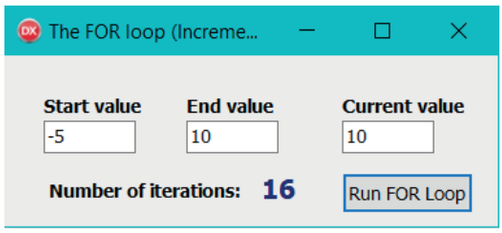
Example

```
var iCount:char;
begin
  for iCount := 'a' to 'g' do    // loop will be executed 7 times
  begin
    Statement/s to execute
  end;
  …
end;
```

- loop counter can only be increased or decreased by a single unit, that means that we cannot step-up or down by 2 or 0.5
- has a START and END value. We need to know these values before the loop is activated
- counter variable's value **cannot be changed** in the loop by **any line of code**.

Activity 7.4

Design the application for which the interface is shown alongside.



7.4.1 Add three edit boxes with descriptive labels, one button and a label (displaying "16" on the right-hand side) for counting the iterations. Give descriptive names to the components.

7.4.2 Save your application under the folder name: 07 – FOR Loop with the project name **ForLoop_p**.

7.4.3 A user must provide the values for the first two edit boxes, namely the START and END values. Provide the coding for the *OnClick* event of the button [Run FOR Loop]. The program must loop through from the minimum (START) value to the maximum (END) value.

Hints:

- For you to see the changing value in the [Current Value] edit box, pause the execution of the program for half a second each iteration. That can be achieved by using the coding below:

```
Sleep(500);      // Pauses the entire program for 500 milliseconds
                 // = half a second
```

- To force the program to refresh the form's window so that you can see the updated values, insert another command:

```
frmFormName.Refresh;    // Form's method Refresh() is enforcing a
                        // visual update of the form
```

7.4.4 Run the program.

a. Type in the Start and End values listed in the table below.

b. For each set of values, record the Current Value and the Number of Iterations once the program stops.

c. Explain the results.

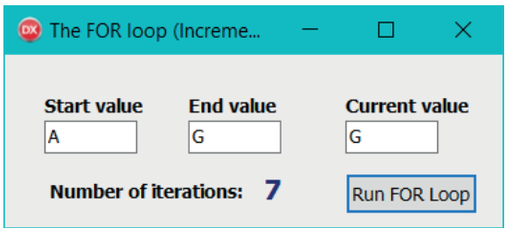
START VALUE	END VALUE	CURRENT VALUE	NUMBER OF ITERATIONS	QUESTION / EXPLANATION
–5	10	10	16	Why 16?
50	70			
1024	2024			Adapt Sleep()-value!
10	4			
–30	–10			
??	??			Add own examples

Activity 7.5

Copy 07 – FOR Loop folder and rename it to 07 – FOR Loop Letters.

7.5.1 Adapt the program slightly so that it can run through letters as shown by the interface alongside.

Hint: Do not forget to change the data types for the START and END values and change the counter variable from integer to **Char**.



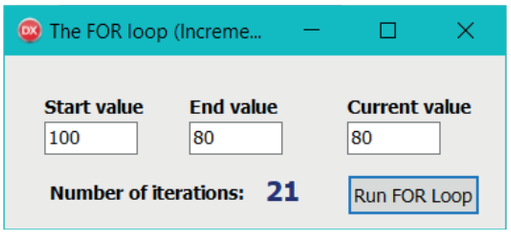
7.5.2 Experiment with more input values.

Activity 7.6

Copy the 07 – FOR Loop folder and rename the copy to 07 – FOR Loop Decrement.

7.6.1 Adapt program so that it becomes a **decremental** FOR loop as shown below.

7.6.2 Experiment with more input values.



Activity 7.7

Write down the Delphi code for following FOR loops:

7.7.1 A loop that shows the message 'Hello, Loop!' five times.

7.7.2 A loop that runs from 1 to 10 and displays the value of the counter with each repetition.

7.7.3 A loop that shows the first twelve multiples of 5.

7.7.4 A loop that calculates the sum of the first 1000 numbers and displays the total once the loop has completed.

Activity 7.8

Write the following FOR loops. A FOR loop that iterates from:

7.8.1 0 to 10 and shows each value.

7.8.2 1 to 100 and shows the square root of each value.

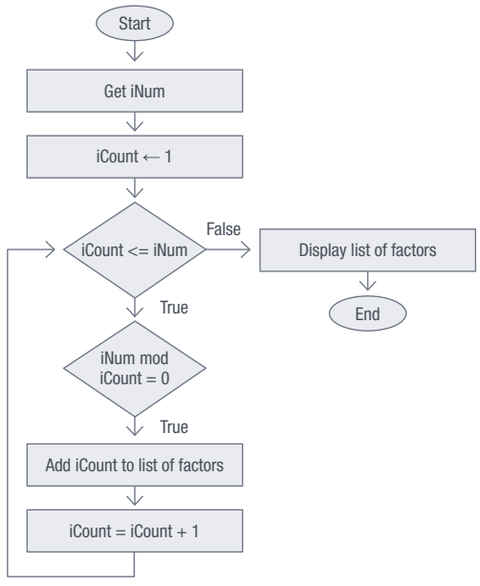
7.8.3 1 to 50 and increases the value of x by 2 inside each loop.

Activity 7.9 Find the factors of a number

A factor of a number is a number that can divide into a number with no remainder, for example, 3 is a factor of 6, because 6 ÷ 3 = 2 without a remainder. However, 3 is not a factor of 5, because 5 ÷ 3 = 1 rem 2.

Use a FOR loop to find the factors of a number.

7.9.1 Study the flow chart below and then write the algorithm for this program.



7.9.2 Factorials are the product of all consecutive numbers, starting from 1, that is, $1 \times 2 \times 3 \ldots n$ where n is a positive integer number. To determine five factorial, $5! = 5 \times 4 \times 3 \times 2 \times 1$

Understand the problem:

- I need to multiply all the numbers that make up the number.
- These numbers count down from the number, for example, 5, 4, 3, 2, 1, or count up from 1 to n.

a. Why can you use a loop to shorten your code?

b. What type of loop must you use to solve this problem?

c. Write the algorithm to find the factorials of a given number.

d. Write a program to determine the factorials of a number entered by the user.

DEVELOPING HIGH QUALITY ALGORITHMS

In Chapter 1 you learned about the principles of a quality algorithm. One of the principles that determined the quality of an algorithm is called the order of the algorithm. The order of an algorithm is a measure of the number of steps that are needed to solve a problem. The fewer steps that are needed, the higher the quality of the algorithm.

The code to determine the factors of a number:

```
…
iNumber := StrToInt(edtNumber.Text);
memDisplay.Lines.Add('The factors of '+IntToStr(iNumber));
for iCount := 1 to iNumber do
begin
  if iNumber mod iCount = 0 then
     memDisplay.Lines.Add(IntToStr(iCount);
end;
…
```

Note:

- In this code all the numbers from 1 to the value of *iNumber* will be checked to determine whether they are a factor of *iNumber* or not.
- For example, if *iNumber* is 50, then each value from 1 to 50 will be checked to determine whether it is a factor 50 or not.

Work through the following example to see how the factor finder algorithm can be improved.

Example 7.5 Improved factor finder (algorithm)

To understand how the algorithm can be improved, you need to look at the factors of a few numbers:

- Factors of 10 : 1, 2, 5, 10
- Factors of 12 : 1, 2, 3, 4, 6, 12
- Factors of 27 : 1, 3, 9, 27

Can you identify a pattern from these numbers?

In each case, the product of two factors is equal to the number. Stated differently, the number divided by the one factor will always give you another factor.

To see how this works, divide the number 10 by each of its factors:

- $\frac{10}{1} = 10$ (this means that 1 and 10 are factors)
- $\frac{10}{2} = 5$ (this means that 2 and 5 are factors)
- $\frac{10}{5} = 2$ (this means that 5 and 2 are factors)
- $\frac{10}{10} = 1$ (this means that 10 and 1 are factors).

Every factor you find is therefore directly linked to a second factor, as shown below.

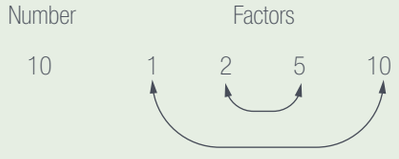


Figure 7.2: *The relationship between the factors of 10*

Example 7.5 Improved factor finder (algorithm) *continued*

This suggests that you do not need to compare all the values from 1 to the number itself to find its factors. Instead, you (at most) need to evaluate half the numbers. In fact, if you look at the factors for the number 9, you will see that, in order to identify all the factors, you only need to compare the numbers up to the square root of the number.

The factors for 9 are 1, 3 and 9. Dividing the number by each of the factors you see that:

- $\frac{9}{1} = 9$ (this means that 1 and 9 are factors)
- $\frac{9}{3} = 3$ (this means that 3 is a factor)
- $\frac{9}{9} = 1$ (this means that 9 and 1 are factors)

This relationship is shown in the image below.

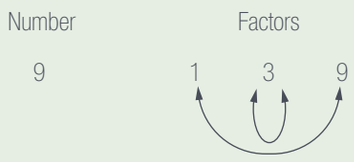


Figure 7.3: *The relationship between the factors of 9*

As this example shows, the biggest factor you need to evaluate is the square root of the number itself. Any number larger than the square root can be found by dividing the number by a smaller factor. So, you can update the factor finder program as follows:

```
begin
  memDisplay.Lines.Clear;
    iNumber := StrToInt(edtNumber.Text);
    memDisplay.Lines.Add('The factors of '+IntToStr(iNumber));
    iEnd := Trunc(Sqrt(iNumber));
    for iCount := 1 to iEnd do
    begin
      if iNumber mod iCount = 0 then
      begin
             iFactor1 := iCount;
             iFactor2 := iNumber div iCount;
             if iFactor1 = iFactor2 then
                  memDisplay.Lines.Add(IntToStr(iCount))
             else
               begin
                   memDisplay.Lines.Add(IntToStr(iCount));
                   memDisplay.Lines.Add(intToStr(iFactor2));
               end;
      end;
    end;

end;
```

Activity 7.10 Improve your Fibonacci sequence program

In Chapter 4 Consolidation Activity 16 you wrote the code to determine a Fibonacci sequence by clicking on the button each time. Open the program you wrote.

7.10.1 Open the project and study the algorithm you wrote:

- **a.** How can you shorten or improve it?
- **b.** Think about how you can improve the way in which this program works. Is it best to have the user click the button repeatedly to determine the length of the sequence?
- **c.** Rewrite the algorithm to improve it.

7.10.2 Change your program to implement your improvements.

7.10.3 Add data validation to ensure that the input from the user is valid.

UNIT

7.4 Looping with components

Earlier in this chapter you learnt about *ListBoxes*, *ComboBoxes* and *RadioButtons*. Look at the examples in the illustration below:

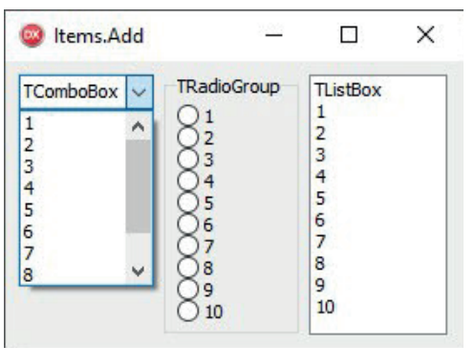


Figure 7.4: *Example showing a ListBox, RadioButtons and ComboBox*

The following is true for all three components:

- The first element is stored in ItemIndex 0.
- You can access any items from the components using the *Items* property. For example:
  - `sNum1:=TListBox.Items[0];` `// will return the string 1`
  - `iNum2:=StrToInt(TlistBox.Items[4]);` `// will return the integer value 5`
  - `sNum3:=TComboBox.Items[2] ;` `//will return the string 3`
  - `iNum4:=StrToInt(TcomboBox.Items[6]) ;` `//will return the integer value 7`
  - `sNum5:=TRadioGroup.Items[5];` `// will return the string 6`
  - `iNum6:= StrToInt(TRadioGroup.Items[5]);` `// will return the integer value 6`
- Items can be added to the components using the format:

```
componentName.Items.Add();
```

- To determine the number of items found in the components:

```
iCount := ComponentName.Items.Count;
```

ADDING ITEMS TO A LISTBOX

To add several lines to a *ListBox* at run time, we can use a loop. For example, the following code will add four lines to the *ListBox lstData*:

```
    For X := 1 to 4 do
      lstData.Items.Add('Item number: ' + IntToStr(X));
```

The values 1,2, 3 and 4 are added to the *ListBox lstData*.

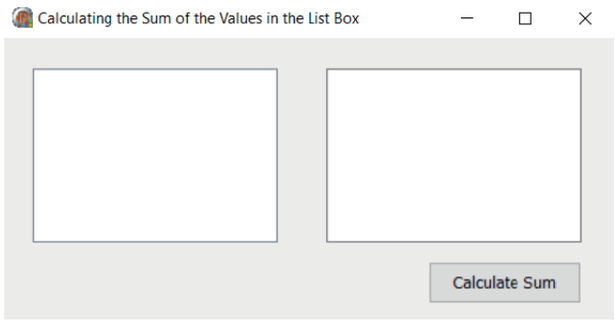
RETRIEVING THE INFORMATION FROM A LISTBOX'S ITEM

We are also able to retrieve several values from a *ListBox* at run time. Let's work through the following guided activity to help you understand how to do this.

Guided activity 7.1 Retrieving information from a *ListBox*

Add the first five integer values starting from 1 to a ListBox. Determine the sum of the values in the ListBox and display the sum with a meaningful message in the memo box memDisplay.

7.1.1 Open the **CalculateListBoxValuesSum_p** project in the 07 – ListBoxSum folder. You should see the following interface.



7.1.2 Create an OnClick event for the [Calculate Sum] button and do the following:

- Add code to store the five randomly generated integers from 0 to 99 in the *lstData ListBox*:

```
For iCount := 1 to 5 do
    lstData.Items.Add(IntToStr(Random(100));
```

  - The first randomly generated number is stored in *ItemIndex* position 0, the second randomly generated number is stored in *ItemIndex* position 1,…, the fifth randomly generated number is stored in *ItemIndex* position 4
  - In general, the n^{th} item in a *ListBox* is stored in *ItemIndex* position (*n*-1)
- Set *iSum* to zero

```
iSum=0;
```

- Read an item from the *lstData* one at a time and add it to the sum:

```
for iCount := 0 to lstValues.Count - 1 do
    begin
      iNum:=StrToInt(lstValues.Items[iCount]);
      iSum:=iSum+iNum;
end;
```

NOTE:

- The loop counter *iCount* starts from 0 because the first element in the *ListBox* has the *ItemIndex* value 0.
- The maximum value is set to *lstValue.count* -1. *lstValue.Count* returns the number of items in the *ListBox*. So if there are five items in the list, then *lstValue.Count* will return 5. Counting from 0 will give the fifth *ItemIndex* the value 4, so 1 must be subtracted from *lstValue.count* to give the correct value for the last *ItemIndex*.

```
iNum:=StrToInt(lstValues.Items[iCount]);
```

- An item is retrieved from the list at the specified *itemIndex* and converted to integer and stored in variable *iNum:*

```
iSum:=iSum+iNum;  //iSum stores a running total
```

- The value of *iNum* is added to the value of *iSum:*

```
memoDisplay.Lines.Clear;
memDisplay.Lines.Add('The sum of the first 5 randomly generated
numbers is:'+IntToStr(iSum));
```

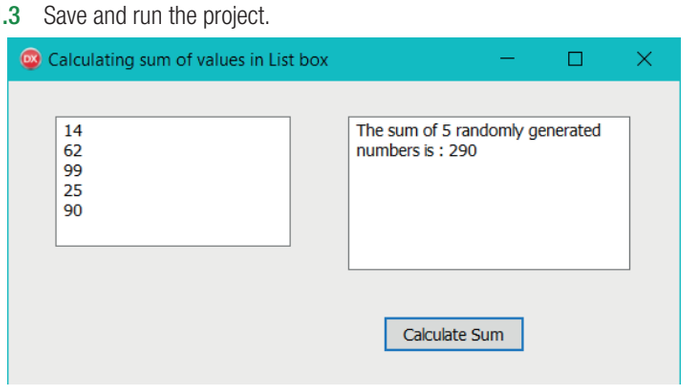
- This clears the *memoDisplay* memo box and displays the message 'The sum of the 5 randomly generated numbers is: ' and the value of the sum.

Guided activity 7.1 Retrieving information from a *ListBox* *continued*

- The code for the *OnClick* event [Calculate Sum] button is:

```
procedure TForm1.btnCalculateSumClick(Sender: TObject);
var iCount,iSum,iNum:Integer;
begin
  for iCount := 1 to 5 do
    lstValues.Items.Add(IntToStr(Random(100)));
  iSum:=0;
  for iCount := 0 to lstValues.Count - 1 do
     begin
       iNum:=StrToInt(lstValues.Items[iCount]);
       iSum:=iSum+iNum;
     end;
  memDisplay.Lines.Clear;
  memDisplay.Lines.Add('The sum of the first 5 randomly generate numbers
                       is :'+IntToStr(iSum));
end;
```

7.1.3 Save and run the project.



Activity 7.11

7.11.1 Open the Delphi project created in the previous activity.

- **a.** Add a button to the form. Call the button *btnAverage*.
- **b.** Add code to find the average of the numbers in the *ListBox*.
- **c.** Add a button called *btnHighest* to the form.
- **d.** Write the code for *btnHighest* to determine the highest number in the *ListBox*.
- **e.** Add a button called *btnLowest* to the form.
- **f.** Add the code to the button to find the lowest number in the list.
- **g.** Display these values in the memo component on the form.

7.11.2 Open the **Scoring_p** project used for scoring a Diving competition that you made in Chapter 4 and expand it as follows:

- **a.** Add two more edit boxes to allow for five judges to score the competitors.
- **b.** Change the code for the [Final Score] button to:
  - **i.** exclude the highest and the lowest numbers.
  - **ii.** find the average of the remaining three scores.
  - **iii.** display the final score.

UNIT

7.5 Using the Input box

Another function you can use to get data input is the *InputBox* function. The *InputBox* function displays an input dialog box that a user can use to enter a string, double or integer when the program is executed.

The syntax of an *InputBox* is:

```
InputBox(label,prompt string,default value)
```

The *InputBox* has three arguments. These are:

- **Label:** Refers to the caption of the *Input Dialogue box.*
- **Prompt string:** Refers to the text that prompts the user to enter input in the edit box.
- **Default value:** Refers to the value that appears in the *Edit* box when the *Input Dialog box* first appears.

Take note

The default value can be blank and is shown as ''.

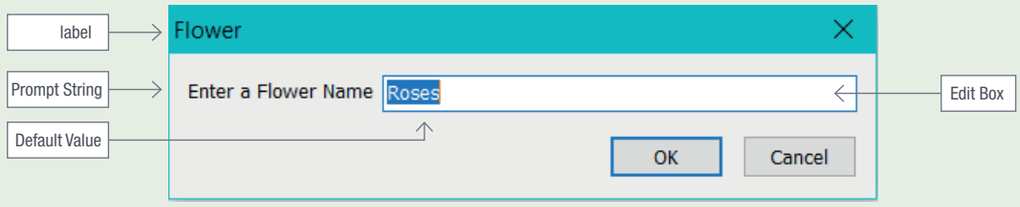
Since the *InputBox* is a function, it must be assigned to a variable. Let's look at an example of this.

Example 7.6

Execute the following code:

```
sFlower:=InputBox('Flower','Enter a Flower Name','Roses');
```

When the above line of code is executed, the *Input Dialog* box will appear:



The *InputBox* function always returns a string and hence for integer and real data the results must be converted:

- iGrade:=StrToInt(InputBox('Grade','Enter Grade','10'));
- rNum:=StrToFloat(InputBox('Amount','Enter an Amount,''));
- cResponse:=InputBox('Flight Seating','Enter your flight no','')[1];
  The [1] following the closing round bracket indicates to Delphi to use only the first character of the string returned by the InputBox.

Example 7.7 Using the Input Box to read numbers

Prompt the user to enter 10 integer numbers. Determine and display the average of these numbers.

```
…
iSum:=0;
For iCount:=1 to 10 do
Begin
  iNum:=StrToInt(InputBox('Integer Number','Enter an integer
  number',''));
  iSum:=iSum+iNum;
end;
rAverage:=iSum/10;
ShowMessage('The average is: '+FloatToStrF(rAverage,ffFixed,8,2));
…
```

Activity 7.12

7.12.1 Study the code segment and then answer the questions that follow:

```
Begin
        …
Line 1  iNum:=StrToInt(InputBox('Chips','How many do you Require?','2'));
Line 2  cResponse:=InputBox('Response','Will you be coming?','')[1];
Line 3  memDisplay.Lines.Add('Number of Chips: '+IntToStr(iNum));
Line 4  memDisplay.Lines.Add('Attending: '+cResponse);
        …
End;
```

a. In Line 1, identify the following in the *InputBox* function:

- label
- prompt string
- default value

b. Explain what the purpose of a default value is.

c. How does a blank default value appear in the default value argument?

d. Why is it necessary for the data conversion in Line 1? Explain your answer.

e. In Line 2:

- What is the purpose of [1] at the end of the line?
- Explain what will happen if the [1] is changed to [2].
- What type will the variable *cResponse* be declared as? Explain.

7.12.2 Open the **UIFPayment_p** project from the 07 – UIF Payments folder. Unemployment Insurance Fund (UIF) contributions must be paid on a monthly basis to the UIF or to SARS.

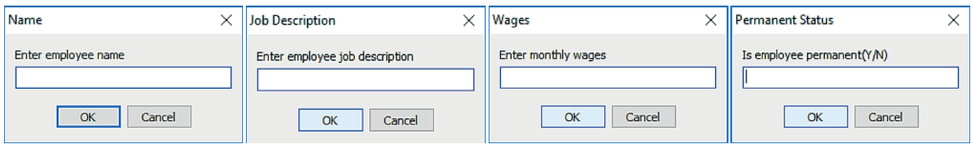
UIF contributions are calculated at 2% of the employee's wage of which the employee must pay 1% and the employer must pay 1%.

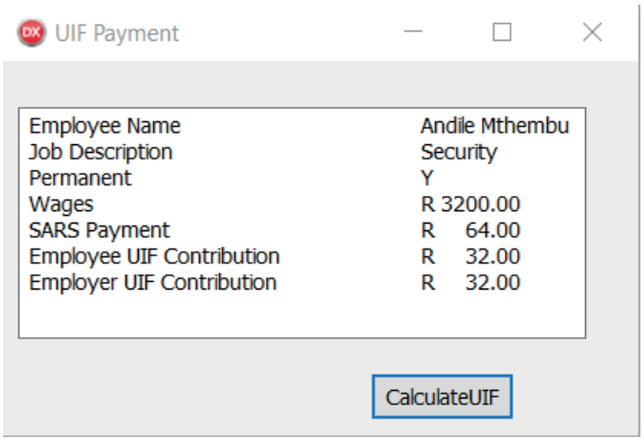
a. Input the employee's name, job description, wage amount and a character of 'Y' for permanent, and 'N' for not permanent employee.

b. When the [CalculateUIF] button is clicked, display *InputBoxes* to obtain the inputs.

Activity 7.12 *continued*

c. Calculate the amount of UIF to be paid and display employee name, job description, permanent status, amount UIF to be paid to SARS, amount UIF payable by employee and employer.





Activity 7.13

A teacher has 24 learners in her class. Create a project, **Mark Statistics_p** that will do the following:

7.13.1 Write code for the [TestTotal] button so that when it is clicked an *InputBox* appears, allowing the user to give the maximum mark for the test.

7.13.2 Write code for the [Capture] button so that an *InputBox* appears when it is clicked, allowing the user to enter the marks. This command must be placed in a loop to capture the marks of all the learners in the class. The marks are displayed in a *ListBox*.

7.13.3 Add data validation to prevent the user from entering a number smaller than 0 or larger than the maximum of the test.

7.13.4 Write code for the [Statistics] button to display the following information on the *TMemo* component:

a. number of learners who wrote the test

b. the average of the test, expressed as a percentage.

c. the highest mark, expressed as a percentage.

d. the lowest mark, expressed as a percentage.

e. the number of learners who achieved a mark above 80%

f. the number of learners who failed the test, given that below 40% is a fail mark.

UNIT

7.6 REPEAT…UNTIL loop

A REPEAT…UNTIL loop has its test at the END of a loop-block. That means **one iteration is always executed** before the condition is tested. The REPEAT…UNTIL loop is called a **post-conditional loop** because it checks whether it should continue running at the end of each loop.

It uses the following syntax:

REPEAT…UNTIL loop syntax

```
Repeat
  // one or more instruction
Until (condition=true);
```

Note:

- The REPEAT…UNTIL loop structure does not need a BEGIN ... END block.
- The REPEAT marks the beginning of the loop and UNTIL marks the end of the loop.
- The statements in the loop body are executed repeatedly until the condition (a Boolean expression) evaluates to TRUE, that is, the loop body executes when the condition is FALSE and terminates when the condition becomes TRUE.
- The condition is tested only after the loop body has been executed.
- The loop body is executed at least once.

```
Var
  iCount : Integer;
begin
  iCount := 0.5;
  Repeat

    // execute instruction(s)
    iCount := iCount + 0.5;
  Until (iCount > 9.5);
end;
```

Initialize Start Value

Change In-/Decrement

Test Condition

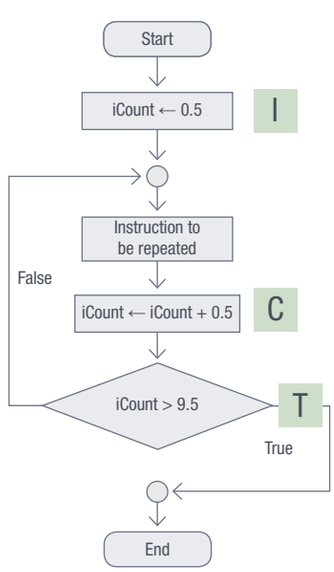


Figure 7.5: *The REPEAT-UNTIL loop*

CHANGES BROUGHT BY THE REPEAT-UNTIL LOOP

The new sequence of logical steps **I-C-T** (initialise, change and test) instead of I-T-C.
In the flowchart and code segment above:

- I : iCount:=0.5
- C : iCount:=iCount+0.5
- T : iCount>9.5

The initial value is set, then the change takes place and then the test.

- Unlike the FOR-loop, the REPEAT-UNTIL loop's control variable does not have to be an ordinal data type, that is it can be *Double*, *String*, and so on. You can step through fractional parts of numbers or any string data changes.
- The START should be fixed before entering a REPEAT-UNTIL loop, but the END value (part of the condition) **must be changed during the execution** of the loop-block.
- An integer value can be changed by a value equal to or greater than one.

Example 7.8

Double a number until the number was larger than 1 000.

You could create the following REPEAT..UNTIL loop..

Doubling a number

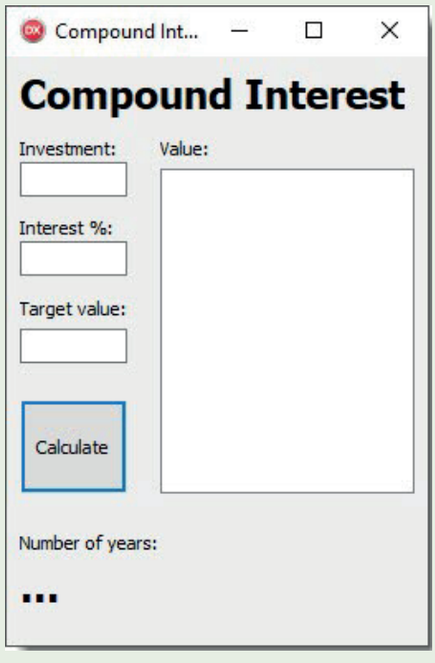
```
iInput := 5;
Repeat
  iInput := iInput * 2;
Until iInput >= 1000;
ShowMessage(IntToStr(iInput));
```

Example 7.9 Compound interest calculator

Whenever you invest your money (principal) in the bank, you earn interest. This interest is added to your principal amount and your investment grows. This is known as compounding. The compounded amount becomes the principal amount on which the next interest will be calculated.

A client invests an amount (principal) at a fixed interest rate. The client has a target amount to reach with his or her investment. Enter the principal amount to invest, the interest rate and the target value that the client wants to reach. Calculate and display how many years it will take the client to reach his or her target amount. Also display the year and the amount accumulated at the end of that year.

1. Open the **CompoundInterest_p** project located in the 07 – Compound Interest folder.



2. Create an *OnClick* event for the Calculate button.

Example 7.9 Compound interest calculator *continued*

3. In the [Calculate] button event handler, declare the following seven local variables:

Variable	Purpose
rInvestment	Stores the initial investment from the edtInvestment edit box
rInterest	Stores the interest percentage from the edtInterest edit box
rTarget	Stores the target value from the edtTarget edit box
iYear	Counter variable to store the number of years
rValue	Initialised to the rInvestment amount
rGrowth	Stores the interest amount calculated for a year
sValue	Stores the compound value as a string for display

```
  var
    iYear  : Integer;
    rInvestment, rTarget,rInterest, rGrowth, rValue : Real;
    sValue : String;
.
```

4. Assign the values entered into the *Investment*, *Interest* and *Target value* textboxes to the appropriate variables.

```
    rInvestment := StrToFloat(edtInvestment.Text);
    rInterest := StrToFloat(edtInterest.Text) / 100;
    // interest rate divided by 100
    rTarget := StrToInt(edtTarget.Text);
```

5. Set value of *iYear* to 0 and the value of *rValue* to the value of *rInvestment.* Your variable initialisation and assignment should now look as follows:

```
    iYear := 0;
    rValue := rInvestment;
```

6. Clear the values of the *ListBox* using the *Items.Clear* method. Your variable initialisation and assignment should now look as follows:

```
    lstValues.Items.Clear;
```

7. Create a REPEAT…UNTIL loop that will repeat until *rValue* is larger than or equal to *iTarget*.

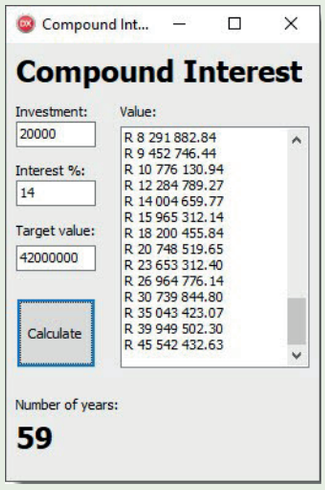
```
repeat                              // repeat body of loop
  rGrowth := rValue * rInterest;    // calculates the interest earned
  rValue := rValue + rGrowth;       // The interest is added to rValue
  // convert the compound amount to a string in currency format
  sValue := FloatToStrF(rValue,ffCurrency,10,2);
  lstValues.Items.Add(sValue);      // displays the converted string in
  iYear := iYear + 1;               // the ListBox
until rValue >= rTarget;            // increments the years by 1
                                    // until rValue>=rTarget evaluates
                                    // to true
```

8. Display the number of years it will take the client to reach his or her target amount.

```
  lblResult.Caption := IntToStr(iYear);
```

Example 7.9 Compound interest calculator *continued*

9. Save and run the project.



Let's now apply the conditional REPEAT-UNTIL loop structure to a more mathematical challenge.

Example 7.10

The Lowest Common Multiple (LCM) or Least Common Divisor (as some call it) can demonstrate the use of a conditional REPEAT-UNTIL loop. In mathematics the LCM is the **smallest positive number** that is a **multiple of both numbers**. For example: Take the two numbers 6 and 15. Multiple values for each respectively would be defined as:

6: 12, 18, 24, **30**, 36, 42, 48, 54, **60**, 66, 72, 78, 84, **90**, etc

15: **30**, 45, **60**, 75, **90**, etc

Matching multiples between 6 and 15 are coloured in red and bold. The **smallest** or **lowest** matching multiple is 30. The LCM of 6 and 15 is 30.

How can we translate this into a program with Delphi code to calculate the LCM?

If we step through the multiples of **one** number (for example, 0, 15, 30, 45, etc) and at the same time test whether a remainder is left if we divide with the second number, that is, if the division with the second number delivers a remainder of 0 (zero), then we know this number is a matching multiple of both!

An algorithm in pseudo code could look as follows:

Lowest Common Multiple

```
BEGIN
    INPUT Num1                              // set Num1 15 or 6
    INPUT Num2                              // set Num2 6 or 15
    LCMnum ← 0                              // start from zero
    REPEAT
        LCMnum ← LCMnum  + Num1             // step through: 15, 30, 45, etc.
    UNTIL ((LCMnum MOD Num2) = 0)           // or 6, 12, 18, 24, etc.
    OUTPUT LCMnum is the calculated LCM
END
```

Example 7.10 *continued*

Translating this algorithm into code we would get the following possible result:

```
begin
  1 iNum1 := StrToInt(edtNum1.Text);
  2 iNum2 := StrToInt(edtNum2.Text);
  3 iLCM := 0;
  Repeat
  4  iLCM := iLCM + iNum1
  5 Until (iLCM mod iNum2) = 0;
  6 lblResult.Caption := 'The LCM between ' + edtNum1.Text + ' and '
    + edtNum2.Text + ' is: ' + IntToStr(iLCM);
end;
```

Tracing the steps line by line in a trace table will make it even clearer.

LINE #	iNUM1	iNUM2	iLCM	(iLCM MOD iNUM2)	= 0 ?	COMMENT
1, 2, 3	15	6	0			
4			15			
5				(15 MOD 6) → 3	False	Keep iterating
4			30			
5				(30 MOD 6) → 0	True	
6						Loop terminated: LCM ← 30

Did you know

You can use this program to help you calculate the 'Common Denominator' even with very large numbers!

Activity 7.14

Update the compound interest calculator you created in the example on page 179 so that it can do the following:

7.14.1 Ask the user to enter the number of years to invest.

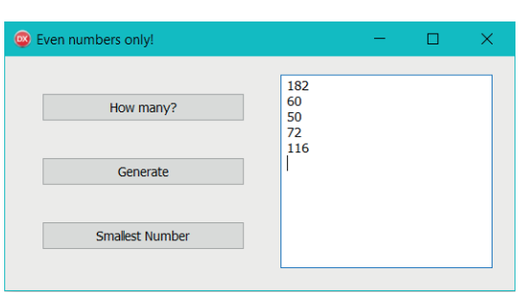
7.14.2 Calculate the value of the investment after the number of years.

7.14.3 Ask the user to enter an amount to invest each year. This value must be added to the total investment at the end of each year after the interest has already been added.

7.14.4 Display the total money invested as well as the total value of the investment at the end of the number of years.

Activity 7.15

Write a new Delphi application that will display only even random numbers between 50 and 200, by following the guidelines below:



7.15.1 Allow the user to input the number of even numbers to be displayed when [How Many] button is clicked.

7.15.2 Use a REPEAT-UNTIL loop to enter as many even random numbers as specified by the user. Display the numbers in the *Memo* component.

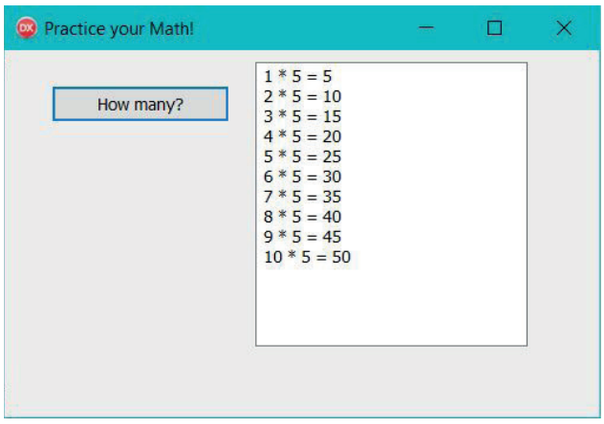
7.15.3 Write code for the [Smallest Number] button to display the smallest number in a *ShowMessage* dialog.

Activity 7.16 Practise your multiplication

You want to create a Delphi application to help your younger brother practise his Math. You want to start with multiplication only. He is young, so you decide that the largest product in his times table should be 50.

Study the interface shown alongside.

Note: When the [Times Table] button is clicked an *Inputbox* asks the user which table must be displayed.



7.16.1 Write the code to display the output as shown.

7.16.2 Use a REPEAT-UNTIL loop that will stop when the product exceeds 50.

Extension:

7.16.3 Add one more button and an InputBox to allow the user to input the highest value for the product.

UNIT

7.7 While…do loop

The WHILE loop does not necessarily run a specific number of times. Instead, the WHILE loop is a **conditional** loop (like the REPEAT loop), but it places its condition first before executing the looping block. Only if the condition is satisfied, will the loop body execute. This means that the loop body executes while the condition is TRUE and exits the loop when the condition is FALSE.

The WHILE loop is called a pre-conditional loop because the condition that determines whether it should run is found at the start of the loop. If the condition is TRUE, the loop activates and continues to repeat until the condition is no longer TRUE. If the condition is not met initially, the entire loop is skipped.

The syntax for the while-loop is:

WHILE..DO loop syntax

```
WHILE (condition = True) do
BEGIN
  // 1 or more instruction
END   // Test Condition again
```

Here are all the steps explicitly written in coding:

```
Var
  iCount : Integer;
begin
  iCount := 0.5;
  while (iCount <= 9.5) do
  begin
    // execute instruction(s)
    iCount := iCount + 0.5;
  end;
end;
```

Initialize Start Value

Test Condition

Change In-/Decrement

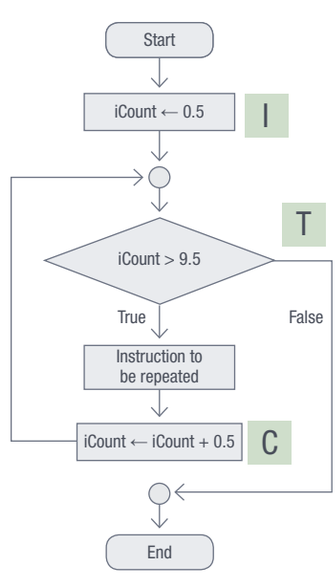


Figure 7.6: *Coding and flowchart for a WHILE loop*

Note:

- The loop starts with the keyword WHILE followed by the condition and then the keyword DO.
- DO is not followed by a semi-colon.
- The body of the loop appears within a BEGIN END block.
- The loop executes while the condition is TRUE and exits the loop once the condition evaluates to FALSE.
- The loop condition is tested at the beginning of the loop. If the loop condition evaluates to FALSE upon entering the loop then the loop is not executed at all.

PROGRAMMING ADVANTAGES USING A WHILE LOOP

The sequence of logical steps I-T-C are the same as in a FOR loop. That is:

- I: iCount:=0.5
- T: iCount<=9.5
- C: iCount:=iCount+0.5 // if there is no change, then we would have an infinite loop

The hidden loop-counter test (from the FOR loop) is replaced by a **Boolean expression**. Any expression that resolves into a **Boolean result** is acceptable.

The loop control variable (if used as a counter) can be about any data type e.g.: Double, String. For Double the change can include fractional parts of numbers. Integers can be increased or decreased by any amount.

The start should be fixed before entering a WHILE loop, but the **end value** (part of the condition) **must be changed during the execution** of the loop-block.

Example 7.11

Double a number until the number was larger than 1 000.

To code the same example using the WHILE..DO loop, use the following code snippet:

While..Do loop example

```
iValue := StrToInt(InputBox('Integer Value','Enter an integer value',''));
While iValue <= 1000 do
begin
  iValue := iValue * 2;
  ShowMessage(IntToStr(iValue));
end;
```

Note:

- The While…Do loop in this example will continue multiplying iValue by 2, until the result is larger than 1000.
- When the input number is 5, the loop will run 8 times.
- If the input number is 500, the loop will only run twice.
- If the input number is 5000, the loop will not run.

Example 7.12

Determine how many multiples of a number *iNum* must be added together to give a sum just greater than 100. The value of *iNum* must be randomly generated in the range 5 to 10. For example, if the random number 7 was generated, then the multiples of 7 are 7, 14, 21, 28, …

```
Begin
…
1   iSum:=0;
2   iCount:=0;
3   Randomize;
4   iNum ;= random(6) + 5
5   iMultiple:=iNum;
6   While iSum<=100 do
7    Begin
8     iCount:=iCount+1;
```

Example 7.12 *continued*

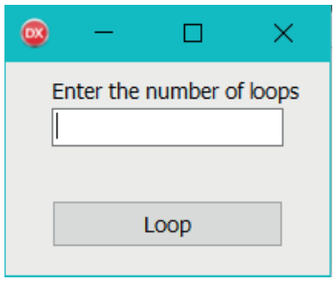
```
9     iSum:=iSum+iMultiple;
10    iMultiple:=iMultiple+iNum;
11  End;
12  memDisplay(IntToStr(iCount),' multiples of ',IntToStr(iNum),' must be
    added together');
….
End;
```

Note:

- Line 1: The sum variable *iSum* is initialised to 0. iSum :=0 refers to the I in the ITC principle
- Line 2: The counter *iCount* to count the number of multiples that must be added is set to 0
- Line 3: The random generator is activated
- Line 4: A random number in the range 5 to 10 is generated
- Line 5: The initial value of *iNum* is assigned to *iMultiple*
- Line 6: The condition iSum<=100 is tested. This refers to the T in the ITC principle. If the condition evaluates to TRUE, then the body of the loop will be executed. If the condition evaluates to FALSE, then the loop is exited and the statement in line 12 is executed
- Line7: Marks the beginning of the loop
- Line 8: The counter *iCount* is incremented by 1
- Line 9:The *iMultiple* value is added to *iSum*. iSum:=iSum+iMultiple; refers to the C in the ITC principle
- Line 10: The next multiple of *iNum* is generated in *iMultiple*
- Line 11: Makes the end of the loop and takes us back up to Line 6, where the condition is tested again.

Activity 7.17

Create a project called **WhileLoopCounter_p** in the folder 07 – While Loop Counter and create the user interface as shown below in Design view and do the following:



7.17.1 Create an *OnClick* event for the [Loop] button to read the number of times the loop must execute from the *Edit* component and store the value in *iTimes* variable.

7.17.2 Use a WHILE loop to execute the loop *iTimes*.

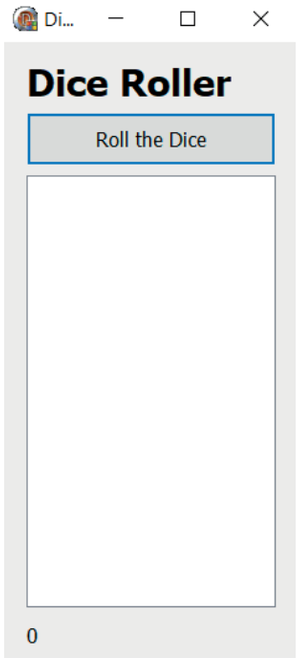
7.17.3 Display the value of the loop counter each time the loop is executed in a Message Box.

7.17.4 Save and run the project.

Guided activity 7.2 Dice Roller

You need to create an application that will calculate how many times you need to roll two die before you roll a 12.

7.2.1 Open the **DiceRoller_p** project located in the 07 – Dice Roller folder.



7.2.2 You should see the following code in the *OnClick* event for the [Roll The Dice] button.

Existing code

```
procedure TfrmDiceRoller.btnRollTheDiceClick(Sender: TObject);
var
    iCount, iDice1, iDice2, iTotal : Integer;
begin
    Randomize;
    lstRolls.Items.Clear;
    iCount := 1;
    iDice1 := Random(6) + 1;
    iDice2 := Random(6) + 1;
    iTotal := iDice1+iDice2; //the sum of the values on the dice are added
    lstRolls.Items.Add(IntToStr(iTotal));
    //Start while loop

    lstRolls.Items.Add(IntToStr(iTotal));
    iCount := iCount + 1;
    //End while loop

    lblRolls.Caption := 'Number of rolls: ' + IntToStr(iCount);
end;
```

In the available space, create a WHILE loop that will repeatedly roll two dice (returning a random value between 1 and 6 for each dice) until the sum of the two dice is equal to 12. With each repetition, the value of *iCount* increases by 1. Once the loop stops, this value will be written to the label at the bottom of the screen.

Guided activity 7.2 Dice Roller *continued*

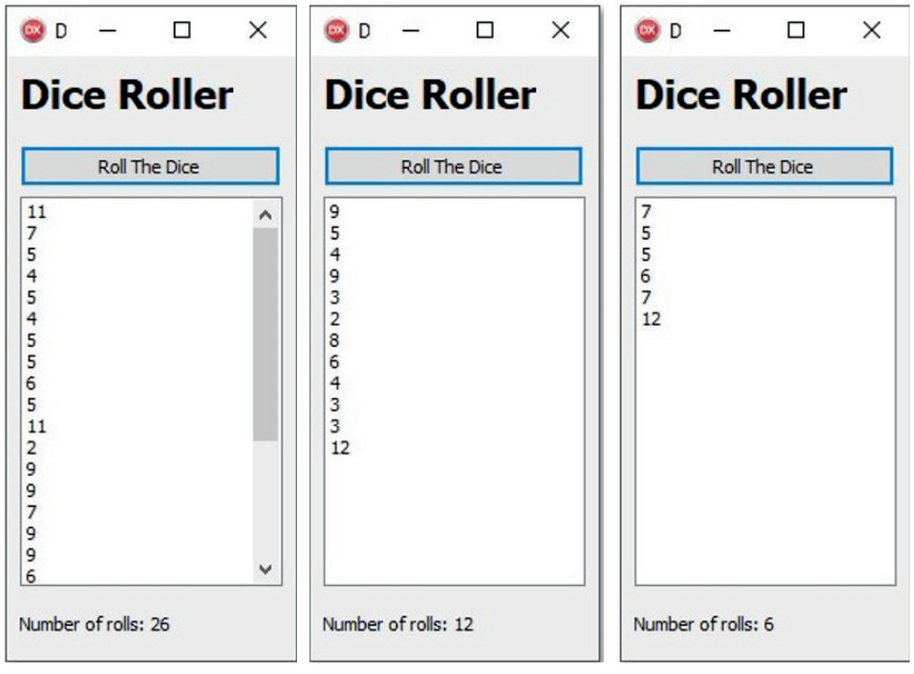
To solve this problem, you can create the following while-loop.

Dice Roller while-loop

```
while iTotal <> 12 do
begin
  iDice1 := Random(6) + 1;
  iDice2 := Random(6) + 1;
  iTotal := iDice1 + iDice2;
  lstRolls.Items.Add(IntToStr(iTotal));
  iCount := iCount + 1;
end;
```

- Before the WHILE loop, iDice1 and iDice2 are 'rolled' out of the loop and the total of the dices is stored in *iTotal*
- The loop will execute as long as *iTotal* is not equal to 12
- Once inside the loop, iDice1 and iDice2 are 'rolled' again
- The total of both dices are stored *iTotal*
- *iCount* is incremented by 1 to keep track of the number of dice rolls
- The loop will repeat as long as the *iTotal* is not equal to 12.

7.2.3 Save and run the project. By clicking the [Roll The Dice] button a few times, you will see that the WHILE loop repeats a different number of times with each click. This is because random numbers are generated, it takes a different number of rolls for the total to be equal to 12.



7.2.4 Change the code in the Onclick event handler to achieve the same results using a REPEAT…UNTIL loop.

CHOOSING BETWEEN A FOR LOOP AND A WHILE LOOP

The FOR loop can be replaced with an equivalent WHILE loop, but the opposite is not true.

Example 7.13

Two values are read from spin edits sedLower and sedUpper. Determine the sum of the values from *sedLower* to *sedUpper*.

The code for a FOR loop:

```
iSum:=0;
for iCount:=sedLower.Value to sedUpper.Value do
    iSum:=iSum+iCount;
```

The equivalent code of the FOR loop in a WHILE loop

```
iSum:=0;
iCount:=sedLower.Value;
While iCount<=sedUpper.Value do
begin
    iSum:=iSum+iCount;
    iCount:=iCount+1;
end;
```

If a loop is counter-driven, and if we know the number of times the loop should execute, then a FOR-loop is appropriate, however, if the loop is not controlled by a counter and you do not know beforehand how many times the loop will repeat, you should rather use a WHILE-loop.

THE DIFFERENCE BETWEEN THE WHILE-LOOP AND THE REPEAT-LOOP

To understand the difference between a repeat-loop and while-loop, compare the following two code snippets.

REPEAT-LOOP	WHILE-LOOP

`iInput := 1500;`
`repeat`
`  iInput := iInput * 2;`
`until iInput >= 1000;`
`ShowMessage(IntToStr(iInput));`
	`iInput := 1500;`
	`while iInput < 1000 do`
	`  iInput := iInput * 2;`
	
	`ShowMessage(IntToStr(iInput));`

It is an I-C-T loop Post-test loop – condition tested at the end of the loop The loop executes while the condition is false and exits the loop when the condition is false The loop executes at least once	It is an I-T-C loop Pre-test loop – condition tested at the beginning of the loop The loop executes while the condition is true and exits the loop when the condition is false The loop may not execute at all

Activity 7.18

Prime numbers are numbers that only have two factors: 1 and the number itself. Name the project **PrimeNumber_p**.

Write a program that will display the first 20 prime numbers.

Hint: In activity 7.9 you wrote an algorithm to identify the factors of a number. If a number has no factors other than itself and 1, it is a prime number.

Activity 7.19

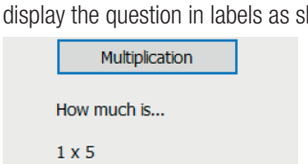
Continue working in the program you created in Activity 7.16 on page 181.

Your brother needs to practise his multiplication. Change your program as follows:

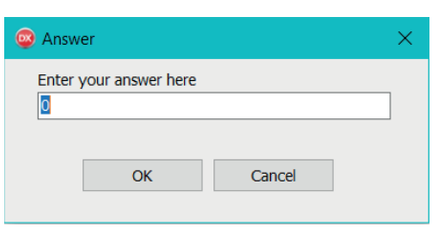
7.19.1 Add the [Multiplication] button.

7.19.2 Write code for this button to:

- **a.** randomly select two numbers between 0 to 10 to multiply.
- **b.** generate the correct answer to this sum by multiplying the two random numbers.
- **c.** display the question in labels as shown In the Image on the below.



- **d.** display an *InputBox* to allow the user to type in the correct answer as shown in the image alongside.



- **e.** compare the answer given by the user to the answer generated by the program.

7.19.3 If the answer is correct, add 1 to the score. The loop stops executing when the score is 10.

7.19.4 Count the number of tries he takes to get 10 correct answers.

7.19.5 Display the percentage correct answers.

Activity 7.20 Build a pyramid

An imaginary Pharaoh wants to build a solid pyramid. He needs to order the blocks to build the pyramid from a stone cutter who will transport the blocks.

Write an app that will calculate and display the number of blocks the Pharaoh needs to order.

Understand the problem:

- A pyramid is built up from a square base.
- This is a solid pyramid; therefore the area of the square has to be filled with blocks.
- Each level has fewer blocks than the previous level.
- The top level consists of a limited number of blocks.
- Which values do you need to get from the user?

Activity 7.20 Build a pyramid *continued*

7.20.1 Write the algorithm to calculate the number of blocks needed.

7.20.2 Create a trace table using the following data to check your algorithm:

- **a.** the base takes 10 blocks per side
- **b.** each level reduces by two blocks per side
- **c.** the top level has one block.

7.20.3 According to your trace table, how many level should your pyramid have?

7.20.4 Create a program to calculate the number of blocks needed to build the pyramid.

Display the following:

- **a.** the number of blocks needed to build the pyramid
- **b.** the number of level in the pyramid.

Activity 7.21 Healthy meals

In Chapter 4 you had to correct the output of a program to order healthy meals from the tuck shop.

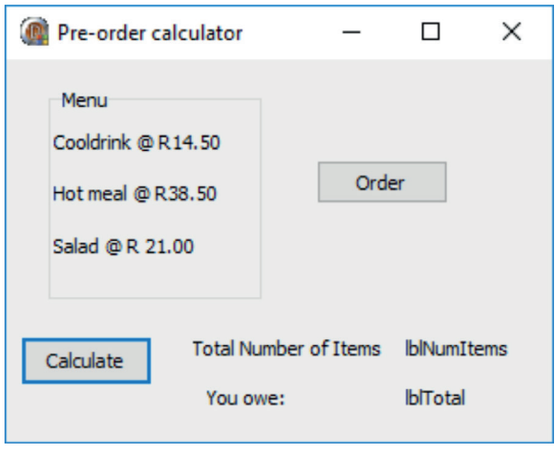
Open the **PreOrder_p** project used in Activity 4.7.3 and improve the program so that it only calculates the order once the user indicates that the order is complete.

Improve the program by following the guidelines below:

7.21.1 Change the interface so that it looks similar to the one shown alongside.

Note: The menu is placed in a groupbox.

The spin edits are removed and replaced by a button.



7.21.2 Add code to *btnOrder* to use an *InputBox* for the user to enter his order. Once an item has been ordered, increase the number of items ordered and add to the total amount owed. A message should appear to ask if the order is complete. If the answer is 'No', another *InputBox* should appear, but if the answer is 'Yes' *btnCalculate* should be enabled, and the loop stops.

Hint: Create a Boolean variable *bStop* and set the value to false.

The while do loop is then:

```
While bStop = false do
use bStop := True to stop the loop.
```

7.21.3 Change the code to *btnCalculate* to use the values saved in the global variables to calculate the total amount of items ordered as well as the total amount owed. Display these values in the labels on the form.

7.21.4 Run your program.

UNIT

7.8 Apply Loop Structures

THE GREATEST COMMON DIVISOR

So far you you have already worked with some interesting mathematical challenges. The fast iteration through loops speed up solving complex mathematical calculations. Think back to the two mathematical calculation examples dealt with:

- testing for a prime number
- calculating the Lowest Common Multiple (LCM) of two numbers.

In this unit we will look at another example that is a bit more complicated.

Example 7.13

The **Greatest Common Divisor** (GCD) is a mathematical challenge and well solved by applying the **Euclidean Algorithm**. It refers to the **largest natural number** that *divides* into two or more non-zero **integers without a remainder**. We will only apply it to two numbers. For example:

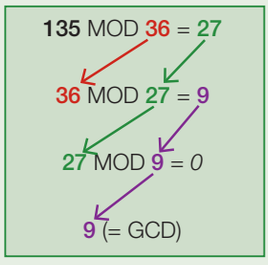
- the GCD of 8 and 12 is 4. The number 2 would also be a divisor, but it is not the *greatest* common divisor.
- the GCD of 9 and 18 is 9.
- the GCD of 24 and 18 is 6.

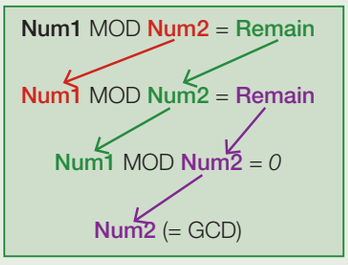
A good reverse check would be to write down multiples of your answer. The sequence of numbers being created should contain the two original numbers for the GCD-testing. Take the last example with a result of 6. The multiples of 6 are: 6, 12, **18**, **24**, 30, 36, and so on. Here you see the original GCD-testing numbers printed in bold.

So, how can we develop an algorithm and/or a small project to calculate the GCD of two numbers?

Let's begin by demonstrating one interpretation we can use for our algorithm to be developed.

Assume we have the two numbers **135** and **36**. Applying the Euclidian Algorithm we would do the following:

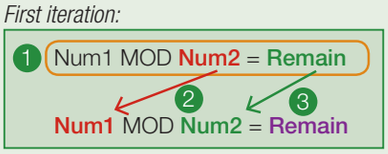




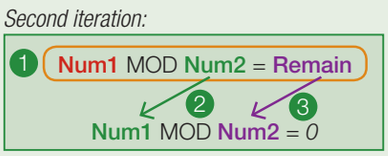
Expressing it in more general terms as algorithm:

```
Find GCD

BEGIN
    INPUT Num1
    INPUT Num2
    WHILE (Num1 MOD Num2) > 0 DO
        Remain ← Num1 MOD Num2   // 1
        Num1 ← Num2              // 2
        Num2 ← Remain            // 3
    END WHILE
    OUTPUT Num2
END
```

First iteration:



Second iteration:

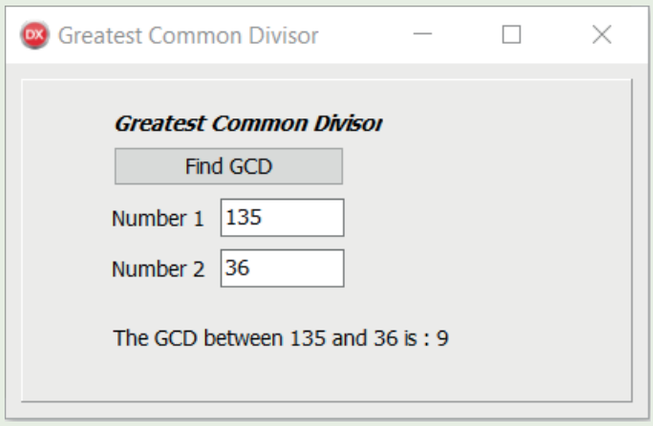


Example 7.13 *continued*

The iteration is based on the *condition* that the *remainder* of the two numbers Num1 and Num2 is **larger than zero** (that is **not** equal to zero). As part of each iteration:

- the remainder (Remain) is calculated from Num1 MOD Num2
- Num1 is overwritten by Num2 (moved to the front – compared to diagram)
- Num2 is overwritten by Remain (moved to the front – compared to diagram)

The loop terminates if the remainder is equal to zero.



UNIT

7.9 Initialising variables using the onshow event

DECLARING LOCAL AND GLOBAL VARIABLES

In Chapter 4 you learnt about the declaration and use of local and global variables. If you are unsure of this work, refer back to page 78 to refresh your memory.

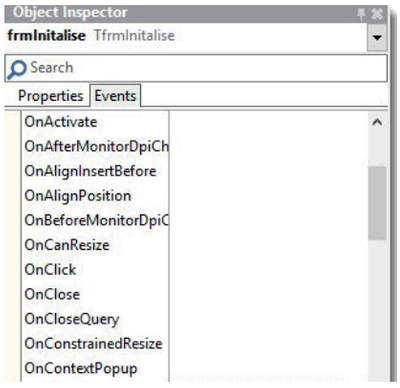
FORM EVENTS

You also learnt about the *OnCreate* event in Chapter 4. The *OnCreate* event is the first event that is activated when when the application is opened. The *OnCreate* event is only executed once. This makes it the perfect event for initialising variables. Another option to initialise variables using the events that activate when your program is opened the first time is to use the *OnShow* event.

EVENT	DESCRIPTION
OnShow	In the sequence of execution, the *OnShow* event activates after the *OnCreate* event. Beyond this, it will be activated each time the form is made visible. This is useful if you are creating an application with multiple forms.

To add create an *OnShow* event:

- select the form component in *Design View*
- in the *Object Inspector*, click on the *Events* tab
- find the *OnShow* event
- double click the space to the right of *OnShow*
- the *OnShow* event handler will display:




```
procedure TForm1.FormShow(Sender: TObject);
begin

end;
```

- For example, you can initialise the variables that you want to use.

```
procedure TForm1.FormShow (Sender: TObject);
begin
     iCounter := 0;
     sName := '';
     bIsReady := False;
end;
```

Remember!

If you want variables to be accessible from any event, they must be declared globally.

No matter which technique you use, make sure that you initialise all variables before using them in calculations!

Activity 7.22

Open the Consolidation Activity no. 17, **Numbers_p**, that you worked on in Chapter 4. This activity required you to play with numbers. The variable *iAnswer* was declared many times.

7.22.1 Improve this program by removing unnecessary repetitions and declaring some global variables.

7.22.2 Initialise all variables that will hold the results of a calculation.

UNIT

7.10 Timers

Up to now, you have been working with loops created in your code. These loops try to complete the statements inside the loop as quickly as possible. This is great if you want your program to finish as quickly as possible, but it is not good if you, for example, are using the loop to control the movement of an object in a game. When you want to control movement, you need a loop that will always run at exactly the same speed.

Delphi allows you to create a loop like this using the *TTimer* component. To add a TTimer component on a form select *TTimer* from System in the *Tool Palette* and place on the form.

Remember!

It is useful to control visual aspects of the GUI.

The timer component is a non-visual component. You can see the timer component as an icon in design view. When the program executes, the timer component is not visible.

PROPERTIES OF THE *TIMER* COMPONENT:

The Timer component has the following properties:

- **Name**: allows you to change the name of the timer component. The prefix *tmr* is used before the name of the timer.
- **Enabled**: This property can be set to either TRUE or FALSE. If the Enabled property is set to TRUE, it allows the timer to run and will trigger an event. If the Enabled property is set to FALSE, it stops the timer, that is, its *OnTimer* event handler does not trigger an event.
- **Interval**: An integer value that tells the *Timer's* event handler when to trigger in milliseconds between two events. For example, an interval of 5 000 tells the *Timer's* event handler to trigger every 5 seconds. An interval of 1 000 tells the *Timer's* event handler to trigger every 1 second and so on.

The *OnTimer* event is the only event that the *Timer* component responds to. You can then use the *OnTimer* event to create code that runs when the timer triggers. Since the code inside the *OnTimer* event will repeat every few milliseconds, this works in the same way as a loop. The only difference is that you can control how quickly the loop repeats.

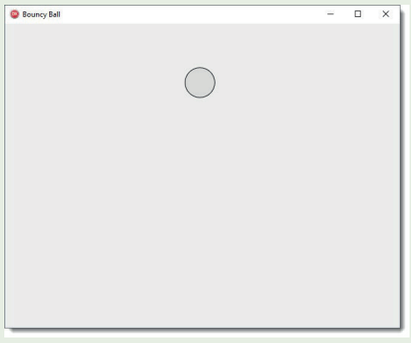
Just like with loops, the code in the timer's event will continue running until you tell it to stop. This can be done by disabling the timer component.

To see how a timer is used, work through the following examples and guided activities.

Example 7.14 Bouncy ball

For this project, you will create a simple animated game in which a ball bounces around the screen until you click on it. To do this:

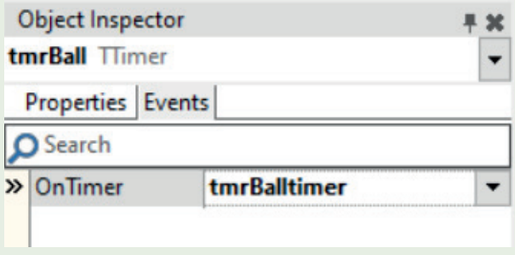
1. Create a new **BouncyBall_p** project and save it in a folder called 07 – Bouncy Ball.
2. Add a *TCircle* component from the *Shapes* list to your user interface.
3. Change the height of the form to 500 and the width to 650.
4. Change the height and width of the circle to 50. Your UI should now look as follows:



Did you know

Users cannot see your timer components during runtime, so they can be placed anywhere on the form.

5. Create two global integer variables *iVerticalDirection* and *iHorizontalDirection*.
6. Assign a value of 5 to both your global variables in the variable declaration.
7. Add a *TTimer* component from the System list to your form.
8. From the Events tab of your timer, create an *OnTimer* event.
9. Set the *Interval* property of the timer to 5. This event will now activate every 5 milliseconds.
10. Add the following line of code to your timer event.

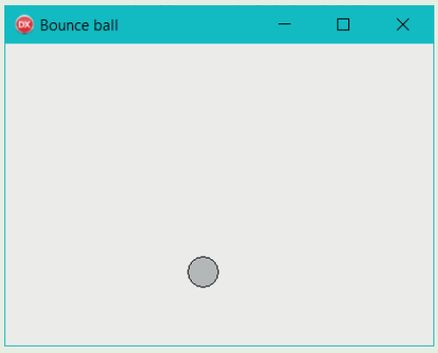


Ball drop code

```
shpBall.Top := shpBall.Top + iVerticalDirection;
```

The vertical position of the ball is determined by the shape's Top property. This property tells your shape how many pixels there should be from the top of the form to the top of the shape. By systematically increasing or decreasing this value, you can cause the shape to move down or up. Since both iVerticalDirection is equal to 1, the code above will increase the ball's Top property by 1 every 5 milliseconds. This will move the ball 1 pixel away from the top of the window every 5 milliseconds.

11. Save and test your application. The moment the program opens, you should see your ball starting to fall!



Example 7.14 Bouncy ball *continued*

12. To cause your ball to bounce, add the following code to your event.

Ball bounce condition

```
if shpBall.Top >= 450 then
  iVerticalDirection := iVerticalDirection * -1;
```

Remember, the ball has a height of 50 pixels while the form has a height of 500 pixels. This means that, when the ball's *Top* property is at 450 pixels, the bottom of the ball is touching the bottom of the form, since 450 + 50 = 500. The moment this happens, the conditional statement becomes true and the vertical direction changes from positive to negative. This, in turn, causes the ball to start moving upwards.

Unfortunately, you will now see the ball move up until it is out of the screen. To fix this:

13. Update the condition in your timer event so that it change vertical direction whenever the *Top* property is larger than 450 or smaller than 0. With this change, the ball's direction will become positive again once it reaches the top of the screen.

14. Save and test your application. The ball should now bounce up and down in your application!

Awesome! You have just animated a moving ball using a timer event. In the next guided activity, you will move the ball sideways and stop the timer when you click on the ball.

Guided activity 7.3 Bouncing ball improvements

In the previous example, you created a bouncing ball game that caused a ball to bounce up and down in window. In this guided activity, you need to complete the game by completing the following tasks:

Did you know

Changing the starting values to 10 makes the game very difficult!

7.3.1 Inside the OnTimer event, increase the ball's *Left* property by the value of *iHorizontalMovement*. This changes the number of pixels from the left of the form to the left of the shape.

7.3.2 If the ball's *Left* property is larger than 600 or smaller than 0, multiply *iHorizontalMovement* by -1. This will change the direction of the ball's horizontal movement.

7.3.3 Create a label with a large font in the middle of the form.

7.3.4 Create a global *iTime* variable and set its value equal to 0.

7.3.5 Increase the value of *iTime* by 1 in the OnTimer event and display its value in the label.

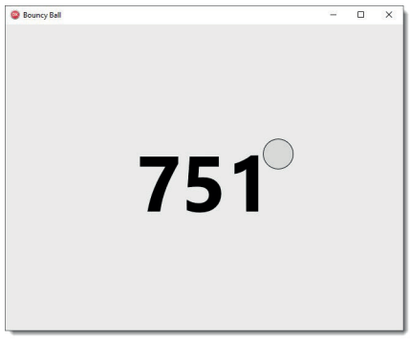
7.3.6 Create an OnClick event for the ball shape.

7.3.7 Inside the OnClick event, change the timer's *Enabled* property to False if it is True, and to True if it is False. This allows you to start and stop the ball game by clicking on the ball.

7.3.8 Inside the OnClick event, reset the value of *iTime* to 0.

7.3.9 Save and test your application.

7.3.10 You can increase the ball speed by increasing the starting values of *iHorizontalDirection* and *iVerticalDirection*.

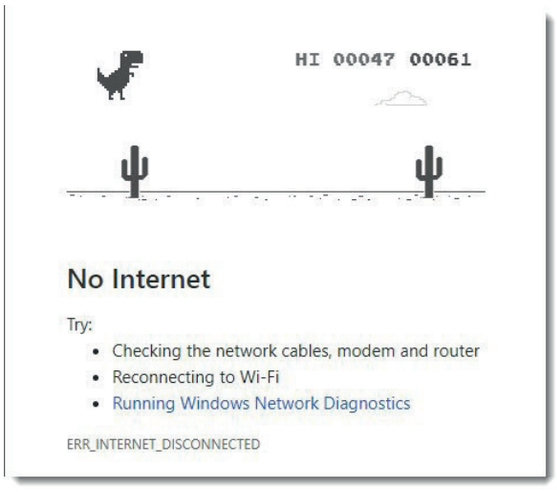


Once your improvements are done, you will have created your first, fully functional game. Well done! If you are interested in creating your own games, the next two years will teach you many techniques that will help you to create new games.

JUMPING DINO ALGORITHM

If you use Google Chrome to browse the internet, you may be familiar with the following screen that is shown when you try to access the internet without an internet connection.

What you may not know, is that this is actually a hidden game that you can activate by pressing the up arrow on your keyboard. This activates the Jumping Dinosaur game in which you are in charge of a dinosaur that needs to jump over (or duck under) oncoming obstacles.



Keeping the bouncing ball game you previously created in mind, develop an algorithm for each of the following:

1. Repeatedly moving the same cactus from the right side of the screen to the left.
2. Jumping and landing with a dinosaur.

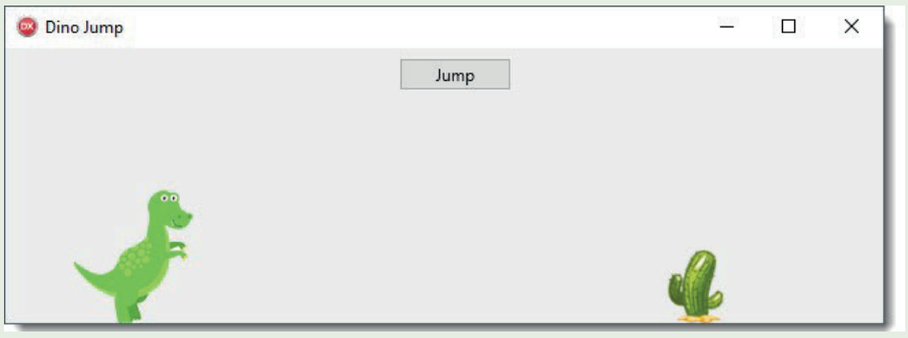
When creating the pseudocode, think carefully about the following questions:

- How will you move the cactus from the right to the left?
- How will the cactus know when it reaches the left of the screen?
- What will you do once the cactus reaches the left of the screen?
- How will you move the dinosaur up when it jumps?
- How will you stop the dinosaur from jumping too high?
- How will you move the dinosaur back to the ground?
- How will the dinosaur know when it has reached the ground?
- What will happen once the dinosaur reaches the ground?

Example 7.15 Jumping dino

To create the Jumping Dino game:

1. Open the project saved in the '07 – Jumping Dino' folder. You should see the following user interface (including a timer that is not shown):



In this user interface, the timer has an interval of 10, while the height and width of the items are given in the table below.

COMPONENT	HEIGHT	WIDTH
Form	200	640
Dinosaur	100	88
Cactus	60	46

The following variables have already been declared.

VARIABLE	VALUE	SCOPE
iJumpSpeed	0	Global
iSpeed	6	Local (OnTimer)
iDropSpeed	1	Local (OnTimer)

2. In the OnTimer event, decrease the *Left* property of the cactus by the value of iSpeed.
3. Create a conditional statement that sets the *Left* property of the cactus to 640 once it is smaller than –48. You can do this using the following code.

Cactus movement

```
imgCactus.Left := imgCactus.Left - iSpeed;
if imgCactus.Left <= -48 then
  imgCactus.Left := 640;
```

This code allows the same cactus to repeatedly appear from the right of the window, by moving your cactus to the right of the window once it disappears from the left of the window.

4. Save and test your application. You should now see the cactus moving across the screen.

Example 7.15 Jumping dino *continued*

5. In the OnTimer event, decrease the Top property of the dinosaur by the value of *iJumpSpeed*.

Dino jump code

```
imgDino.Top := imgDino.Top - iJumpSpeed;
```

Remember, *iJumpSpeed* starts with a value of 0. As long as the value is 0, this code will not change the dinosaur's position. However, as soon as *iJumpSpeed* has a positive value, this line will cause the dinosaur to move upwards. If *iJumpSpeed* has a negative value, this will cause the dinosaur to drop down.

6. In the *OnClick* event for the [Jump] button, set the value of iJumpSpeed to 14. Since this value is negative, it will cause your dinosaur to jump whenever you click on the [Jump] button.
7. In the *OnTimer* event, create a conditional statement that checks if the dinosaur is in the air. The dinosaur is in the air whenever its Top position is smaller than 100.
8. Inside the conditional statement, decrease the value of *iJumpSpeed* by *iDropSpeed*. This will cause your dinosaur to move upwards more slowly (and eventually fall downwards) each time the timer activates.
9. Using an ELSE-statement, set the dinosaur's Top position to 100 and *iJumpSpeed* to 0 when the dinosaur is not in the air. The full conditional statement is given below.

Conditional statement

```
if imgDino.Top > 100 then
  iJumpSpeed := iJumpSpeed + iDropSpeed
else
begin
  imgDino.Top := 100;
  iJumpSpeed := 0;
end;
```

In real life, whenever you jump, you start by moving upwards at a high speed. However, for every second you are in the air, you decelerate. This deceleration eventually causes you to fall back down to the ground. Once you land on the ground, you stop decelerating again. The code from the Jumping Dino game works in the same way.

Every time the timer activates, it checks if the dinosaur is in the air. If the dinosaur is in the air, it decreases the dinosaur's jumping speed by *iDropSpeed* until the dinosaur starts falling. Once the dinosaur hits the ground (that is, once the Top property is smaller than 100), the jumpspeed is reset to 0.

10. Save and test your application. The dinosaur should now jump whenever you press the [Jump] button.

Awesome! In this application you combined real-life physics with loops to create an interactive game. To do this, you had to use all the techniques you have learned so far.

If you want to continue improving your game, you could:

- Create a second timer to keep track of the score. This timer can increase the score by 100 points every second until the dinosaur fails to jump over the obstacle.
- Create a flying bird obstacle or a second cactus obstacle.
- Gradually increasing the speed of the cactus. To do this, you can increase the speed of the cactus by a few percent every time it reaches the left side of the screen.
- Randomising the timing of the cactus. This can be done by using a random number to start the cactus closer or further away from the dinosaur.
- Detecting when the cactus strikes the dinosaur. This can be done by comparing the Top and Left positions of both the cactus and the dinosaur.

All of these improvements are possible using the techniques you have learned so far!

Activity 7.23

7.23.1 Using the Dino Jump game:

a. Implement any of the two suggested improvements to the game.

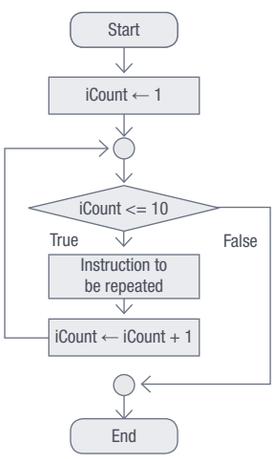
b. Create two buttons that allow you to increase or decrease the speed of the game by changing the timer's *Interval* property.

7.23.2 Create a timer application. The timer should display the number of minutes and seconds since the [timer] button was clicked.

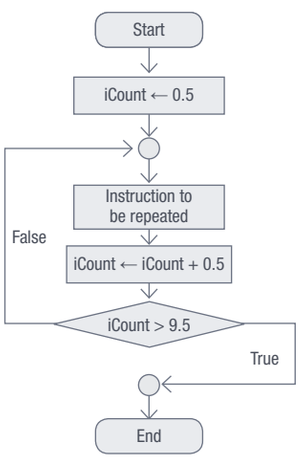
Summary

REPETITION – LOOPS

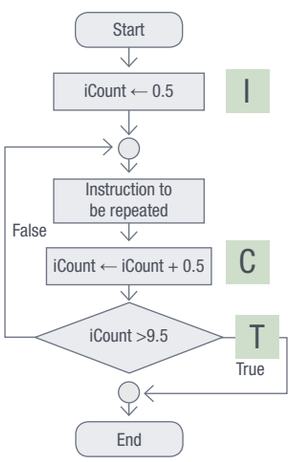
WHILE loop (*Pre*-condition **I-T-C**)



FOR Loop (loop with ***ordinal*** values **I-T-C**)



REPEAT … UNTIL loop (*Post* condition **I-C-T**)



Rules for LOOPS

- Know or calculate the start value before starting the loop
- Produce a CHANGE in the loop to impact on your condition
- Ensure that your CONDITION terminates your loop

STRUCTURES WITH …

ListBox, **C**omboBox, **R**adioGroup box components (LCR) with important properties & methods listed

PROPERTY / METHOD	LISTBOX	COMBO BOX	RADIO GROUP BOX
ItemIndex	✓	✓	✓
Columns	✓	×	✓
Items	✓	✓	✓
Sorted	✓	✓	×
TabWidth	✓	×	×
Items.Add()	✓	✓	✓
IndexOf()	✓	✓	✓
SaveToFile()	✓	✓	✓
LoadFromFile()	✓	✓	✓

Main Purpose of respective LCR-components

- ListBox mainly used for *listing tabular data*
- Combo & Radio group boxes are mainly for *selection purposes*

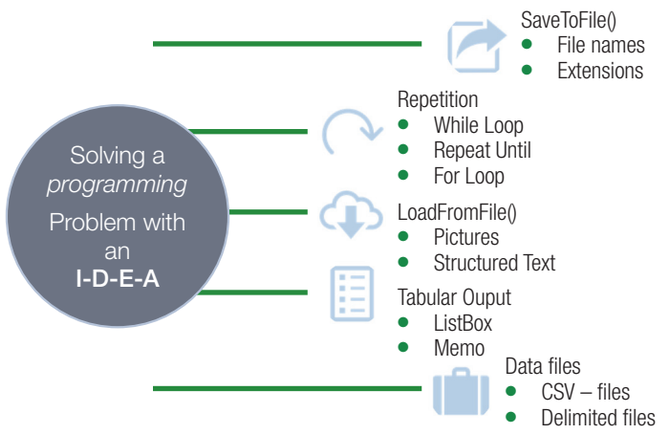
Repetition Activities do help with …

- Find *lowest* / *highest* values
- Calculate *sum* or *average* of a column of numbers
- *Search* or *Filter* based on a selected item's value

VIEW DATA IN TABLE FORMAT

Problem Solving for Developers: POLYA adapted for programmers – the **IDEA**

- **Identifying** the Problem (Understand the text. Gather known, unknown facts. Visualise facts)
- **Develop** a Plan (Design an Interface, an Algorithm, Variables, Data structures, etc)
- **Execute** the Plan (Translate your design into a real interface. Switch algorithm to code, and so on)
- **Assess** the Solution (Evaluate you running, functioning application – Evaluate critical)



In most programming languages, including Delphi, there are three types of loops that can be used:

- FOR-loop
- WHILE-loop
- REPEAT-loop

These loops generally do the same thing: They repeat several instructions until some condition is met. However, they differ in how they decide when to start and stop the loops. The table below describes the three loops and highlights the differences between the loops.

CONSTRUCT	DESCRIPTION	INITIALISATION	CHANGE	TEST
For	Repeats instructions for a given number of times. Will always execute the number of times given by the upper boundary minus the lower boundary plus 1.	The programmer does not have to initialise the 'counter' or change and test it. Instead, the counter is automatically initialised when the loop is declared.	The counter is automatically incremented by 1 with each repetition. The counter cannot be changed by the programmer.	The condition is tested at the start of the loop. The loop ends when the counter is larger than the specified upper boundary.
While	Repeats while a certain a condition is met. May not execute at all if the condition is false before the loop starts.	The programmer must initialise the loop control variable. If the variable is not initialised, it can start with unexpected values resulting in unexpected errors.	The programmer must ensure that the code manually changes the loop control variable. If this line is left out, the loop will continue running forever, causing the application to crash.	The condition is tested at the start of the loop. The loop executes while test is true. The loop ends when the condition is False.
Repeat	Repeats until a certain condition is met. It will always execute at least once.	The programmer must initialise the loop control variable. If the variable is not initialised, it can start with unexpected values resulting in unexpected errors.	The programmer must ensure that the code manually changes the loop control variable. If this line is left out, the loop will continue running forever, causing the application to crash.	Unlike the while-loop, the repeat-loop's condition is tested at the end of the loop. This means the loop will always run at least once. The loop executes while test is False. The loop is ended when the condition is True.

Answer the following in your own words.

1. Complete the table below to describe the different loop constructs and show their differences.

CONSTRUCT	DESCRIPTION	LOOP CONTINUES	LOOP ENDS
Repeat			
While			
For			

2. Give the syntax for each loop construct:

REPEAT	WHILE	FOR

3. What is the difference between loops and a timer?
4. What are user-controlled loops?
5. The following source code for a loop is given:

```
For X := A to B do
    // A single Instruction
```

Write down how many times the 'single instruction' will be executed if:

a. A ← 10 B ← 20
b. A ← 15 B ← 10
c. A ← 12 B ← 12

6. The following source code for a loop is given:

```
iCalc := 102; iNum := 200;
While iNum > iCalc do
    iNum := iNum MOD iCalc;
```

Which statements are true and correct and which are false?

a. The integer iNum ends with a value of 98.
b. The while-loop iterates endlessly.
c. The while-loop never executes.
d. iNum may not be changed inside the loop.

7. The following code demonstrates a REPEAT-UNTIL loop:

```
Repeat
 iNum := Random(51) * 2 + 50;
Until (iNum = 51);
```

Which statements are true and correct and which false?

a. The loop never executes.
b. This loop will execute at least once.
c. The loop produces even numbers.
d. Numbers between 50 and 150 will be produced.
e. This is an endless loop.

8. Go back to example 7.10. Complete the trace table if the two numbers 6 and 15 were exchanged.

LINE #	iNUM1	iNUM2	iLCM	(iLCM MOD iNUM2)	= 0 ?	COMMENT
1, 2, 3	6	15	0			

9. Write a Delphi application: Save your project as **ReverseString_p** in a folder named 07 – Reverse String. The application must accept a small sentence, like: 'Grade 10 pupil' and has to reverse the string, so that the first letter is capital again and the last letter changed to a small letter, like: 'Lipup 01 edarg'.
Hint: Use the Upcase function to convert a character to uppercase, and use the Lowercase function to convert a character to lowercase.

10. The program in the 07 – Error Data File folder is meant to help a person keep his cellphone safe. The program should do the following:
    - Generate a new UserName as follows:
      - The user enters his name in the *edtName* component.
      - When *btnNewUserName* is clicked the name is changed by replacing every vowel in the name with a random number between 0 and 9. The numbers are then added and the sum of the random numbers is added to the new UserName.
      - The name typed into the edit box and the newly generated user name are displayed in the memo component.
    - Generate a password based on the user's cellphone number as follows:
      - The user enters the cell phone number in the *edtCell* component.
      - When the btnPassword is clicked the cellphone number is converted to a password by replacing the numbers 3,6,7 and 9 with the characters %,*,@,#, respectively.
      - Display the password in the memo component.

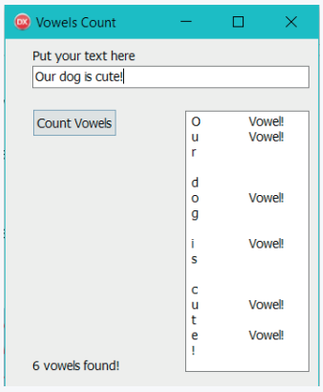
    a. This program does not work properly. Run the program with your name and cell number and try to identify the errors.

    b. Draw a trace table for each of the two procedures to identify the errors made in the program.

    c. Apply the corrections needed to fix the identified errors.

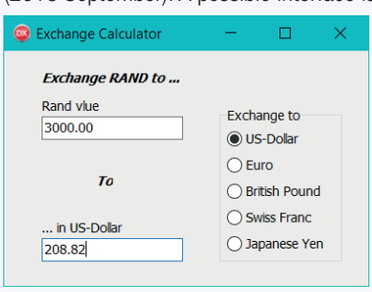
11. Write a Delphi application. Save the project as **VowelsCount_p** in a folder named 07 – Vowels Count.

    This application accepts any small sentence and counts the number of vowels as shown in the image to the right. Display the total number of vowels.



12. Write a Delphi application. Save the project as **ExchangeCalculator_p** in a folder named 07 – Currency Converter.

This Delphi application must accept any amount in Rand, which is then converted into the selected currency. A list of the top 10 currencies exchange rates has been provided (2018 September). A possible interface is given below.



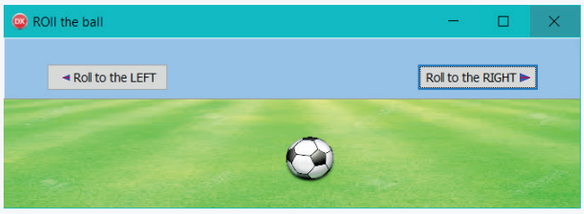
RATES TABLE

1 South African Rand Rates table

Top 10 South African Rand	1.00 ZAR	Sep 25, 2018 11:32 UTC inv. 1.00 ZAR
US Dollar	0.069606	14.366600
Euro	0.059091	16.923012
British Pound	0.052903	18.902518
Indian Rupee	5.060812	0.197597
Australian Dollar	0.096013	10.415296
Canadian Dollar	0.090246	11.080881
Singapore Dollar	0.095071	10.518411
Swiss Franc	0.067231	14.874013
Malaysian Ringgit	0.287864	3.473867
Japanese Yen	7.851692	0.127361

In September 2018 the exchange rate defined that R 3 000.00 would be converted to $ 208.82. Write a Delphi application that can convert to at least five other top currencies as illustrated.

13. a. Write a Delphi application. Save the project as **MoveBall_p** in a folder named 07 – Moving Soccer Ball



img0.bmp

Version A: Pick any single bitmap image from the five provided to you by the teacher.
Follow the example interface that has two bitmap buttons.
Each button will move the ball 20 small steps either to the right or then to the left.
Load all pictures: **lawn** and **soccer** ball in design-time.

Take note

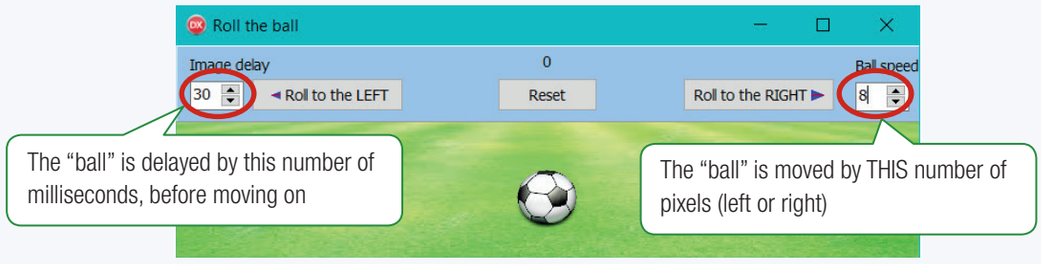
To display the ball as a round object on the 'lawn' switch on the property *Transparent* from the *TImage* object where you load the ball. You could use the Sleep() method, which delays the execution of your application in units of milliseconds, for example: Sleep(500) will pause your application for half a second before it goes on.

b. Write a Delphi application. Save the project as **RollBall_p** in a folder named 07 – Rolling Soccer Ball

Five bitmap images will given to you by the teacher. As you move from one image to the next you can observe that the ball 'turned' a few degrees. Quickly loading the balls one after the other will give you the illusion of a rolling ball.

img0.bmp img1.bmp img2.bmp img3.bmp img4.bmp img5.bmp

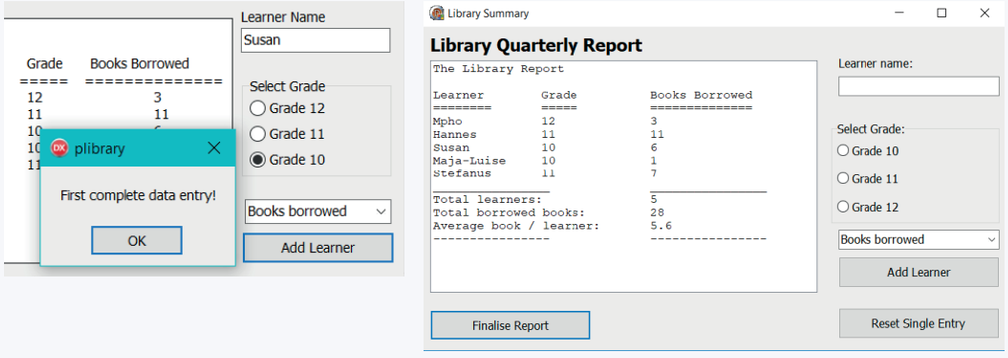
The B-version is more sophisticated than the A-version. We add another two *Spin-edit* boxes, a [Reset] button in the centre, and a small label on top of the [Reset] button. The small label indicates the number of the soccer ball image that is currently displayed. The range for the 'Image delay' is 30 to 300. The range for the 'Ball speed' is 2 to 8. [Reset] brings the ball back to the original position with the first image '0'.



Take note

Each time the ball moves to its next position it also changes to the follow-up image or previous image (depending whether we move right or left), to keep up the rolling illusion! This time images are loaded during runtime.

14. Write the following Delphi application. Save the project as **Library_p** in a folder named 07 – Library.



a. Design an interface like the one displayed. Make sure that component names are descriptive. The application gathers information of borrowed books from Grade 10 to 12 learners.

b. Validate data input for all three items: Learner name, grade selected, and number of books borrowed. If any input is missing, inform the user with: 'First complete data entry!' and do not add any incomplete data.

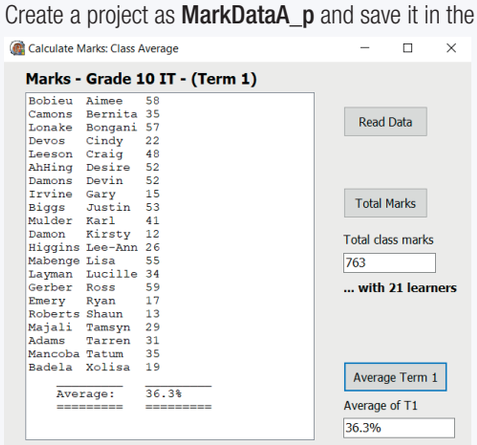
c. The [Reset Single Entry] button clears the 'Learner Name' field, unselects any grade and resets the combo box for a new selection.

d. The [Finalise Report] button adds the summary at the bottom, for example, Total Learners, and so on.

The next three exercises are based on the same tab delimited file: **Grd10Term1.txt.** Each row represents a learner's Surname, First name and mark, respectively.

Bobieu	Aimee	58
Camons	Bernita	35
Lonake	Bongani	57
Devos	Cindy	22
Leeson	Craig	48
AgHing	Desire	52
Damons	Devin	52
Irvine	Gary	15
Biggs	Justin	53
Mulder	karl	41
Damons	Kirsty	12
Higgins	Lee-Ann	26
Mebenge	Lisa	55
Laymen	Lucille	34
Gerber	Ross	59
Emery	Ryan	17
Roberts	Shaun	13
Majali	Tamsyn	29
Adams	Tarren	31
Mancoba	Tatum	35
Badela	Xolisa	19

15. Create a project as **MarkDataA_p** and save it in the folder named 07 – Mark Data A.



a. Read the text file data into the *ListBox* when the [Read Data] button is clicked.

b. Display the total of all the marks when the [Total Marks] button is clicked.

c. Display the total number of learners in the list as illustrated.

d. Calculate and display the term 1 average mark correct to one decimal place.

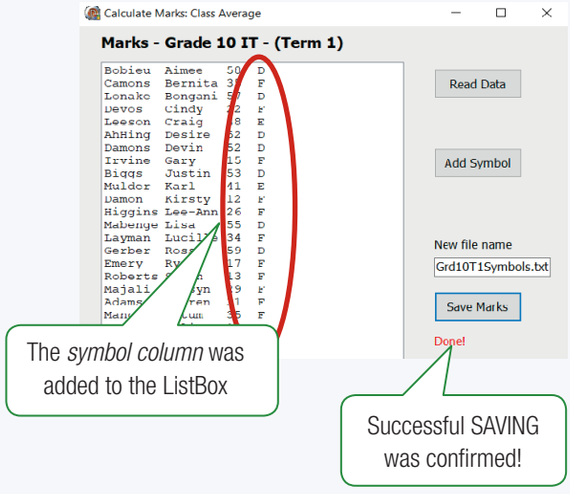
e. Add the result of the average to the bottom of the list as illustrated.

16. Write the following Delphi application. Save the project as **MarkDataB_p** in a folder named 07 – Mark Data B.

This application reads the same file as used in question 15. BUT this time the data file is saved inside a subfolder named **Data**.

When the [Add Symbol] button is clicked, the symbol achieved by each learner must be determined and displayed as illustrated. Symbols are allocated as follows:

80–100% –> A
70–79 % –> B
60–69 % –> C
50–59 % –> D
40–49 % –> E
Otherwise –> F



The [Save Marks] button will:

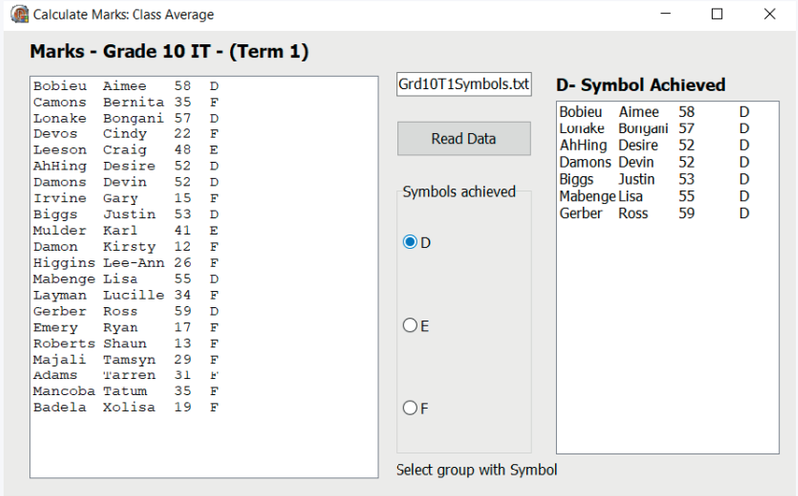
a. save all *ListBox* data under a new file name in subfolder Data.

b. save the contents of the list box to a new file called Grd10T1Symbols.txt in the folder named 07 – Mark Data C.

17. Create the project as **MarkDataC_p** in a folder named 07 – Mark Data C.

This application will read the file produced and saved in the previous question.

a. When the [Read Data] button is clicked, the data must be loaded into the *ListBox*. At the same time, unique symbols for marks must be identified and listed inside a RadioGroup box. In our example only D, E and F symbols were identified. (The sequence of symbols (like from A to C) does not matter at this point in time).



b. Next the user must be able to click on one of the symbols listed. An OnClick-event for the RadioGroup box must be programmed to extract all the learners that achieved the selected symbol. These learners are displayed on the *ListBox* on the right. The header of that ListBox must change accordingly.

c. Save and close the project.

TERM 3

CHAPTER 8

STRING MANIPULATION

CHAPTER UNITS	
Unit 8.1	Combining strings and determining the length of a string
Unit 8.2	Formatting characters
Unit 8.3	Scrolling through a string
Unit 8.4	Manipulating strings

Learning outcomes

At the end of this chapter, you should be able to:

- explain the concept of string manipulation
- scroll through a string
- search for a character in a string.
- change a string
- build a string.

INTRODUCTION

Often data is encoded into a string. Strings are variables that are made up of a sequence of letters, numbers and symbols. One example of a string is your South African identity number. Your date of birth and gender are just two of the data items encoded into your ID number.

Strings can be manipulated and changed in a wide variety of ways in Delphi. We need to be able to decode strings and separate data items – this is an essential coding skill.

In this chapter you will learn how multiple small strings can be combined into a single large string, or one string can be broken into multiple smaller strings. You will learn how to find the position of one character inside a string and replace all the characters of one type with another. You will also learn how to delete text from strings and insert text into strings.

UNIT

8.1 Combining strings and determining the length of a string

In every chapter so far, you have been using strings to display output with appropriate messages. For example:

```
memDisplay.Lines.Add('The sum is: '+IntToStr(iSum));
```

In this example:

- the *MemoBox* component displays strings
- two strings are displayed: 'The sum is: ' and IntToStr(iSum)

COMBINING STRINGS

To combine strings, we use the following syntax:

Combining strings syntax

```
sNew := String1 + String2 + … + String1000;
```

This syntax is straightforward – you simply add all the strings together using the + operator:

In the example above, we could also have combined the two strings before displaying them. For example:

Combining strings syntax

```
    sOutput := 'The sum is: '+IntToStr(iSum);
    memDisplay.Lines.Add(sOutput);
```

You can combine any number of strings. However, doing so needs careful consideration. You need to consider things such as spacing between strings, whether strings should be displayed on different lines or in columns. For example, imagine that you have a list of first names and surnames and you want to use to create a new list in which the name and surname are combined. You might write the following code:

Combining names and surnames

```
sFirstName1 := 'peter';
sFirstName2 := 'joe';
sSurname1 := 'smith';
sSurname2 := 'ali';

sName1 := sFirstName1 + sSurname1;
sName2 := sFirstName2 + sSurname2;
ShowMessage(sName1); // displays petersmith
ShowMessage(sName2); // displays joeali
```

However, when you display string sName1 and sName2, you realise that the output is displayed without a space between the name and surname!

Take note

A space is indicated as follows:	An empty string is indicated as follows:
• Type a single quote ' • Press the space bar once • Type another single quote ' • ' '	• Type a single quote ' followed immediately by another single quote' • ''

Rather than simply combining the name and surname variables, you need to add a space between the two strings. Here is the correct code that you should have added:

Combining strings correctly

```
sName1 := sFirstName1 + ' ' + sSurname1; // peter smith
sName2 := sFirstName2 + ' ' + sSurname2; // joe ali
```

This problem is very common since most strings are not saved with a space at the end. So, before you combine any strings, ask yourself if your combined result will have the correct spacing before running the application.

DISPLAYING OUTPUT THAT HAS AN APOSTROPHE

What about output that has an apostrophe in it? Look at the examples below to help you understand this:

Take note

To display an apostrophe in a string: place two single quotes '' immediately one after another

Example 8.1

If you want to display the string 'God's Window, Mpumalanga', then the code will be:

```
showMessage('God''s Window, Mpumalanga')
```

Example 8.2

You have the following code:

```
sName := 'John';
iMark := 78
```

To display the learner's name and mark in the format: John's mark is 78

```
sMessage := sName + '''s' + mark is 'IntToStr(iMark);
ShowMessage(sMessage)
```

DETERMINING THE LENGTH OF A STRING

Remember!

The *Length* function was introduced in Chapter 5.

Remember that a string is a list of characters. For example, the following strings:

- 'John' has 4 characters
- 'John Smith' has 10 characters – space is also a character
- 'jsmith@gmail.com' has 16 characters
- '032 551 4241' has 12 characters

You can count the numbers of characters in a string using the *Length* function. It will return the number of characters in a string, for example:

```
ilen := length('Delphi');
ShowMessage('The length of your name is:
'+IntToStr(length(sName)));
```

Example 8.3 Best birthday card

After struggling to write a nice birthday message on a card for a friend, you decide to create a program that will automatically generate birthday cards with beautiful messages for your friends. With some online research, you identify five birthday messages that you can place on your cards:

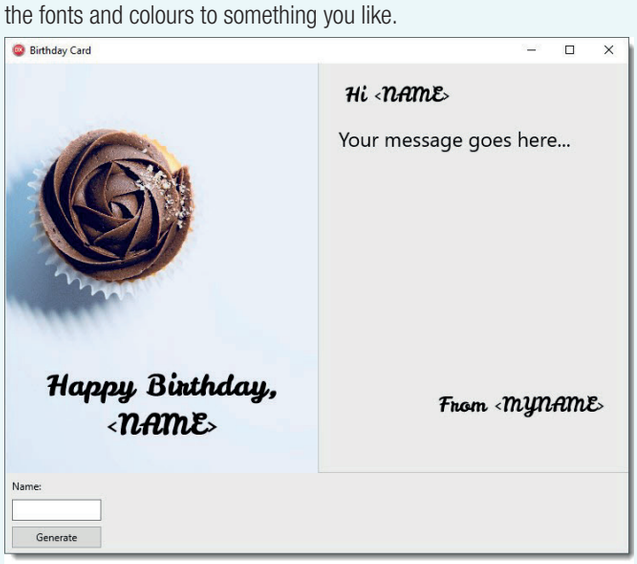
- 'Count your life by smiles, not tears. Count your age by friends, not years. Happy birthday, <NAME>!'
- 'May your Facebook wall be filled with messages from people you never talk to. Happy birthday, <NAME>!'
- 'You are only young once, but you can be immature for a lifetime. Happy birthday, <NAME>!'
- 'From good friends and true, from old friends and new, may good luck go with you and happiness too! Happy birthday <NAME>!'
- 'I hope all your birthday wishes and dreams come true. Happy birthday, <NAME>!'

Take note

<Name> refers to the name of a friend that you are sending a message to.

The program must prompt for your friend's name and randomly select one of the five messages to generate a birthday card.

1. Open the **BestBirthdayCard_p** project from the 08 – Best Birthday Card folder. If you wish to, you can change the fonts and colours to something you like.



2. Create an *OnClick* event for the [Generate] button to:
   - declare three local string variables: *sMyName*, *sFriendName* and *sMessage*
   - declare a local integer variable: *iRandom*
   - set the value of sMyName to your own name
   - prompt the user to enter a friend's name in the text box and assign the value to sFriendName
   - generate a random integer in the range 1 to 5 and assign it to *iRandom*. Remember to use the Randomize statement before you generate a random number
   - using a CASE statement, set the value of sMessage to one of the five birthday messages, based on the randomly generated number
   - generate the birthday card.

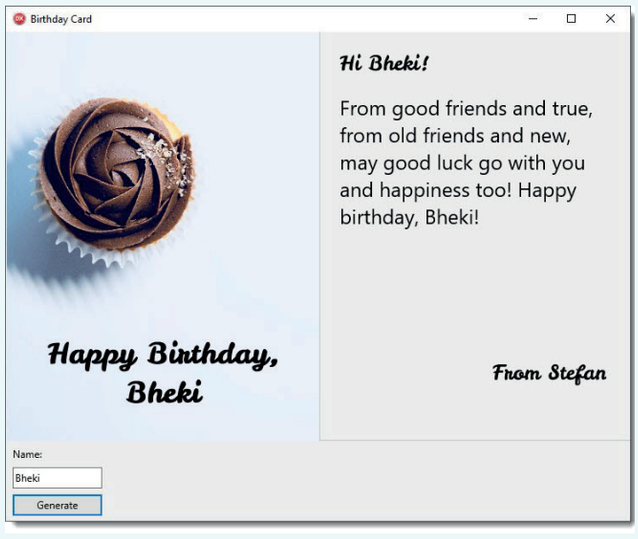
Example 8.3 Best birthday card *continued*

Here is the code for the event handler:

The code for the event handler:

```
procedure TfrmBirthday.btnGenerateClick(Sender: TObject);
var
       sMyName, sFriendName, sMessage : String;
       iRandom : Integer;
begin
      sMyName := 'Stefan';
      sFriendName := edtName.Text;
      Randomize;
       iRandom := Random(5) + 1;

     case iRandom of:
       1 : sMessage := 'Count your life by smiles, not tears. Count your
     age by friends, not years. Happy birthday,';
       2 : sMessage := 'May your Facebook wall be filled with messages from
     people you never talk to. Happy birthday,';
       3 : sMessage := 'You are only young once, but you can be immature
     for a lifetime. Happy birthday,';
       4 : sMessage := 'From good friends and true, from old friends and
     new, may good luck go with you and happiness too! Happy birthday,';
       5 : sMessage := 'I hope all your birthday wishes and dreams come
     true. Happy birthday,';
       end;
       lblCover.Text := 'Happy Birthday, ' + sFriendName;
       lblGreeting.Text := 'Hi ' + sFriendName + '!';
       lblSalutation.Text := 'From ' + sMyName;
       lblBody.Text := sMessage + ' ' + sFriendName + '!';

end;
```

3. Save and run your application. You can click the [Generate] button multiple times to create different cards.



Guided Activity 8.1 Advanced calculator: Operator buttons

To create the *OnClick* procedure for the mathematical operator buttons, you first need to consider the workflow of a normal calculator: With a normal calculator, you start by entering a number. Once the number has been entered, you click on one of the mathematical operators. This stores the number that you have entered in memory and allows you to enter a new number. When you are done entering the second number, you can press the equals sign to calculate a solution.

To recreate this workflow in our application:

8.1.1 Open the **AdvancedCalculator_p** project from the 08 – Advanced Calculator folder.

8.1.2 Create two global variables called *rStoredNumber* and *sOperator*.

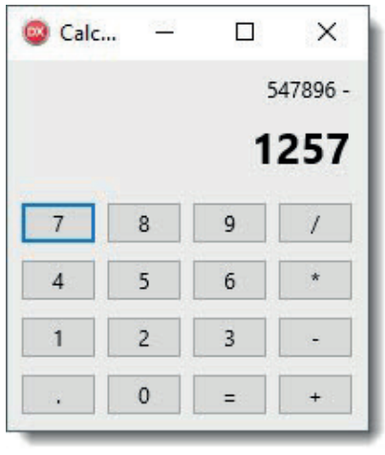
8.1.3 Create an *OnClick* event for the plus(+) button and add the following code:

Plus button OnClick event

```
rStoredNumber := StrToFloat(lblValue.Caption);
sOperator := '+';
lblValue.Caption := '';
lblCalculation.Caption := FloatToStr(rStoredNumber) + ' ' +
sOperator;
```

The first line will convert the string from the *lblValue* label to a real number and store it in the global variable *rStoredNumber*. Next, the code will store the type of mathematical operator being used in a global variable. The *lblValue* label is set to an empty string.Finally, it will show the calculation being completed in the *Calculation* label at the top of the calculator.

8.1.4 Create similar *OnClick* events for the other mathematical operators, only changing the value stored in the *sOperator* variable.



8.1.5 *Save and run your project.*

It is starting to look good! The only thing left to do is create an equals button event. This will be done in the next activity.

Guided Activity 8.2 Advanced calculator: Equals to (=) button

The final step of your calculator program is to add, subtract, multiply or divide the value stored in the *rStoredNumber* variable with the value shown on the *lblValue* label.

To do this:

8.2.1 Open the **AdvancedCalculator_p** project.

8.2.2 Create an *OnClick* event for the equals to (=) button:

- Create two local variables *rCurrentNumber* and *rTotal*.
- Store the value in *lblValue* in *rCurrentNumber*.

```
rCurrentNumber := FloatToStr(lblValue.Caption);
```

- Add the contents of *rStoredNumber*, *sOperator* and *rCurrentNumber* and store the value in *lblCalculation*:

```
lblCalculation.Caption := FloatToStr(rStoredNumber) + ' ' +
sOperator + ' ' + FloatToStr(rCurrentNumber);
```

This will store the current value in a variable *rCurrentNumber* and update the calculation string. To ensure the correct operator is used, you will need to use conditional statements.

8.2.3 Create conditional statements that calculate the value of *rTotal* based on the operator stored in *sOperator*. For example, you could create the following IF-THEN-statement for the plus operator:

if-then statement

```
if sOperator = '+' then
  rTotal := rStoredNumber + rCurrentNumber;
```

8.2.4 Convert *rTotal* to a string and assign it to the *lblValue.Caption* property. Once done, your procedure should look as follows.

Full procedure code

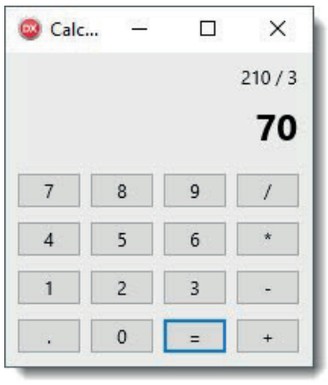
```
procedure TfrmCalculator.btnEqualClick(Sender: TObject);
var
  rCurrentNumber, rTotal : Real;

BEGIN
  rCurrentNumber := StrToFloat(lblValue.Text);
  lblCalculation.Caption := FloatToStr(rStoredNumber) + ' ' +
sOperator + ' ' + FloatToStr(rCurrentNumber);

  if sOperator = '+' then
    rTotal := rStoredNumber + rCurrentNumber;

  if sOperator = '-' then
    rTotal := rStoredNumber - rCurrentNumber;

  if sOperator = '*' then
    rTotal := rStoredNumber * rCurrentNumber;

  if sOperator = '/' then
    rTotal := rStoredNumber / rCurrentNumber;

  lblValue.Caption := FloatToStr(rTotal);
END;
```

Guided Activity 8.2 Advanced calculator: Equals to (=) button *continued*

8.2.5 Instead of four separate IF-THEN statements you can use a CASE-statement:

```
case sOperator of
    '+' :   rTotal := rStoredNumber + rCurrentNumber;
    '-' :   rTotal := rStoredNumber - rCurrentNumber;
    '*' :   rTotal := rStoredNumber * rCurrentNumber;
    '/' :   rTotal := rStoredNumber / rCurrentNumber;
end;
```

8.2.6 Save your program and test it. You should now be able to use it like a normal calculator.

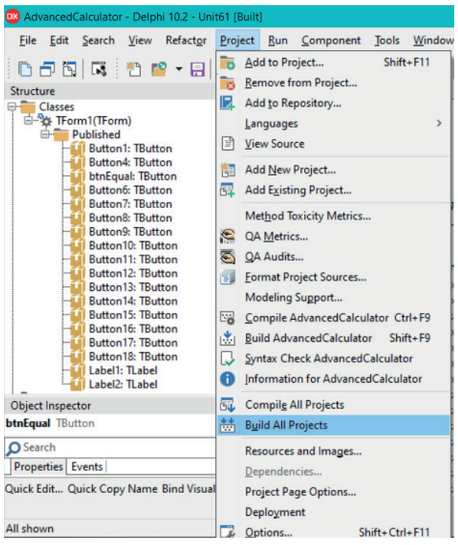


Congratulations on creating a real calculator! For this program, you needed to create a large number of events, local and global variables of different types, as well as combine strings and use conditional statements to complete calculations. By combining everything you have learned so far, you were able to create a fully functional program!

For Enrichment Creating an executable file for your advanced calculator project

To celebrate completing your calculator application, you can convert it to an executable file which can be used on any computer. To do this:

1. Open the *Project* menu at the top of RAD Studio.
2. From the menu, select Project and click on the *Build All Projects* button.



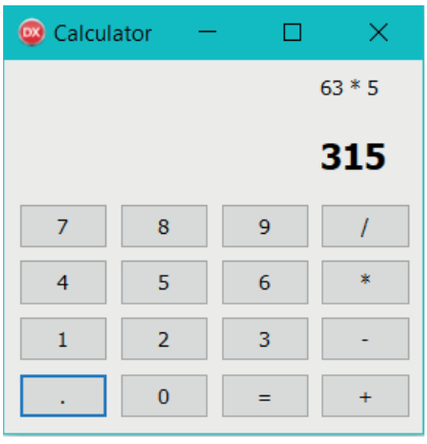
For Enrichment Creating an executable file for your advanced calculator project *continued*

3. Once the build is complete, click on the [OK] button.
4. Open the project's folder in Microsoft Explorer.
5. You should see an application file with your project's name.

Application File created which can be run on any computer

AdvancedCalculator.dpr	5/23/2019 10:48 PM	Delphi Project File	1 KB
AdvancedCalculator.dproj	5/23/2019 10:48 PM	Delphi Project File	29 KB
AdvancedCalculator.dproj.local	5/23/2019 10:49 PM	LOCAL File	17 KB
AdvancedCalculator.exe	5/23/2019 10:43 PM	Application	10,897 KB
AdvancedCalculator.identcache	5/21/2019 1:48 AM	IDENTCACHE File	1 KB
AdvancedCalculator.res	5/23/2019 10:49 PM	Compiled Resourc...	59 KB

6. Once the build is complete, click on the [OK] button.
7. Open the project's folder in Microsoft Explorer.

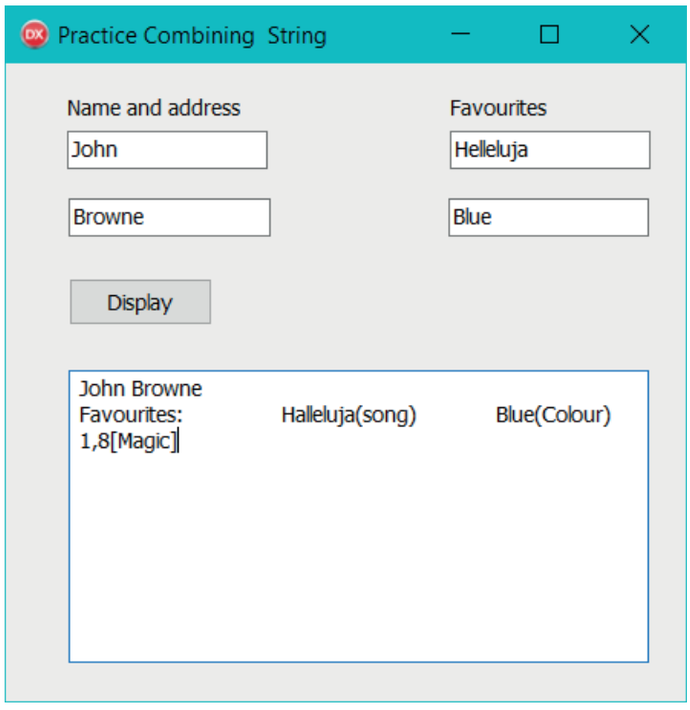


8. You should see an application file with your project's name.
9. Run this application to open your calculator.
10. You can now copy this file onto a flash disk and use it on any computer!

Activity 8.1

Open the **Example_p** project from the 08 – Practise Strings folder.

Study the example shown below.



8.1.1 Create a [Magic Number] button by dividing a random number between 50 and 100 by your age and multiplying the result with 0.93.

8.1.2 Display information from various inputs as a single string.

8.1.3 Complete the *OnClick* event for the [Display] button by following the guidelines below:

Create a single string that displays the name and surname on the first line. On the next line, display the word 'Favourites' followed by a colon, and the name of the song, with the word 'Song' in brackets. Then display the colour with the word 'Colour' in brackets. On the last line, display the magic number and the word '*Magic*' in square brackets.

UNIT

8.2 Formatting strings

You have already learnt how to format output using the *FloatToStrF* function. In this unit you will use a **control string** to format strings. A control string is a sequence of one or more control characters, each of which consists of the # symbol followed by an integer constant. The integer constant denotes the corresponding ASCII character.

For now we will use three formatting characters. That is: #9 (tab character), #10 (line Feed) and #13 (Carriage Return)

One way to make sure that your line format and spacing is correct, is to use the *Return character* and the *Tab character*. Look at the examples in the table below.

Table 8.1: *Using characters to format strings*

CHARACTER	FORMAT CHARACTER	DESCRIPTION	EXAMPLE
Tab Character <tab>	#9	• The ASCII number 9 represents a <tab> character. • #9 adds tab spaces between strings. This is especially useful when the string following #9 must be aligned to a specific position in the component.	lblOutput.Caption:='First Line'+#9+'Second line'; Display on the lblOutput label First Line Second Line
Carriage Return Character <CR>	#13	• The ASCII number 13 represents a <Carriage return> or <enter> • #13 adds a line break to the string so that the string following #13 appears on a new line.	lblOutput.Caption:='First Line'+#13+'Second line'; Display on the lblOutput label First Line Second Line
Line Feed <LF>	#10	• The ASCII number 10 represents a <Line Feed> • It moves the cursor to the beginning of the next line	lblDisplay.Caption:='First Line'+#10+'Second Line'; First Line Second Line

Notes:

- You can use the #9 and #13 formatting characters to format strings.
- #9, #10 and #13 do NOT appear in quotes
- To combine strings with formatting characters, you use the + operator
- Sometimes, it will be necessary to use more than one tab character to align strings of different lengths. To do this, you simply add the #9 character multiple times. For example:

```
sDisplay := 'Mark1'+#9#9+'71%';
memDisplay.Lines.Add('First Line'+#9#9#9+'Second Line');
```

- The only way to know if you have added the correct number of tab characters is to test the application and see the result. It should immediately see if you have added the incorrect number of tab characters.

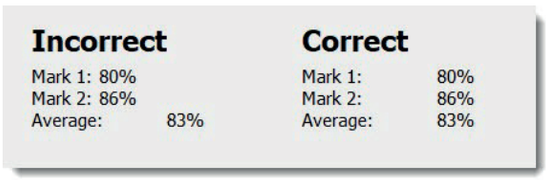


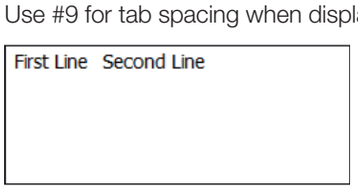
Figure 8.1: *Incorrect and correct number of tab character*

- Fonts like "Courier New" and "Consolas" are called **monospaced** fonts. This means that each character has the same width. This includes characters like "i" and "l" (which are usually very narrow) and characters like "m" and "w" (which are usually very wide). Monospaced fonts are useful if it is important that your characters are lined-up.
- As with #9, if you need many carriage returns or line feeders, you add the format characters as required.

DISPLAYING STRINGS WITH FORMATTING CHARACTERS ON A *MEMOBOX*

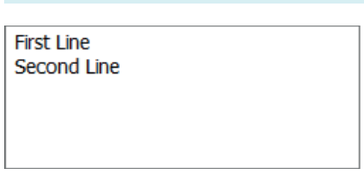
You can also display strings with formatting characters on a *MemoBox* as follows:

- Use #9 for tab spacing when displaying strings: memDisplay.Lines.Add('First Line+#9+'Second Line').



- Use #13#10 combination to leave a line in a memo box NOT just #13 as is used in labels:

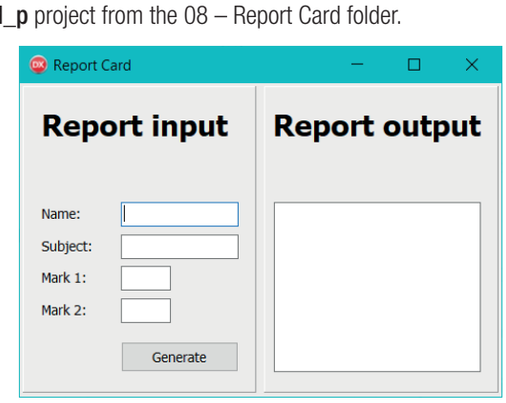
```
memDisplay('First Line'+#13#10+'Second Line');
```



Guided Activity 8.3 Report Card

In this Guided activity, we will create a report card.

8.3.1 Open the **ReportCard_p** project from the 08 – Report Card folder.



Guided Activity 8.3 Report Card *continued*

8.3.2 Create an *OnClick* event for the [Generate] button as follows:

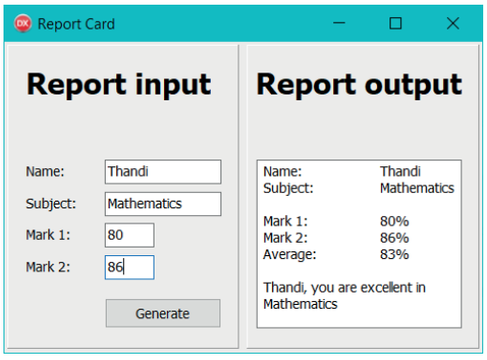
- Read the name, subject, mark1 and mark2 from the corresponding edit boxes. You may assume that the marks will be out of 100.
- Calculate the average of the two marks.
- Determine the message that will be displayed in the string variable *sMessage*. The message is based on the rounded integer average mark as follows

ROUNDED INTEGER AVERAGE MARK	MESSAGE
80 to 100	excellent
60 to 79	strong
40 to 59	improving
0 to 39	struggling

8.3.3 Display the output on the memo box component as follows:

```
Name:                Thandi
Subject:             Mathematics

Mark 1:              80%
Mark 2:              86%
Average:             83%
Thandi, you are excellent in Mathematics
```

8.3.4 Save and run the project.

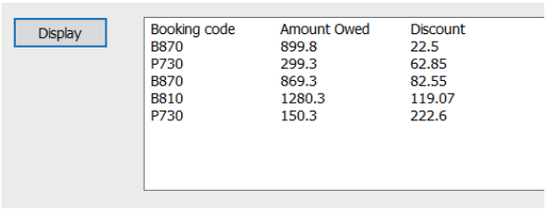


Activity 8.2

Open the **Booking_p** project from the 08 – Booking folder. Three text files: BookingCode.txt; AmountOwed.txt and Discount.txt are provided.

Complete this app as follows:

8.2.1 Complete the code for *btnLoad* to load the textfiles into the *ComboBox* components by using the *LoadFromFile* method of the *ComboBoxes*.



8.2.2 Complete the code for the [Display] button by building one string to display the values selected on each of the ComboBox components in columns as shown in the screenshot alongside.

UNIT

8.3 Scrolling through a string

In Chapter 3 you learnt that a string is a sequence or list of characters stored in a single variable. Here are some examples of strings:

- 'John',
- '54 Bird Street'
- '123'

Remember!

A character can be a letter, digit, space or any special character such as $, #, ;, @ and so on. Refer to Appendix B for the ASCII table

DECLARATION OF STRING VARIABLES

To declare a string variable:

```
Var sName:string;
    sSurname:String[15];
```

Notes

- the variable *sName* can hold up to 255 characters
- the variable *sSurname* can hold up to 15 characters. If you assign a string with more than 15 characters to *sSurname*, then only the first 15 characters of the string will be assigned.

Each character in a string has a position with the first character of the string starting at position 1. For example, in the string *sString*:= 'I LOVE CODING'

CHARACTER	I		L	O	V	E		C	O	D	I	N	G
POSITION	1	2	3	4	5	6	7	8	9	10	11	12	13

- the first character starts from position 1
- space/s have position/s as well. See positions 2 and 7
- the length of the string is 13
- the last character 'G' is stored in position 13.

An empty string, denoted by '', has 0 characters.
You can access each character in a string by using its position. To access a character in a string use the syntax:

```
    variableName [position]
```

- the string *variableName* followed by the position of the character in square brackets[]
- for example, sString[5] will refer to character 'V' and sString[10] will refer to character 'D' in the table above.

Example 8.4

Given: sName := 'Thandi';

sName[1] will have the value 'T'

sName[2] will have the value 'h'

…

sName[6] will have the value 'i'

sName[n] is also a variable of type **Char** and represents a single character. The following code snippets show how this can be done.

Table 8.2: *Code snippets for representing single characters*

READING SYNTAX	WRITING SYNTAX	EXAMPLES

Reading characters
`cChar := sString[iCharNumber];`
	Writing characters
	`sString[iCharNumber] := cChar;`
		Accessing characters in a string
		`sValue := 'Hello, World!';`
		`cFirst := sValue[1]; // H`
		`cSecond := sValue[2]; // e`
		`sValue[13] := '?'; //`

• The character stored in position iCharNumber of sString is stored in char variable cChar	• The value stored in char variable cChar is stored in position iCharNumber in string sString	• cFirst is assigned the value 'H' • cSecond is assigned the value 'e' • The character in position 13 in string sValue is overwritten with a '?'. sValue now has the string value 'Hello, World?'

The position variable also known as the index can be used as a loop variable in a FOR-loop. This will allow you to scroll through the string as follows:

```
for index ← 1 to Length(sName) do    //index used as loop variable
begin
  Output sName[index]                //index used as position variable
  Newline
    endfor
```

Guided Activity 8.4

Display each character of *sName* on a new line in a memo component *memOutput*. Use the Courier New font.
sName:='THANDI';

The Output must be:

T

H

A

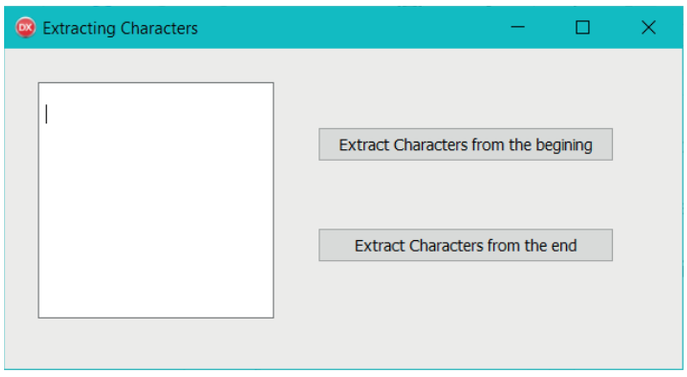
N

D

I

This program must display one character at a time from string 'Thandi' one below the other starting from the first character.

8.4.1 Open the **ExtractingCharacters_p** project from the 08 – Extracting Characters folder.



Guided Activity 8.4 *continued*

8.4.2 Create an *OnClick* event for the [Extract Character from the beginning] button and do the following:

a. Declare two local variables *sName* and *iIndex*. *sName* will hold the name 'THANDI', and *iIndex* will hold the position of a character in the string *sName.*

b. Clear the *memDisplay* memo box.

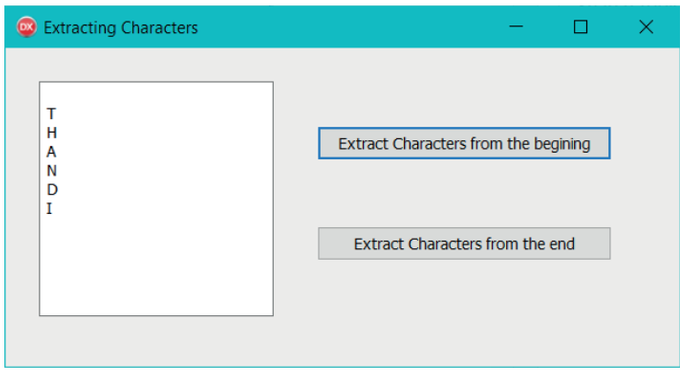
c. Assign 'THANDI' to *sName*.

d. Loop from first character position 1 to the position of the last character (length(sName)).

e. Display the character at each position in *sName* in the *memDisplay*

```
procedure TfrmExtractingCharacters.
btnExtractFroBeginClick(Sender: TObject);
var
    sName: String;
    iIndex: Integer;
begin
    sName := 'THANDI';
    memDisplay.Lines.Clear;
    for iIndex := 1 to Length(sName) do
    begin
      memDisplay.Lines.Add(sName[iIndex]);
    end;
end;
```

8.4.3 Save and run the project



8.4.4 Create an *OnClick* event for the [Extract Characters from the End] button to extract the characters from *sName* and display the output is *memDisplay* memo component as follows:

I

D

N

A

H

T

a. Declare two local variables *sName* and *iIndex*. *sName* will hold the name 'THANDI' and *iIndex* will hold the position of a character in the string *sName*.

b. Assign 'THANDI' to *sName*.

c. Clear the *memDisplay* memo box.

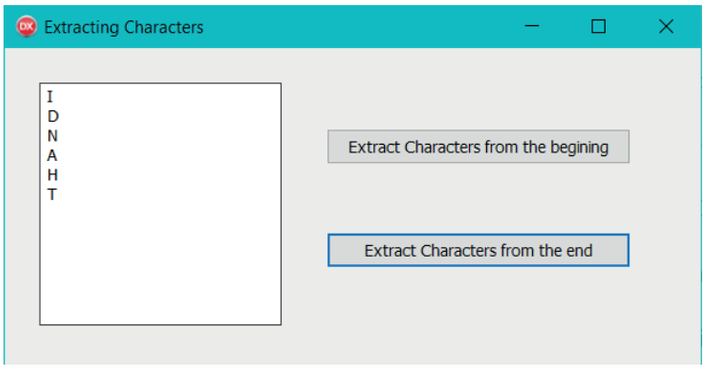
d. Loop from last character position (length(sName) down to the first character position 1.

Guided Activity 8.4 *continued*

e. Display the character at each position in *sName* in the *memDisplay*.

```
procedure TfrmExtractingCharacters.
btnExtractFromEndClick(Sender: TObject);
var sName:string;
    iIndex:Integer;
begin
    memDisplay.Lines.Clear;
    sName := 'THANDI';
    for iIndex := Length(sName)downto 1 do
       memDisplay.Lines.Add(sName[iIndex]);
end;
```

8.4.5 Save and run the program and click on the [Extract Characters from the End] button.



Being able to access an individual character in a string allows you to use a character from a string in different ways or to manipulate the string by replacing, deleting or inserting a character. You can:

- create conditional statements based on specific characters.
- copy certain characters.
- compare specific characters in different strings.
- assign new values to individual characters.
- delete individual characters.

Activity 8.3

8.3.1 Given:

```
shoutOut ← I Love computer programming
```

a. Give the reference to each vowel in *shoutOut* in the form variable[position].

b. Which characters are found at the position indicated by the following?

- shoutOut [3]
- shoutOut [7]

c. Write down the value of Length(shoutOut).

8.3.2 Which of the following programming code structures would you use to scroll through the string? Motivate your choice.

a. FOR-loop
b. REPEAT-UNTIL
c. IF-statement
d. WHILE-DO

Activity 8.3 *continued*

8.3.3 Read and trace the following algorithm:

```
Line 1:       greeting ← 'Hello'   //5 characters
Line 2:       Output ← ''
Line 3:       for j ← 1 to 5 do
Line 4:           output ← output+greeting[j]+'-'
              endfor
```

Reproduce the trace table given below to trace the algorithm:

LINE	GREETING	J	J <= 5	OUTPUT

8.3.4 Given: shoutOut ← 'Accessing a character in a String is easy'

n ← 16

m ← 3

Write down the characters found at the given position in each case:

a. shoutOut[n + 2] **b.** shoutOut[n – 6] **c.** shoutOut[n × 2]

d. shoutOut[n DIV 2] **e.** shoutOut[n + 3 × m] **f.** shoutOut[n × 3]

8.3.5 Data items for a road runner is given as a comma delimited string in the format: Name,Age group,Race distance,Time Given:

line ← John,50-59,10,39:45

a. aGive the variable[Position] reference for each of the commas in line.

b. Complete the table below, where Pos1 and Pos2 is the start position and end position of the data item in Line. Also provide a concatenation of characters for each data item.

DATA	POS1	POS2	LINE[ ]+ LINE[]…
Name	1	4	line[1]+line[2]+line[3]+line[4]
Age group	6	10	line[6]+...
Race distance			
Time			

c. Given the pseudocode:

dataItem ← ''

for index ← pos1 to pos2 do…,

dataItem ← X + Y[Z]

To extract the data from line, what must X, Y and Z be?

Activity 8.4

8.4.1 Write algorithms (pseudocode or flowchart) to:

a. display a word in reverse order.

For a solution do the following:

- Get user input and store it in a variable.
- Initialise a string variable to hold the reversed word to an empty string.
- Loop from the last character down to the first character.
- Concatenate these characters to the variable for the reversed word.
- Output the reversed word.

Take note

The new notation string Variable[position] makes accessing each character so easy.

b. display every second character in a string.

For a solution consider the following:

- Focus on the exit point in the for-loop.
- Must the loop scroll to the end of the string or just halfway?
- Use DIV to calculate the middle of a string if needed.
- How can the loop variable be used to increase the position variable by 2?

c. Provide trace tables for the algorithms in a) and b).

8.4.2 Write pseudocode to display the greetings 'Hello' and 'Hello World' as follows:

a.

H				
	e			
		l		
			l	
				o

b.

H					W				
	e					o			
		l					r		
			l					l	
				o					d

c. Test your solutions to a) and b) in a trace table.

8.4.3 Rectangular Text:

Let rectangular text be text that wraps around in a rectangle of a given width, where the width represents the number of characters per line.

Example input:	Output:
Rectangle width = 11 Text = 'labels are commonly used to display information'	`Labels are commonly used to display information`

Write an algorithm in pseudocode to display any text and width input, as Rectangular Text. Pay careful attention to the position variable when the characters are concatenated to form a line and don't forget to re-initialise the line variable to an empty string.

Example 8.5

Write a program that will display the user input with each character followed by a hyphen.

If the input is:	groovy	The output must be:	g-r-o-o-v-y-
	OCR software		O-C-R- -s-o-f-t-w-a-r-e-
	17 Church Str		1-7- -C-h-u-r-c-h- -S-t-r-
	R 100.00		R-1-0-0-.-0-0-

Note: spaces, digits, punctuation symbols are all characters.

IPO:

Input	Processing	Output	The Memo box component's Lines. Add(s) method will displays in one line. The for-loop loops through the word one character at a time from the first to the last character. The line g-r-o-o-v-y- must be prepared before it can be displayed. Line ← Line + word[k] + '-' , prepares the line, where word[k] is the letter in groovy at position k.
word	Line ← '' For k ← 1 to Length(word) Line ← Line + word[k] + '-' endfor	Line	
Component: inputBox		Component: Memo box	

Delphi code:

```
var
      sWord: String;
      k: Integer;
      sLine: String;
begin
             memOutput.Clear;
Line 1:   sWord := InputBox('Question 1','Enter text: ','');
Line 2:   sLine := '';
Line 3:   for k := 1 to Length(sWord) do
          begin
Line 4:       sLine := sLine + sWord[k] + '-';
          end;
Line 5:   redOutput.Lines.Add(sLine);
end;
```

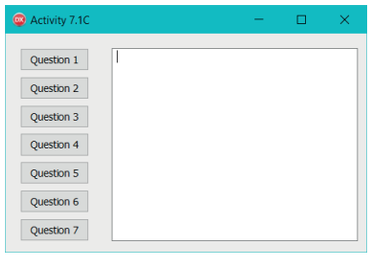
Example 8.5 *continued*

Complete the trace table:

LINE	sWORD	sWORD[k]	sLINE	k	k <= LENGTH(sWORD)	OUTPUT
1	groovy					
2			''			
3				1	true	
4		g	g-			
3				2	true	
4		r	g-r-			
3				3	true	
4		o	g-r-o-			
^						
5						
Stop						

Activity 8.5

Open the Delphi project **CharacterAccess_p** in folder 08 – Character Access. Code for the example as shown alongside, run the program and click question 1. Input the text, groovy, and confi rm the output, g-r-o-o-v-y-.



8.5.1 Button [Question 1]

Change the program to output g-r-o-o-v-y, without the last hyphen. The following questions will assist you to find one of many possible solutions.

a. Must *sLine* be initialised to ''? What if it is initialised to the first character of *sWord*?

```
sLine ← sWord[1];
```

b. What effect will such a change have on Line 3 and Line 4 in the example code on the previous page?

c. What value must *k* start with?

d. Will the current order of concatenation work? i.e. sLine + sWord[k] + '–'. If not how will you change it?

8.5.2 For each of the question below, complete the code.

a. Button [Question 2]

Display the word entered by the user in reverse order.

Example: Input: YOLO output: OLOY

Activity 8.5 *continued*

b. Button [Question 3]

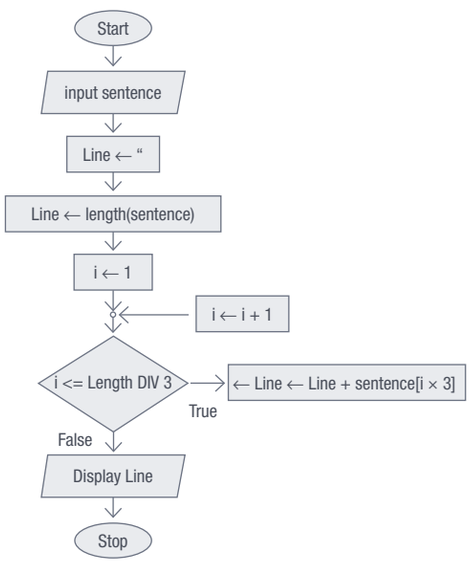
Display every third character of the user input for sentence by implementing the given flowchart.

Example:

Input: sentence ← learning to code

Output: Line ← ai d

There are 2 spaces between the i and d.



c. Button [Question 4]

Display the word entered with each letter below each other starting from character at position 1.

Example input: CAT output: C
A
T

d. Button [Question 5]

Display the word entered for each letter below each other starting from the last character.

Example input: CAT output: T
A
C

e. Button [Question 6]

Display the string 'Hello' as indicated:

H				
	e			
		l		
			l	
				o

f. Button [Question 7]

Display the string 'Hello World' as indicated:

H					W				
	e					o			
		l					r		
			l					l	
				o					d

Activity 8.6

Create a Delphi project to implement your Rectangular text algorithm. Name the project **RectText_p** and the unit **RectText_u** and save them in a folder named 08 – Rectangular Text.

Provide the interface with the following:

- Labels for width and text.
- Edit boxes for width and text.
- Set the width edit to accept numbers only. The Edit Box for width must be set to receive numbers only.
- A button with the caption 'Show Rectangular Text'.
- A memo component to display the text. Set the font for the memo to Courier New.

Activity 8.6 *continued*

8.6.1 Write down a Delphi code statement to do each of the following:

- **a.** Declare variables sText, *iWidth*, iIndex and *sLine*. The prefix indicates each data type.
- **b.** Clear the memo component.
- **c.** Read the width and text from the edit boxes and assign it to *iWidth* and *sText*, respectively. Convert the data type if required.
- **d.** Initialise *sLine* to an empty string.
- **e.** A for-loop to scroll from position 1 to the length of the text using the loop variable *iIndex*.
- **f.** Concatenate the character to *sLine*.
- **g.** Test if the position is a factor of *iWidth*. Use the MOD operator.
- **h.** If *iWidth* is a factor of *iIndex* then: add *sLine* to the memo.
- **i.** Re-initialise sLine to an empty string.
- **j.** After the execution of the For-loop, add the remaining text to the memo.

8.6.2 Insert your code statements in 8.6.1 as code for button [Show Rectangular Text] to be executed when the button is clicked.

8.6.3 Improve your program by checking if data is present in the edits. The edits should not be empty when the button [Show Rectangular Text] is clicked. If it is empty then show a suitable error message in a message box.

8.6.4 Add a second button with caption [Best Fit] to the program.

Code button [Best Fit] to:

- **a.** Hardcode iWidth1 = 6 and iWidth2 = 7.
- **b.** Test which of the two values in (a) gives the best rectangular fit of characters. This would be a rectangle with the biggest number of characters in the last line. If more than one of the hardcoded width numbers give the same number of unfilled places in the last line, then choose the larger width.
- **c.** Display the best width in edtWidth.

Guided Activity 8.5 Accessing individual characters

Look at the following code snippet.

Strings

```
sName := 'Lundi';
sPhrase := 'I love programming!';
```

Using a pen and paper, answer the following questions.

8.5.1 What is the value of the following characters?

- **a.** sPhrase[1]
- **b.** sPhrase[5]
- **c.** sPhrase[12]
- **d.** sName[3]
- **e.** sName[2]

8.5.2 Give the location of each vowel (a, e, i, o and u) in the strings above, using square bracket notation.

UNIT

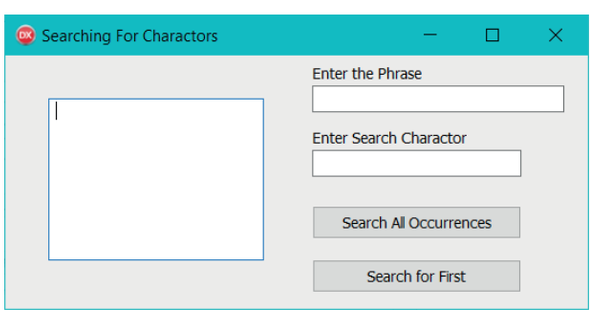
8.4 Manipulating strings

Now that you know how to scroll through each character of a string, you can start manipulating the string. In this unit, you will learn how to do the following:

- find all instances of a specific character.
- change all instances of a character into a different character.
- delete a character from a string.
- insert a character into a string.
- determine the position of a character in a string.
- find a character.

Guided Activity 8.6 Finding all occurrences of a character in a string

8.6.1 Open the **SearchCharacter_p** project from the 08 – Find Character folder.



8.6.2 Create an *OnClick* event for the [Search All Occurrences] button to determine the positions of a search character in a string:

a. Declare four local variables: *sPhrase*, *cSearch*, *iIndex* and *bFlag* where *sPhrase* is the search string, *cSearch* is the search character, *iIndex* will be used as a loop counter and *bFlag* will be used to determine whether the search character *cSearch* is found in the search string *sPhrase*

```
var sPhrase:string;
    cSearch:char;
    iIndex:Integer;
    bFlag:Boolean;
```

b. Clear the *memDisplay* memo component:

```
memDisplay.Lines.Clear;
```

c. Set a variable *bFlag* to FALSE. This variable will be used to determine whether the search character is in the search string or not.

```
bFlag := False;
```

d. Read the search string from the *edtPhrase* edit box component and assign the value to the *sPhrase* variable.

e. Read the search character from the *edtSearch* edit box and assign the value to the *cSearch* variable. Remember that you are reading string data from the *edtSearch* edit box and therefore cannot assign it directly to the character variable *cSearch*. You will get a type mismatch error. To ensure you don't get an error:

```
sPhrase := edtPhrase.Text;
cSearch := edtSearch.Text[1];
```

f. Loop using a loop counter from 1 to the length of sPhrase

```
for iIndex := 1 to Length(sPhrase) do
```

g. Check whether the character at the loop counter position is equal to the search character *cSearch*.

```
if sPhrase[iIndex] = cSearch then
```

If the character at the loop counter position is equal to the search character *sSearch* then display the loop counter position. Also set the *bFlag* variable to true. This indicates that a match has been found.

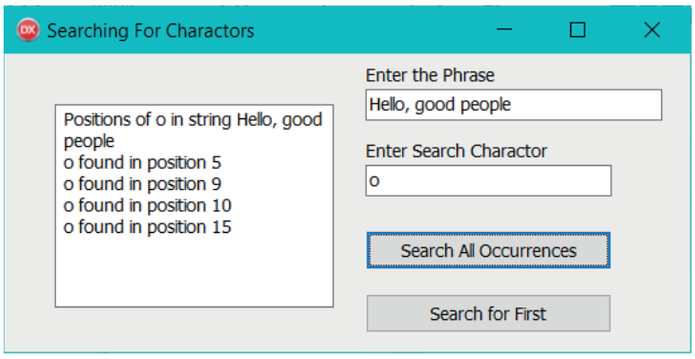
```
memDisplay.Lines.Add(cSearch+' found in position
'+IntToStr(iIndex));
bFlag:=True;
```

h. Once the loop is exited, determine whether a match has been found. If a match has not been found, then display a message 'Character not found in string'.

```
if not(bFlag) then
memDisplay.Lines.Add('Character not found in string');
```

```
procedure TForm1.btnSearchAllClick(Sender: TObject);
var sPhrase:string;
    cSearch:char;
    iIndex:Integer;
    bFlag:Boolean;
begin
    memDisplay.Lines.Clear;
    bFlag := False;
    sPhrase := edtPhrase.Text;
    cSearch := edtSearch.Text[1];
    memDisplay.Lines.Add('Positions of '+cSearch+' in string
      '+sPhrase);
    for iIndex := 1 to Length(sPhrase) do
    begin
      if sPhrase[iIndex] = cSearch then
      begin
         memDisplay.Lines.Add(cSearch+' found in position
    '+IntToStr(iIndex));
         bFlag := True;
      end;
    end;
    if not(bFlag) then
      memDisplay.Lines.Add('Character not found in string');

end;
```

8.6.3 Save and run the project.



8.6.4 You can also find the FIRST occurrence of a character in a string. Study the code below:

Finding the position of the first matching character

```
Line 1:   iPosition := 0;
          // the initial position of the search
          //character in the search string
Line 2:   sPhrase := edtPhrase.text;
          // read the value for sPhrase – the
/         /search string
Line 3:   cSearch := edtSearch.text[1];
          // read the value for cSearch – the search character
Line 4:   for i:= 1 to Length(sPhrase) do
          //loop from position 1 to length(sPhrase)
Line 5:   begin
          //if the character at loop counter position i=the search
          //Character and iPosition=0
Line 6:   if (sPhrase[i] = cSearch) and (iPosition = 0) then
Line 7:   iPosition := i;
          //if condition in line 6 is true, assign value
          //of loop counter to iPosition
Line 8:   end;
Line 9:   if iPosition = 0 then
          // when the loop is exited, check whether iPosition=0
Line 10:  memDisplay.Lines.Add('o not found')
          //if iPosition =0 is true, then no match found
Line 11:  Else
          //otherwise a match is found.
          //Display the position of the match
Line 12:  memDisplay.Lines.Add(('o found at position ' +
            IntTostr(iPosition));
```

Guided Activity 8.6 Finding all occurrences of a character in a string *continued*

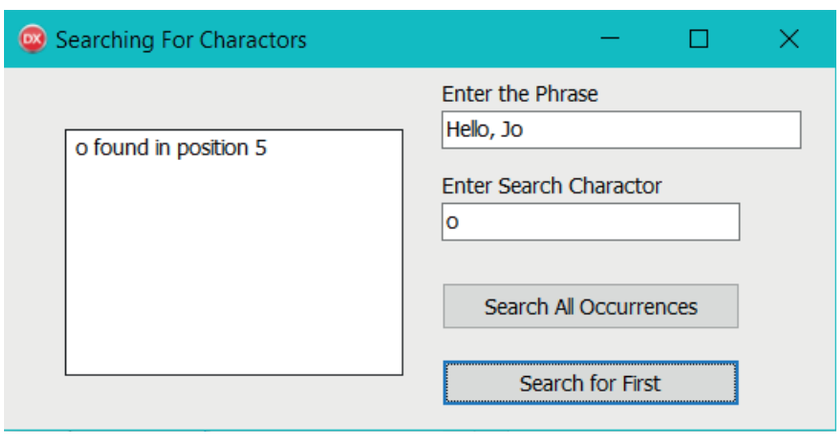
Assume 'Hello Jo' will be read for *sPhrase* in Line 2 and character 'o' for *cSearch*. Trace through the flowchart using these values.

Line Number	iPosition	sPhrase	cSearch	i	i <= 8	sPhrase[i]=cSearch and iPosition=0	iPosition=0	Output
1	0							
2		Hello Jo						
3			o					
4				1	T			
6						F		
4				2	T			
6						F		
4				3	T			
6						F		
4				4	T			
6						F		
4				5	T			
6						T		
7	5							
4				6	T			
6						F		
4				7	T			
6						F		
4				8	T			
6						F		
4				9	F			
9							F	
12								5

Guided Activity 8.6 Finding all occurrences of a character in a string *continued*

8.6.5 Open the **SearchCharacter_p** project. Create an *OnClick* event for the [Search for First] button to determine the FIRST occurrence of a search character in a search string.

8.6.6 Save and run the project.



Activity 8.8

8.8.1 Given:

sentence ← 'I love computer programming'

searchChar ← 'r'

count ← 0

index ← 1 to length(sentence)

a. What notation do we use to refer to a character in a string? Give an example.

b. Give two Boolean expressions to test for the letter 'r' in sentence.

c. Now place one of your Boolean expressions as the test condition of an IF-statement.

d. Give a reason for your choice of Boolean expression in (c).

e. In the body of the IF-statement, provide code to count the number of 'r' characters in the sentence.

f. Which character would you use as test character if you had to count the number of words?

g. Write down a test condition for vowels.

h. Describe a test condition that will determine if a character is a consonant in a sentence.

i. Provide Delphi code for your test condition in (h)

8.8.2 Complete the trace table for the algorithm given below.

Line 1	`Label ← 'programming'`
Line 2	`ch ← 'r'`
Line 3	`for i ← 1 to length(Label) do begin`
Line 4	`  if ch = Label[i] then`
Line 5	`  Display ch + ' found in position ' + i`
	`  newline`
	`  end if`
	`end for`

Activity 8.8 *continued*

Line	Label	ch	i	i <= Length(Label)	Label[i]	Label[i] = ch	Output
1	programming						
2		r					

8.8.3 Read the following Delphi code.

```
    var
Line 1:     sIdNumber: String;
Line 2:     sLine: String;
          begin
Line 3:   redOutput.Clear;
Line 4:   sIdNumber := edtIDNumber.Text;
Line 5:   if StrToInt(idNumber[7]) >= 5 then
Line 6:     redOutput.Lines.Add('Gender: male')
          else
Line 7:     redOutput.Lines.Add('Gender: female')
          end;
```

a. Explain the code for each numbered line.

b. The purpose of the code is to determine a person's gender. Explain how the code determines the gender?

c. Will the idNumber[7] >= '5' give the same result? Open the **Gender_p** project from the 08 – Show Gender folder. Run it, then make the suggested change to see if gives the same result.

d. Explain why the change in (c) worked or did not work.

Activity 8.9

8.9.1 Write an algorithm in pseudocode to count the number of vowels in a string.

Example: Input: 'Count the vowels' Output: 5

8.9.2 Give an algorithm in flowchart form to display a * in place of a vowel.

Example: Input: mother Output: m*th*r

8.9.3 Give a flowchart to count the number of words in a sentence.

Example: Input: 'Count the words' Output: 3

Activity 8.9 *continued*

8.9.4 Complete the algorithm given below by providing the missing pseudocode. The algorithm should accept an encoded #-delimited string, search for the delimiter and display each data item below each other.

```
Line ← Enter Line
Length ← Length(Line)
Item ← ''
For k ← 1 to Length do begin
If (a) then
       Display Item
       Newline
       (b)
   Else
       (c)
End for
```

Example 8.6

Input: John Doe#27/06/2018#Accident on N2

Output: John Doe
27/06/2018
Accident on N2

NOTE: the delimiter is the '#' sign.

8.9.5 Provide trace tables for your algorithms in 8.9.2 to 8.9.3 above. Trace using the given examples.

Example: Write Delphi code that will count the number of vowels in the text input. To write the Delphi code, you must have a solution. Writing the Delphi code is a translation of your solution that can be in the form of pseudocode or a flowchart.

The IPO table will assist you in formulating a solution because it breaks up the task into three smaller tasks focussing on input, processing and output.

If the input is 'Count the number of vowels' then the output is 8.

Here is the IPO table:

INPUT	PROCESSING	OUTPUT

text	`count ← 0`
	`For k ← 1 to Length(text)`
	`  If text[k] in [a,e,i,o,u,A,E,I,O,U] then`
	`  count ← count + 1 // code for a running total`
	`  endif`
	`endfor`
		count

Component: inputBox		Component: MemoBox

Take note

The algorithm loops through the text and increases count if a vowel is found.

Note how the IN operator is used to check if the character at position k is a vowel.

Some of the questions you can ask when translating the solution into Delphi code includes:

- What are the variables and how are they declared?
- What are the data types for these variables?
- Where must count be initialised? Inside or outside the loop?
- How do I implement the for-loop?

Activity 8.9 *continued*

Here is the Delphi code:

```
      var
        sText: String;
        k: Integer;
        iCount: integer;
      begin
         redOutput.Clear;
Line 1:    sText := InputBox('Question 1','Enter text: ','');
Line 2:    iCount := 0;
Line 3:    for k := 1 to Length(sText) do begin
Line 4:   if sText[k] in ['a','e','i','o','u','A','E','I','O','U'] then
Line 5:      iCount := iCount + 1;
          end;
Line 6:    redOutput.Lines.Add('The text has '+IntToStr(icount)+'vowels');
      end;
```

Complete the trace table:

Line 1: sText = Count the number of vowels						
Line	**iCount**	**k**	**k <= Length(sText)**	**sText[k]**	**sText in [set of vowels]**	**Output**
2	0					
3		1	true			
4				C	false	
3		2	true			
4				o	true	
5	1					
3		3	true			
4				u	true	
5	2					

Activity 8.10

Open the **SearchChars_p** project from the 08 – Search Characters folder.

8.10.1 Button [Question 1]

Complete the incomplete code for counting vowels as indicated in the example given above.

Input: Output:

Complete the code

Count the number of vowels Number of vowels: 6

Number of vowels: 8

Activity 8.10 *continued*

8.10.2 Button [Question 2]

Provide the missing code to display the consonants of the text input. Also display a * where a vowel is found.

Input: mother Output: m*th*r

The cat sat on the mat Th* c*t s*t *n th* m*t

8.10.3 Button [Question 3]

Complete the code to count the number of words in a sentence. Consider the following:

- What character will you use in your test?
- Must the count be initialise to zero?

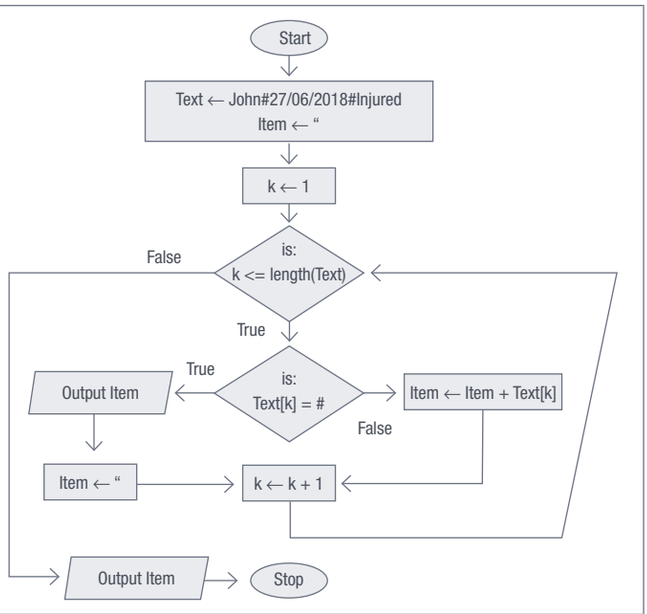
Input: Count the number of words Output: Number of words: 5

8.10.4 Button [Question 4]

Complete the code to fully implement the given flowchart. The completed program should display data items encoded into a #-delimited string below each other.

Example:

Input: John#27/06/2018#Injured
Output: John
27/06/2018
Injured



8.10.5 Button [Question 5]

Complete the code to display the input text over two lines. Letters in the first line and all digits in the second line. The lines must vertically break up the text. Removal of the digit must leave a space and the digit must be below the space when displayed. Consider the following:

- Should string concatenation happen in both the TRUE and FALSE part of the IF-statement?
- Do you test for letters or digits?

Input: Sh1am9iel Dra5m9at Output: `Sh am iel Dra m at`

`1 9 5 9 1 9 5 9`

Digits are replaced by spaces for display on the first level. The removed digits are displayed on the second level immediately below the original position.

Activity 8.10 *continued*

8.10.6 Button [Question 6]

Complete the code to determine the sum of all the even digits in the input. The input string must consist of digits only. No input validation is needed.

Input:	74658349	Output:	22
	5906065123084		26

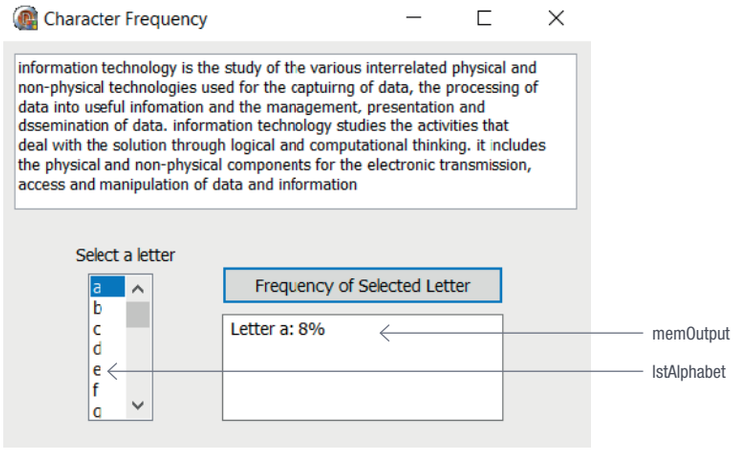
8.10.7 Button [Question 7]

The prime digits are 2, 3, 5, and 7. Provide code to count the number of prime digits in a number string.

Input:	74658349	Output:	3
	5906065123087		5

Activity 8.11

Open the Delphi project **CharFrequency_p** located in the 08 – Find Character Frequency folder. The program determines how often the selected letter is found in a paragraph. The result is displayed as a percentage of the total number of characters in the paragraph.



Given the following variables:

VARIABLE	PURPOSE
sText	Represents the paragraph
k	Loop index
cLetter	Selected letter
iTotal	Total number of characters in Text
iCount	How many times the Letter is found in Text
rFrequency	The percentage of Letter in Text

Activity 8.11 *continued*

8.11.1 Write down a Delphi code statement or expression for each of the following:

a. Read the selected letter from the list box.

b. Determine the total number of characters in the paragraph.

c. Initialise iCount to zero.

d. A FOR-loop to scroll from position 1 to the length of the paragraph.

e. Test if the character at the current position of the loop index is equal to the selected letter.

f. Increment iCount.

g. Calculate the frequency using iCount and iTotal.

8.11.2 Use the code in 8.11.1 to complete the code for [Frequency of Selected Letter] button.

Input:	a	Output:	Letter a: 8%
	b		Letter b: 0%

CHANGING ONE OR MORE CHARACTERS IN A STRING

You can replace a character in a string as follows:

```
    var sVariable:string;
        iPosition:integer;
        cChar:char;
begin
      …
      sVariable[iPosition] := cChar;
      …
end;
```

The value in variable *cChar* is assigned to the character at position *iPosition* in the string variable *sVariable*. The 'old' value in *sVariable*[iPosition] is overwritten by the value of *cChar*.

Changing a single character

```
sWord := 'Truck';

sWord[3] := 'i';
ShowMessage(sWord); // Trick

sWord[1] := 'B';
ShowMessage(sWord); // Brick
```

This technique can be combined with the technique to search for a character to replace either a specific character or all instances of a character. The code snippet below shows how all instances of the character 'o' are replaced by the character 'a'.

Replacing all matching characters

```
sPhrase := 'Hello, World!';
for i := 1 to Length(sPhrase) do
begin
  if sPhrase[i] = 'o' then
    sPhrase[i] := 'a';
end;
ShowMessage(sPhrase); // Hella, Warld!
```

Activity 8.12

Write a Delphi program to create a strong password based on the user's name. Save the project as **StrongPassword_p** in a folder named 08 – Strong Password.

Write code for the [Password] button to create a password as follows:

8.12.1 Receive the name of the user by using an *Inputbox*.

8.12.2 Determine the length of the name. If the name is shorter than eight characters, add as many of the characters '#', '$', as required to make the length of the string eight.

8.12.3 Replace all the vowels in the word with the letter 'Q'.

8.12.4 Replace 'b', 'm', and 'z' with the symbol '^'.

8.12.5 All other letters remain as they are.

8.12.6 Display the password.

DELETING A CHARACTER

The final two techniques you will learn about both use the same general algorithm. In the simplest terms, the algorithm works as follows:

- Store the position of the character you want to delete.
- Add all characters before this position to a new string called *sStart*.
- Add all characters after this position to a new string called *sEnd*.
- Set the existing variable equal to *start* + *end*.

This algorithm works by storing all the characters before the deleted character and all the characters after the deleted character in new strings. Since neither of these strings contain the deleted character, they can be combined to form the starting word without the deleted character.

Did you know

There are many different algorithms that can be used to delete or insert characters. However, the algorithm discussed in this section is one of the easiest algorithms to remember and use.

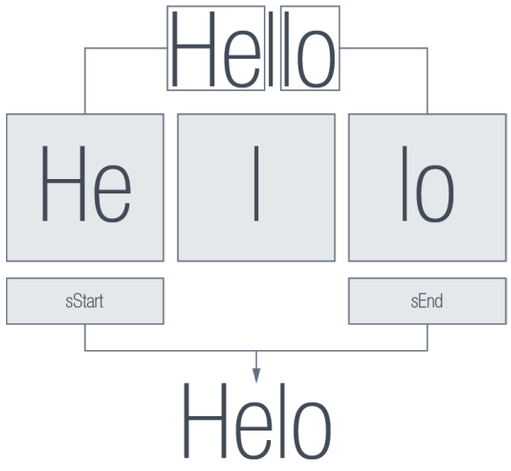


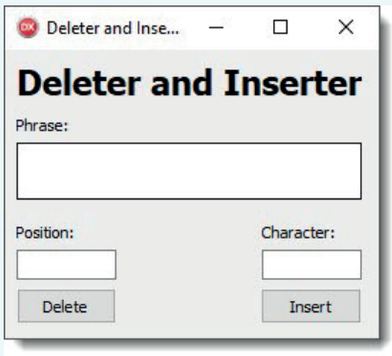
Figure 8.2: *To delete a character, you can recreate the string without the character*

To see how this algorithm is used in practice, work through the following example.

Example 8.7 Deleting a character at a specific position

For this project, you will allow the user to delete any character from a string. To do this:

1. Open **DeleteAndInsert_p** Project from the 08 – Deleter and Inserter folder.



2. Create an *OnClick* event for the [Delete] button:
   a. Declare five local variables: *sPhrase* , *sStart*, *sEnd*, *iCount* and *iPosition*.
   b. Initialise *sStart* and *sEnd* as empty strings.
   c. Read the string from the memo component and store the value in *sPhase*.
   d. Read the position of the character that you want to delete from the *edtPosition* edit box and store the value in variable *iPosition*.
   e. Create a new string *sStart* to extract all the characters before the character that will be deleted, and another string *sEnd* to extract all the characters found after the position of the deleted character:

```
for iCount := 1 to Length(sPhrase) do
begin
  if iCount < iPosition then
    sStart := sStart + sPhrase[iCount];
  if iCount > iPosition then
    sEnd := sEnd + sPhrase[iCount];
  //
end;
```

   f. If *sPhrase*:='Peter Pan Outfit' and the position *iPosition* of the character that must be deleted is 8, then once the loop is executed *sStart* will be assigned the value 'Peter P' and *sEnd* will be assigned the value 'n Outfit'
   g. Join *sStart* and *sEnd* and assign it to *sPhrase*.
3. Save and run the project.
4. Write code for the OnClick event of the [Insert] button to insert a character in the position where you deleted a character. Look at the example on the next page before attempting this.

INSERTING A CHARACTER/S

You can insert a character/s in a string at a given position.
Given:

```
sPhrase := 'Today is looming';
```

Suppose you want to insert the character 'B' at position 10 in the string *sPhrase* then follow the steps outlined below:

- Create a string *sStart* by joining the characters in positions 1 to 9:

```
sStart:='';  //  empty string
for icount := 1 to 9 do
  sStart := sStart + sPhrase[iCount];
```

- Create a string *sEnd* by joining the characters in positions 10 to length of *sPhrase*:

```
sEnd := '';
for iCount := 10 to length(sPhrase) do
  sEnd := sEnd + sPhrase[iCount];
```

- Join *sStart*, the new character to insert and the *sEnd* strings and assign to *sPhrase*:

```
sPhrase := sStart+'B'+sEnd;
```

Here is the Delphi code:

Insert event

```
procedure TfrmDeleterAndInserter.btnInsertClick(Sender: TObject);
var
  sPhrase, sStart, sEnd, sChar : String;
  i, iPosition : Integer;
begin
  sStart := '';
  sEnd := '';
  sPhrase := memPhrase.Text;
  sChar := edtCharacter.Text;
  iPosition := StrToInt(edtPosition.Text);
  for i := 1 to Length(sPhrase) do
  begin
    if i < iPosition then
      sStart := sStart + sPhrase[i];

   if i >= iPosition then
      sEnd := sEnd + sPhrase[i];
  end;
  sPhrase := sStart + sChar + sEnd;
  memPhrase.Text := sPhrase;
end;
```

For enrichment

The setLength function can be used to change the length of a string.

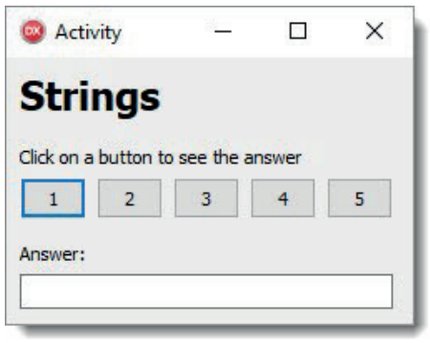
For example:

setLength (sText, 6)

This sets the new length of sText to 6 characters. If the original length is more than 6 the characters from position 7 onwards will no longer be part of the string. If the original length is less than 6, say 4 then space for 2 more characters are reserved but not initialised. The two extra spaces contains garbage and can result in errors. To avoid, this always initialise the extra space. Fill it with the space-character.

Activity 8.13

8.13.1 Use the user interface below to complete the following tasks. Open the **Activities_p** project from the 08 - String Manipulation folder. Each of the buttons should be used for one of the questions.



a. Show how long your name is.

b. Insert the dash symbol (-) between each letter of your surname.

c. Replace each vowel in your surname with the asterisk character (*).

d. Show the position of all the 's' characters in the word 'Mississippi'.

e. Delete the third character of any word entered into the text box.

8.13.2 Open the program **SmartphoneLogin_p** from the 08 – Smartphone Login folder. Using this application, update the 'forgotten password' clue so that it shows three randomly selected letters from your password.

Activity 8.14

8.14.1 Given

Text ← '—abcd—

POSITION	1	2	3	4	5	6	7	8
TEXT	-	-	a	b	c	d	-	-

Provide an algorithm to move the characters in position 3 to 6:

a. two places to the right

b. two places to the left.

POSITION	1	2	3	4	5	6	7	8
TEXT	a	b	c	d	-	-	-	-
	-	-	-	-	a	b	c	d

In each case avoid assigning characters beyond the string boundary. The length of text must remain eight characters long.

8.14.2 Given the string:

Word ← 'characters'

Provide Delphi statements to do the following in order:

a. replace the letter 'e' with a letter 'o'.

b. Given:

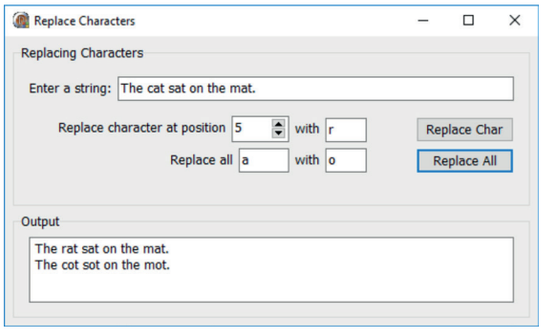
Text ← 'ming_ ',

Add the contents of the variable Text at position 5 in valuable Word.

8.14.3 Provide an algorithm to show how four characters are deleted from a 10-character string starting at position 2.

Activity 8.15

8.15.1 Complete the following:



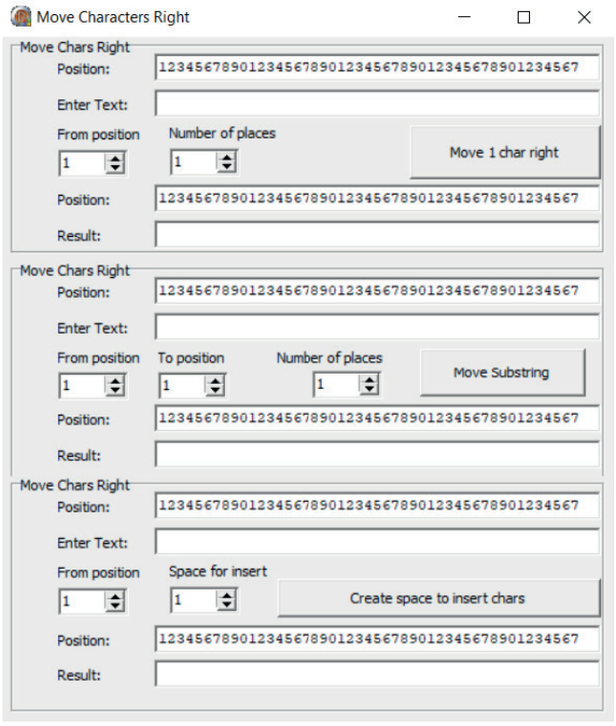
a. Write a pseudocode algorithm that will replace a character at any position with a new character in a string.

b. Provide a flowchart to replace all occurrences of a user selected character with a second character provided by the user.

c. Now open the incomplete Delphi project, **ReplaceChars_p** located in the 08 – Replace Characters folder.

- Provide code for the [Replace Char] button to implement your algorithm in a).
- Provide code for the [Replace All] Button according to your solution in b).

8.15.2 Open the project **MoveCharsRight_p** from the 08 – Move Characters Right folder.



a. Provide algorithms for b) to d) below before coding the solutions.

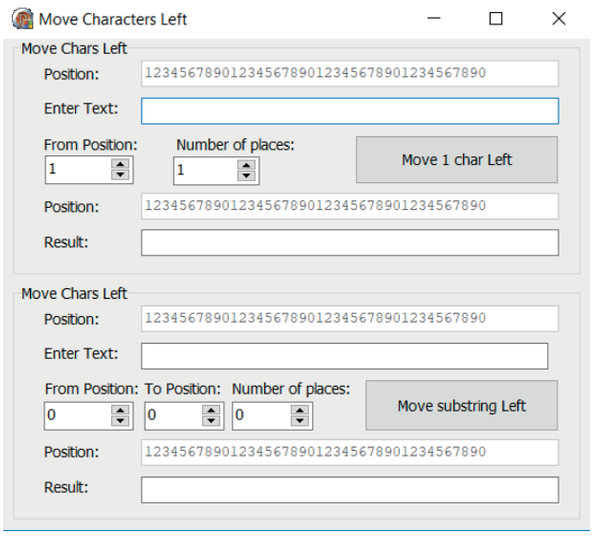
b. Given the position of the character and the number of places it must be moved to the right, complete the code for button [Move 1 char right].

c. Provide code for the button [Move Substring] that will move the text (between two positions inclusive) the chosen number of places to the right.

d. Code the button [Create space to insert chars] so that all the characters from the position indicated moves to the right the required number of places.

Activity 8.15 *continued*

8.15.3 Open the project **MoveCharsLeft_p** from the 08 – Move Characters Left folder.

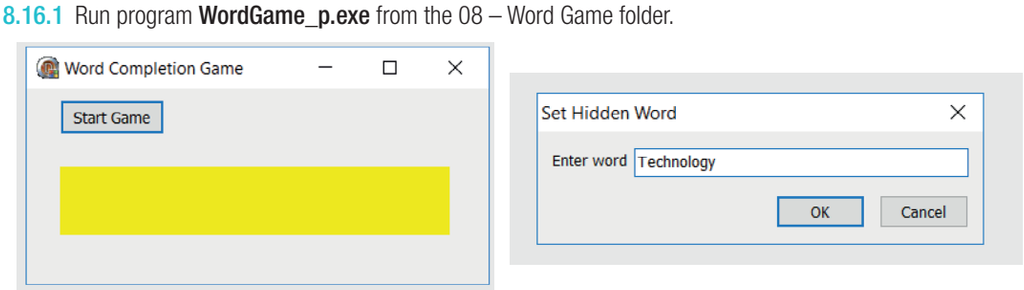


a. Provide code for the button [Move 1 char Left] to move the character from the position indicated to the number of places to the left.

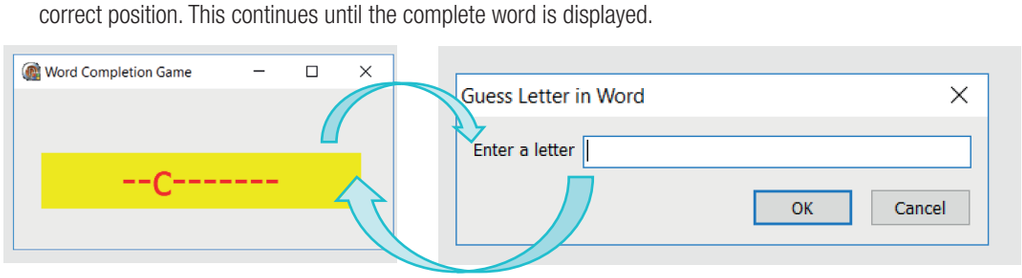
b. Button [Move substring Left] moves consecutive characters between from and to positions to the left.

Activity 8.16

8.16.1 Run program **WordGame_p.exe** from the 08 – Word Game folder.



When the game starts Player 1 clicks the [Start Game] button to set the hidden word and enters the word that must be hidden. The computer randomly selects a character from the hidden word and displays all occurences of that character and '-' s for all other characters. Player 2 starts guessing the missing characters. The computer determines if Player 2's character is in the word and if found, displays it in the correct position. This continues until the complete word is displayed.



a. Recreate the word game and in addition provide one improvement, for example, count the number of guesses.

b. Change the user interface to include your improvement.

Activity 8.16 *continued*

8.16.2 In many languages the adjective follows the noun in the sentence structure.

'Inenekazi ehlihle lingena eholweni'

A word for word translation for this isiXhosa sentence to English becomes:

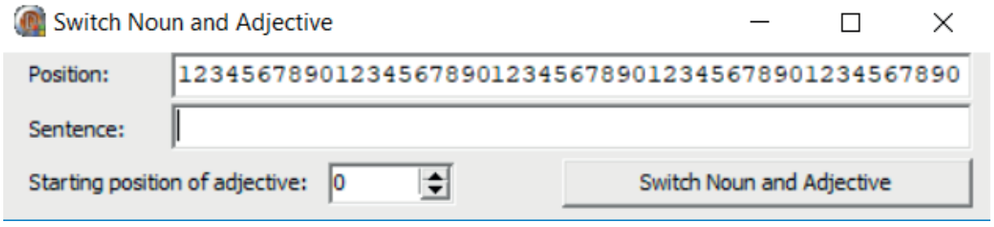
'The lady beautiful enters the hall'

The adjective 'beautiful' comes after the noun 'lady'.

Given the **sentence** and the **starting position** of the adjective, develop a Delphi program to switch the noun and adjective around so the sentence reads:

'The beautiful lady enters the hall'

Open the project **NounAdjectiveSwitch_p** from the 08 – Noun Adjective Switch folder.



Complete the code for the [Switch Noun and Adjective] button to switch the noun and adjective.

Summary

STRING MANIPULATION

Combining strings and determining the length of a string

In Delphi multiple small strings can be combined into a single large string.

```
sFirstName := 'Sharise';
sSurname := 'Marone';
sName := sFirstName + ' ' + sSurname; // 'Sharise Marone'
```

A String can consist of any number of characters. Manipulating a string like appending, inserting, replacing and deleting one or more characters would require that the computer can access the individual characters in the string and has some way to determine the length of the string.

- sPhrase := 'I love programming!';
- iLength := Length(sPhrase); //19

Formatting Characters

One way to ensure that your spacing is correct when combining string is to make use of the tab and newline characters.

Name	Character	Example

Tab	#9	'M1:' + #9 + '71%';
		'Average:' + #9 + '85%;'
		//M1: 71%
		//Average: 85%

Newline	#13	'M1: 71%' + #13 + 'M2: 73%';
		//M1: 71%
		//M2: 73%

Scrolling through a string

It is possible to access each character in a string. To do this, you can use the string variable with a square bracket and a number between the square brackets. The number refers to the position of the character you are referring to.

```
sValue := 'Hello, World!';
cFirst := sValue[1]; // H
cSecond := sValue[2]; // e

sValue[13] := '?'; // sValue = 'Hello, World?'
```

```
var
      sName: String;
index: Integer
begin
      sName := 'THANDI';
      for index := 1 to Length(sName) do
      begin
         redOutput.Lines.Add(sName[index]);
      end;
end;
```

The Output:

T
H
A
N
D
I

NOTE: the downto keyboard

```
var
      sName: String;
      index: Integer
begin
      sName := 'THANDI';
      for index := Length(sName) downto 1 do
      begin
         redOutput.Lines.Add(sName[index]);
      end;
end;
```

The Output:

I
D
N
A
H
T

Manipulating strings

Individual characters in a string can be accessed as follows:

VariableName[position of character]

Outcomes:

- Find all instances of a specific character.
- Change all instances of a character into a different character.
- Delete a character from a string.
- Insert a character into a string.
- Determine the position of a character in a string
- Finding a character

Answer to following short questions in writing:

1. Given the string: sText ← 'Creative people will benefit most from changes in technology'
   a. True or false?
      i. Delphi code Length(sText) will have a value of 60.
      ii. Number of words in *sText* is equal to the number of spaces.
   b. What is the word represented by sText[2] + sText[3]+sText[4] + sText[40] + sText[51]?
   c. What is the effect of the assignment sText[31] := 'u' ?
2. Provide an algorithm in pseudocode to delete a word from a sentence.
3. Given the comma delimited text:

   Western Cape#Theewaterskloof#43.5%#13/8/20187

   Write an algorithm display the information as follows:

   Western Cape

   Theewaterskloof

   43.5%

   13/8/20187
4. Trace through the following algorithm to find out what it does:

```
Sentence ← 'password to enter secret room is opensesame'
Word1 ← ''
Word2 ← ''
For k ← 1 to length (sentence)
  If k MOD 2 = 0 then
    Word2 ← Word2 + sentence[k]
  Else
    Word1 ← Word1 + sentence[k]
Output Word1 + Word2
```

Complete the following programming challenges:

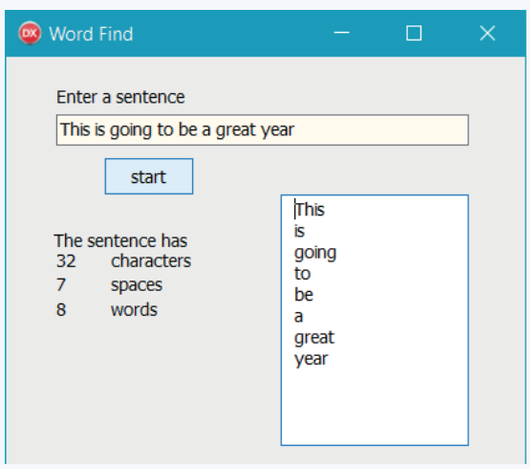
5. The program **WordFindError_p** should work as follows:
   - The user enters a sentence in the edit box.
   - The code in the [Start] button should do the following:
     - Apply data validation to ensure that an empty string cannot be entered into the variable
     - Save the sentence into a variable
     - Find the length of the sentence
     - Use a while loop to go through the sentence to find the number of spaces
     - Test if the word is a space
     - If it is not a space use a repeat loop to put the letters in a word
     - Display the words in the memo component
     - Display the number of letters in the sentence, number of spaces in the sentence and the number of words in the sentence in a label as shown below.

   The program does not work correctly.

   a. Open the project **WordFindError_p** in the 08 – Word Find Error folder and run the program to see the display.
   b. Use a trace table to find the errors.

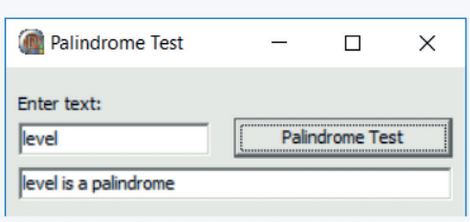
c. Correct the errors and run the program.

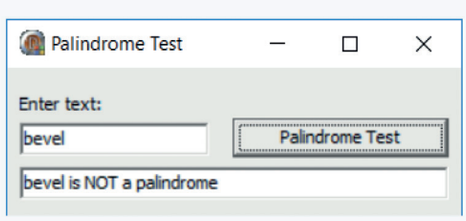
d. Test your program with the words 'This is going to be a great year'. The display should resemble the image below.



6. A palindrome is text that reads the same from left to right and right to left, for example, level and 123321.

a. Write a program to determine if a word or number is a palindrome.

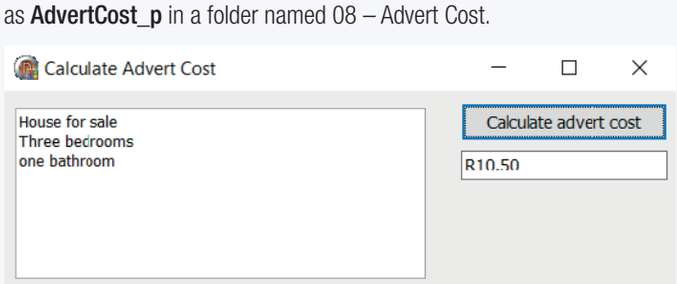




b. Save the project as **PalindromeTest_p** and save it in a folder named 08 – Palindrome.

7. The price for an advert in the Delphi Herald is R1.50 per word.

Create a program that will accept the text, determine the number of words and calculate the cost of the advert. Save the project as **AdvertCost_p** in a folder named 08 – Advert Cost.



8. The format of a #-delimited string is as follows:

<School name>#<Excursion>#<Date>#<Group size>

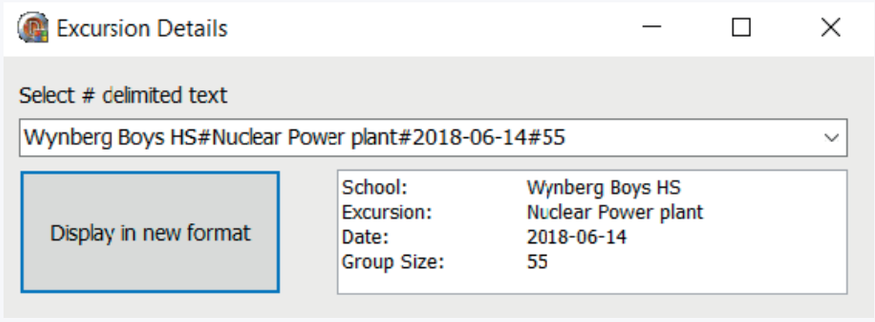
Example: Wynberg Boys HS#Eskom: Koeberg Centre#2018-06-14#55

a. Open the pExcursionInfo project from the 08 – Excursion folder and complete the program.

b. Write a program that will accept the # delimited string and output the information in the format.

School name:	<School name>	School name:	Wynberg Boys HS
Excursion:	<Excursion >	Excursion:	Eskom: Koeberg Centre
Date:	<Date>	Date:	2018-06-14
Number of learners:	<Group Size>	Number of learners:	55

c. Some # delimited strings are already loaded in the ComboBox.



9. Upon analysis of large amounts of English text, the frequency of letters are as follows:

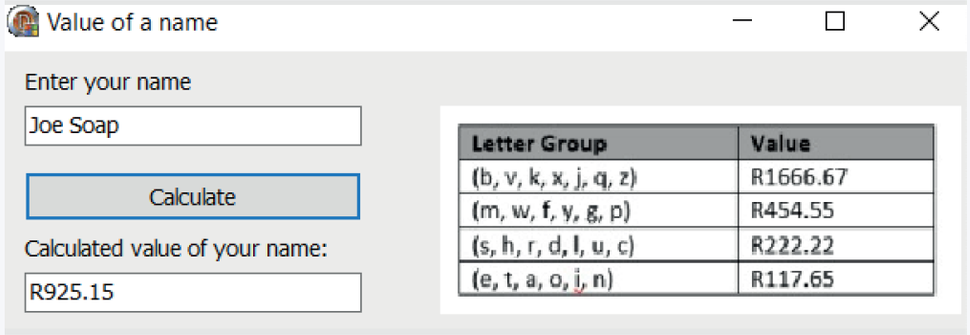
e	t	a	o	i	n	s	h	r	d	l	u	c
12.7	9.1	8.2	7.5	7.0	6.7	6.3	6.1	6.0	4.3	4.0	2.8	2.8
m	**w**	**f**	**y**	**g**	**p**	**b**	**v**	**k**	**x**	**j**	**q**	**z**
2.4	2.4	2.2	2.0	2.0	1.9	1.5	1.0	0.8	0.2	0.2	0.1	0.1

These frequencies are used to decrypt code like the Caesar Cypher. Although the letters are substituted the frequency remains the same.

For this problem we will add a monetary value to groups of letters on the basis that the least frequent letters are more expensive.

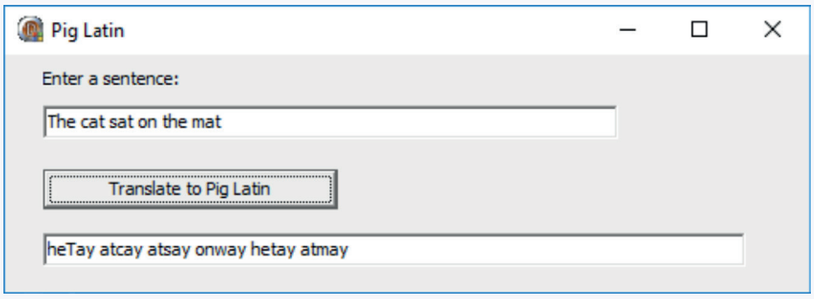
LETTER GROUP IN TABLE ABOVE	AVERAGE FREQUENCY	VALUE
(b, v, k, x, j, q, z)	0.6	R1666.67
(m, w, f, y, g, p)	2.2	R454.55
(s, h, r, d, l, u, c)	4.6	R222.22
(e, t, a, o, i, n)	8.5	R117.65

Open the **NameValue_p** project from the 08 – Name Value folder and complete the program.
The program must accept your full name and calculate the monetary value of your name according to the table above. Use lowercase letters.



10. Pig Latin is a secret language formed by moving the first letter of a word to the end if the letter is a consonant. The letters 'ay' is then added, for example, if the word is 'must', it becomes 'ustmay' – 'm' moved and 'ay' added. Vowels at the beginning are not moved to the end and 'way' is simply added at the end.

    Write a program that will translate a sentence into Pig Latin. Save the project as **PigLatin_p** in a folder named 08 – Pig Latin.



11. Open the **ScrollingBanners_p** project from the 08 – Scrolling Banners folder and complete the program. The program must allow the user to enter a message. The program must scroll the message as indicated below.

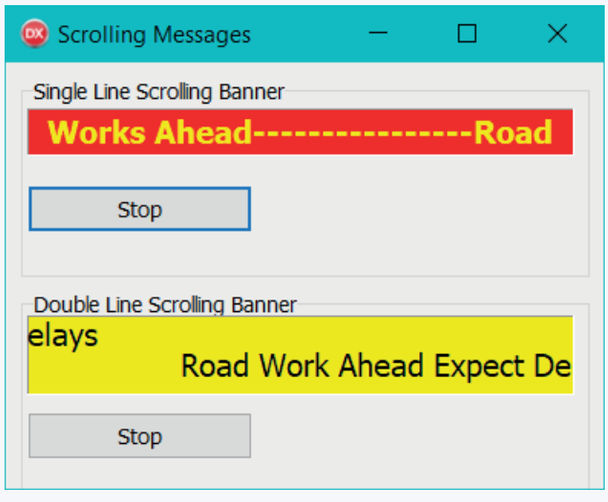
    a. Left to right, disappearing on the right and re-appearing on the left. For example:

`Road works ahead`	`d Road works ahea`	`ad Road works ahe`

    b. Exiting on the right of the top line and entering on the left of the line below, scrolling along the bottom line and exiting on the right, finally entering on the left of the top line. For example:

`Road works ahead`	`* Road works ahea`	`**********`	`d **********`
`**********`	`d **********`	`Road works ahead`	`* Road works ahea`

    Run the **ScrollingBanners_p.exe**, enter a message and click the [Start] button to start the scrolling banners.



TERM 4

CHAPTER 9

PAT PREPARATION

CHAPTER UNITS	
Unit 9.1	Tools and techniques to create a software solution to a problem
Unit 9.2	A problem solving approach
Unit 9.3	Analysing user interfaces

Learning outcomes

At the end of this chapter, you should be able to:

- explain what problem solving is
- understand how to approach a programming problem
- explain and apply problem solving steps and techniques to a given problem
- use the correct problem-solving tools.
- understand how to use the correct tools, principles and techniques to do the PAT

INTRODUCTION

In Grade 10 you need to complete a programming project called the Practical Assessment Task (PAT). The PAT is a software development project. You can use the PAT to demonstrate your programming skills and your understanding of the connections between the different content areas of solution development that you have learnt about.

You need to demonstrate your knowledge and understanding of the software development life cycle through analysis, design, coding and testing. You will also have to demonstrate the effective use of the software design tools and techniques which you have learnt.

You need to provide the following outputs:

- provide a brief description of the problem you will be solving
- discuss the research/investigation done regarding the project
- provide a brief description of the purpose and scope of your project
- a document in which the layout of your software design is shown
- a working Delphi program that implements the planned solution.

Take note

- The PAT is a compulsory component of the final end-of-year examination for IT.
- The PAT counts 25% of your final mark for IT. It is important that you produce work of a high standard.
- You need to complete your PAT before you start the Grade 10 end-of-year examination. Not submitting your PAT or any part of the PAT, will mean that you will be awarded a zero ("0") for the PAT component of the examination or the parts not submitted.

You will be required to demonstrate and discuss your program during a debriefing session.

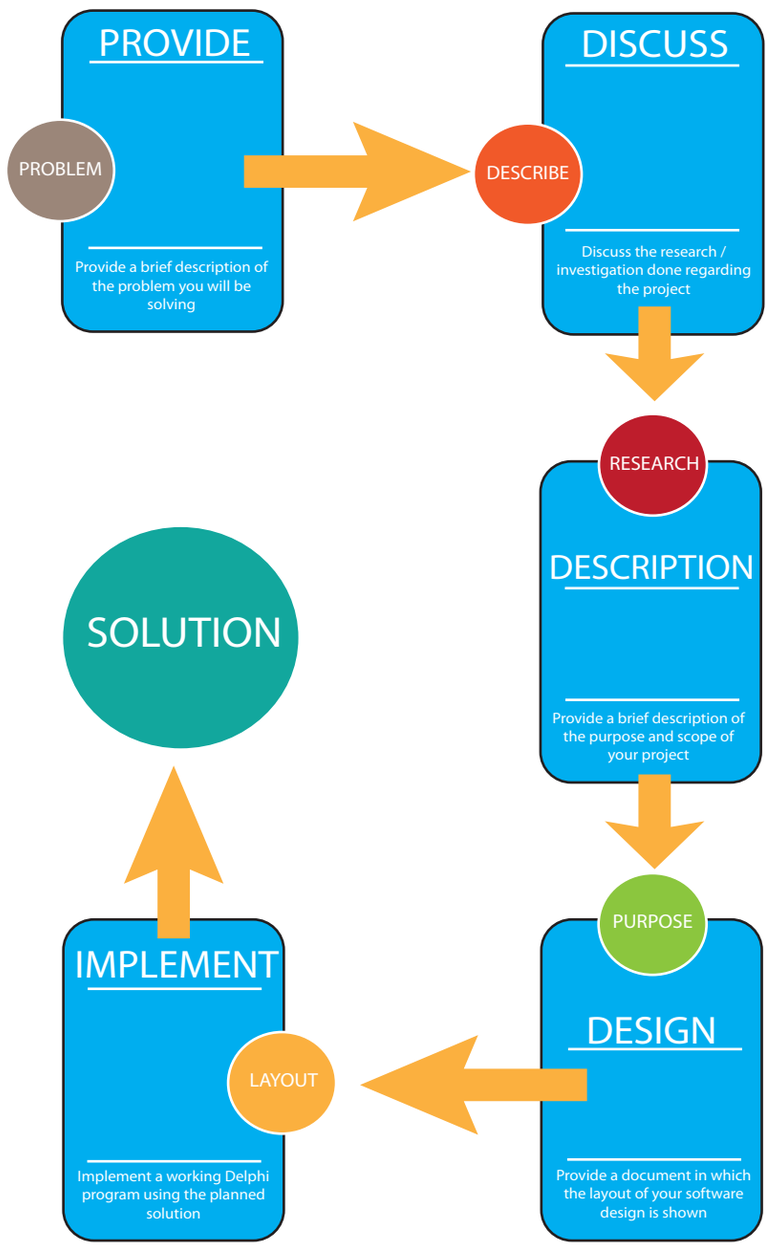


Figure 9.1: *Outputs for the effective use of software design tools and techniques*

In this chapter, you will learn how to approach the PAT. You will also learn some additional skills you could use to enhance your project.

UNIT 9.1

Tools and techniques to create a software solution to a problem

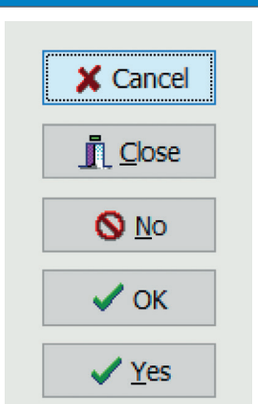
When you want to create a software solution to a problem, there are certain tools and techniques that support a professional product. There are also certain principles that you need to adhere to, to ensure that your solution (app) is user-friendly.

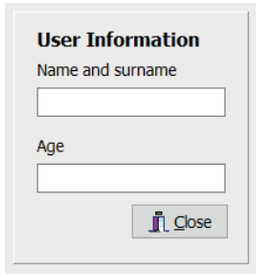
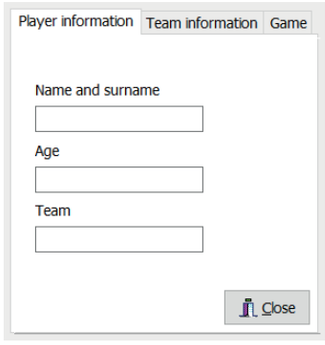
TOOLS AND TECHNIQUES

GUI COMPONENTS

To ensure that your solution has a professional, user friendly appearance, it is important to plan the graphical user interface (GUI) and to use the correct components. This year you have learnt to use the most important components that allowed you to create simple user interface.

We are now going to look at a few other components that would be helpful to enhance your project's user interface.

COMPONENT	DESCRIPTION	EXAMPLE
BitButton	Sometimes you want to use buttons such as, [OK], [Cancel] or [Help], for your GUI. To help you there is a button control that can include a bitmap image on its face. This feature is called a BitButton. You can choose from set styles or or you customise how the button will look. These buttons need very little or no coding at all, so you can easily add them to your application. To add a BitButton [Cancel]: • Place a TBitBtn component on the form from the Additional Palette • Set the Kind property to bkCancel • This will display:  • Double click on the button to create an OnClick event To create other BitButtons change the Kind property as follows: • bkCancel: A [Cancel] button. • bkClose: A [Close] button that closes a form. • bkNo: A [No] button. • bkOK: An [OK] button. • bkYes: A [Yes] button. These buttons are shown in the example on the right.	

COMPONENT	DESCRIPTION	EXAMPLE
Panel	The Panel component is used as a container for the other components. Components are grouped for a specific task. To place a panel on a form: • Select the TPanel component from the Standard Palette and place on the form • Use the prefix pnl before the panel name • You can resize the panel according to your own specifications • Place the components that you require on the panel An example of a panel requesting user information is shown on the right.	
PageControl	Instead of using panels to organise related information, you can use tabbed panes. To create a tabbed pane: • Select TPageControl from the Win32 panel and place on the form • Click the ellipse (…) in the Tabs property and list the tab headings on separate line in the String List Editor Dialog box • For each heading, a tab sheet will be created • Select a tab and place the required component on it The example on the right shows a PageControl being used to record information about a player, her team and her game.	
Hint property	Use the Hint property of a component to provide a short message when the cursor is moved over the component. The message should indicate what is required for that component. To create a hint on the TEdit component: • Select the TEdit component • Set the Hint property to the message that you want • Set the ShowHint property to true The example on the right shows an Edit with its Hint property set to the message 'Enter your name' and its ShowHint property set to true.	

STORING AND RETRIEVING DATA

To store data permanently, you need to store data in a text file on your computer. The kind of information you can store:

- A list of usernames and passwords
- Statistics for your games (such as high scores and win percentages)
- User's settings for your application.

If you want to store data permanently, then you need to save it to a file.

You have already learnt how to save data to files and read data from files. When saving you use the Delphi components that work with a list of strings (like the *TListBox* and *TMemo*). This is because these components have built-in methods that allow you to save their items directly to a text file, or to load data from a text file into the component. This is done using the following two commands.

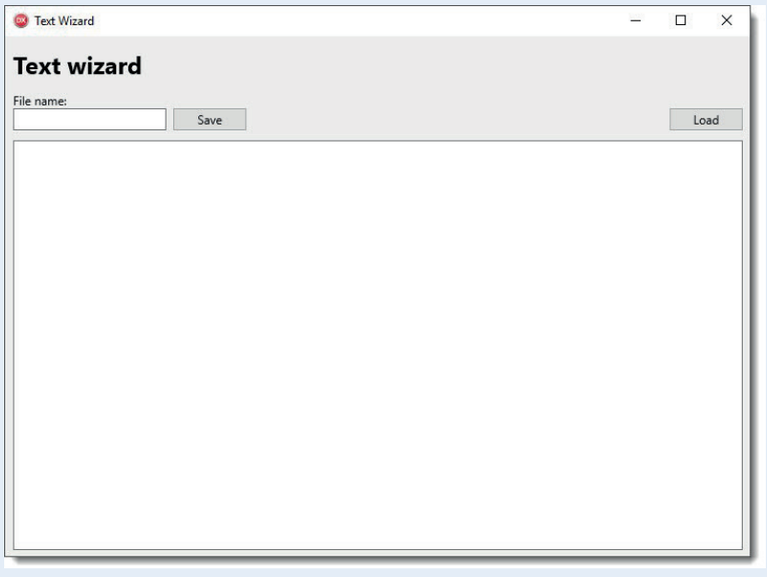
FUNCTION	DESCRIPTION
`memName.Lines.SaveToFile('file.txt');`	Saves the contents of a memo directly to a file.
`lbxName.Items.SaveToFile('file.txt');`	Saves the contents of a list box directly to a file.
`memName.Lines.LoadFromFile('file.txt');`	Loads the contents of a file into a memo.
`lbxName.Items.LoadFromFile('file.txt');`	Loads the contents of a file into a list box.

The example below shows how these commands can be used to create a simple text editor.

Example 1.1 Text Editor

To create a text editor:

1. Create a new project named **TextEditor_p** and save it in the folder 09 – Text Editor.
2. Create the following user interface. Take note, the large white box is a *TMemo* component.



Example 1.1 Text Editor *continued*

3. Create a global variable called sFileName.
4. Create an OnClick event for the [Save] button.
5. Inside the OnClick event, read the text from the edtFileName component and assign it to the sFileName variable.
6. Add the file extension .txt to sFileName

```
    sFileName := sFileName+'.txt';
```

Add the following statement to your event:

Save text from the memo component to a file

```
memText.Lines.SaveToFile(sFileName);
```

7. Run the program, enter the file name, type in the memo component and click on the [Save] button.
8. A text file with the file name that you specified with a .txt extension will be created. The data from the memo component will be saved in the text file.

To load a information from a textfile to a memo component:

9. Create an OnClick event for the [Load] button and add the following code to it.

Load from file method

```
sFileName := edtFileName.Text+'.txt';
memText.Lines.Clear;  //OR memText.Clear;  clears
                      //the memo component
memText.Lines.LoadFromFile(sFileName);
```

This will load the text from the specified filename into your memo component.

10. Save and test your application.

You should be able to create new files or open existing files by using the [Save] and [Load] buttons.

While the technique was used in this application to simply save and load text to a file, it could also be used with a listbox component.

PLANNING TOOLS AND TECHNIQUES

To make it easier to solve problems, there are four tools that you can use that help you with the different steps.

- IPO tables
- TOE (Task, Object (component), Event) chart
- Algorithms and flowcharts
- User stories
- Acceptance tests

In this section, you will learn more about each of these tools.

IPO TABLE

For Input, Processing and Output (IPO) tables refer to chapter 4.

TASK, OBJECT, EVENT (TOE) CHART

A Task Object Event (TOE) chart is a three-column chart that lists the tasks your application (or a part of your application) will do, the object involved with that task, and any events that the object might require to perform the task. The example below shows a TOE chart that could be used to create a smartphone login screen.

TASK	OBJECT	EVENT
Get user information		
User email address	edtEmail	
User password	edtPassword	
Verify user details and open application	btnLogin	OnClick
Provide password hint to user	btnHint	OnClick
Close application	btnClose	OnClick

Take note

You could also include TOE chart elements in your IPO table.

ALGORITHMS AND FLOWCHARTS

Refer to chapter 1 for algorithms and flowcharts.

USER STORIES

A user story is one or two sentences that are written by your product's users that explain what they want from your product. For example, your users might say something like "I need an application that allows me to quickly encode passwords on the Internet" or "The hardest part of being a teacher is setting up and marking quizzes."

After speaking to enough users, the programmer might have an entire list of user stories. These stories help the programmer to understand the problem and can be used to focus the programmer's attention on the most important features.

HOW TO WRITE GOOD USER STORIES

https://www.youtube.com/watch?v=tKSUokG3Y0w

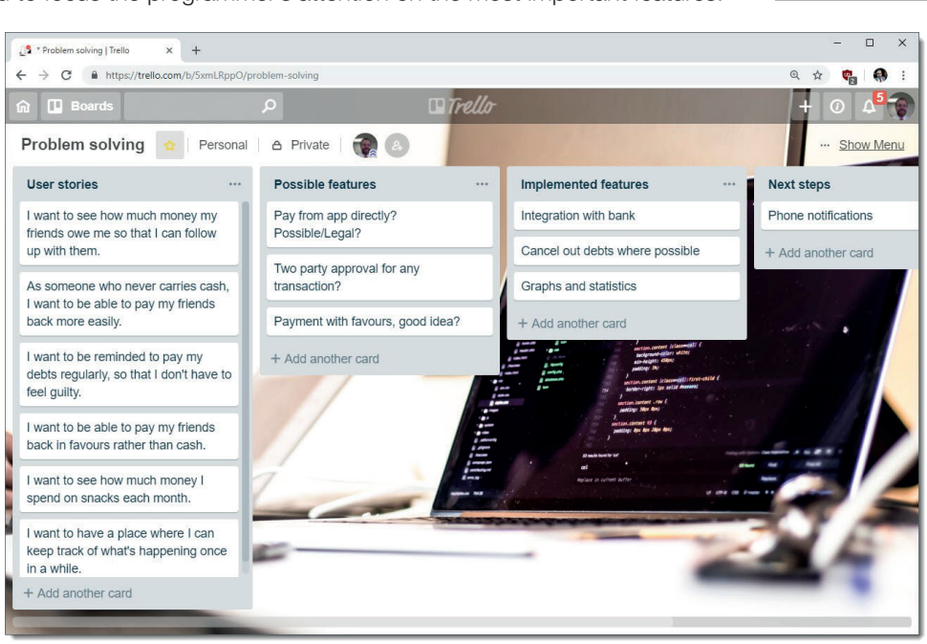


Figure 9.2: *A web application like Trello is good for recording user stories*

User stories help the programmer in a number of different ways:

- They make sure the programmer focuses on the user, not on his or her own goals.
- They make it easier to divide work between different people. With user stories, each person can take responsibility for one user's story.
- Since stories describe what users want, not how to achieve it, they allow programmers to come up with creative solutions to meet user's needs.
- User stories can help motivate programmers, since every complete story is tangible progress that has been made on the application, and there is at least one happy user.

For these reasons, most programmers should try to interview some of their users before starting to program their application. This can be as easy as asking your mother, your sister and a few of your friends questions about what they would want from your application and recording their answers.

ACCEPTANCE TESTS

Each user story must have an acceptance test. Acceptance tests are derived from user stories. In acceptance testing, the product is tested to see if it solves the problem and meets the project's requirements. The program either passes the test or fails. This determines if the program is ready to be officially released and sold to customers, or if the program needs further development.

PROGRAMMING PRINCIPLES

NAMING CONVENTIONS

Use the naming convention used throughout the book.

DECLARATION OF VARIABLES

In Chapter 2 you learnt how to declare variables both locally and globally.

DATA VALIDATION

You learnt about data validation in Chapter 6.

GUI DESIGN PRINCIPLES

The user interacts with the program though the GUI. This includes the screens the user sees, the buttons they press, the information they enter, the feedback they receive, etc. The user experience is therefore important – it could determine whether they will use the program or not.

PUT USERS IN CONTROL

- Allow them to correct mistakes or undo actions
- Make the user interface easy to use and master
- The user interface must be predictable for example always put the [Submit] button in the same place on every screen
- Provide tool tips, hints or help for users
- Accommodate visually challenged users

MINIMISE THE EFFORT

- Minimise the effort it takes to use the program
- Never ask user to enter the same information more than once
- Never ask users to complete tasks that do not benefit them directly
- Use words and terms that are familiar with users
- Use visual cues to make the next step obvious
- Provide feedback to users

REDUCE COGNITIVE LOAD

- Cognitive load is the amount of mental processing required to use a product
- Chunking the sequence of information – that is information is input in some chunks. For example, instead of inputting your cellphone number as a 10-digit sequence 0842123256, get the user to input the cellphone number in the format 084 212 2356 using an input mask
- Reduce the number of actions for a required task
- Promote recognition over recall
- Provide visual cues

MAKE USER INTERFACE CONSISTENT

- User interface must be consistent, that is, if you have more than one screen the look and feel of the screens must be consistent. For example components should remain in the same place, be of the same size and have the same colour/icons across screens.
- Buttons must work in the same way in one screen as in the other screens for the same task. Generally, by convention, certain buttons are placed in certain position on a screen. For example the [Forward] and [Backward] button are placed at the bottom of the screen with the [Forward] button on the right and the [Backward] button on the left.

UNIT

9.2 A problem-solving approach

Now that you understand what tools and techniques contributes, you can continue exploring a problem-solving approach that can be used when you complete your PAT.

Problem solving refers to the process through which a solution is found to a problem. Programmers spend most of their time solving problems, whether it is a small technical problem like a program bug or developing a software solution for a specific use. For example: Uber is attempting to provide an affordable but flexible transport solution to people without a car.

In this unit, you will learn more about the techniques used by programmers to solve problems.

PROBLEM SOLVING STEPS

In IT, problem solving consists of the following steps:

- Understanding the problem
- Developing a plan
- Implementing the plan
- Evaluating the solution
- Repeating the process

Let's look at each of these steps in more detail.

The following sections demonstrates how the above steps are used to develop the IT PAT.

PROBLEM

You need to develop a program that encrypts messages using a cipher and decrypts messages using the same cipher. For example, if you want to send an encrypted message to a user using your application, that user must be able to use the program to decrypt the message.

You could also have a cipher game where others are challenged to crack the ciphers.

The program needs to use at least two different ciphers:

- Use at least one existing cipher (see list of examples below)
- Use at least one cipher that you developed yourself

Here are some examples of existing ciphers for encrypting/decrypting messages:

- Various examples: http://www.simonsingh.net/The_Black_Chamber/chamberguide.html
- Vigenere Cipher: http://sharkysoft.com/misc/vigenere/
- RSA Cipher: http://cisnet.baruch.cuny.edu/holowczak/classes/9444/rsademo/
- Keyword Cipher: http://www.secretcodebreaker.com/keyword.html
- The hobby and art of cryptanalysis – that is, learning to break ciphers, see http://cryptogram.org/
- Cracking ciphers: http://cryptogram.org/solve_cipher.html or http://simonsingh.net/cryptography/cipher-challenge/the-ciphertexts/

Here are some ideas for developing your own cipher. Using:

- Binary numbers
- ASCII codes /symbols
- Standardised numbers such as ID numbers, ISBN numbers
- Calculations
- Mathematical processes such as check digits, LCM, etc.
- String processes
- Combining aspects of existing ciphers, etc.

UNDERSTANDING THE PROBLEM

The first step of the problem solving method is to understand the problem. To do this, you first need to determine what the problem is about, then you need to research the problem.

The following will help you manage your understanding of a problem:

- write down the main ideas and requirements of the problem
- read the problem and underline the key concept you need to understand in order to interpret the problem clearly
- research what you do not understand.

Example:

~~You are one of the learners at IT Academy~~. The pass rate at this Academy is 60%. Write a program to display a message "YOU HAVE PASSED" if your mark is 60% or more.

DEVELOPING A PLAN

You spend hours creating an implementation plan, which might feel like a waste of time, however in the long run it pays off when you start creating your application. Rather than struggling to create a massive program that you do not understand, the implementation plan breaks it into small, manageable tasks that can be completed systematically. With each completed task and line of code that you write, you are one step closer to having a working product.

REPRESENT THE PROBLEM

Once you develop your plan, you can represent the problem using diagrams, tables, flow charts, descriptions or any other method to indicate how you understand the problem.

For example: You want to develop a simple educational program for Grade 3 learners that will test how they are progressing in Mathematics and if they are ready to progress to the next level (Questions can be like the ANA-type questions set by the Department of Education). The application must give a specific user feedback on how they are progressing with the content, the percentage of quizzes they have completed, how many questions they have gotten right/wrong, and possible steps to help them improve their scores in tests and quizzes.

PLAN DETAILED STEPS FOR EACH SCREEN

When you have a clear understanding of the functionalities that the program need to provide, you can then determine the purpose of each screen. From there you are able to plan each screen using a TOE chart, then plan the IPO table for each screen. Finally start work on your application.

IMPLEMENTING THE PLAN

Once complete, you will now be ready to implement your plan. You can do this by:

- coding each screen in Delphi according to your planning.
- determining which code/constructs you will use to input data.
- determine which statements you will use to process or calculate the data.
- determine which statements you will use to display the output.
- finally, compile the program, test it and correct any errors that may occur during run-time.

TESTING THE PROGRAM

Once the program has been created, you need to test the program with extreme cases of test data. Repeat this process until the program runs correctly and provides the desired output. The problem-solving method does not end once you have implemented your plan! You need to see if the plan has actually solved the problem. Look back to your initial problem statement and discussions with the users and compare your solution to their requirements.

EVALUATE THE SOLUTION (REFLECTING)

Always review your work by looking back to see how well you have solved the initial stated problem. Re-read your original problem statement and determine if you have reached the goal. Evaluate whether you need to make any changes to the program, and if necessary, implement your changes and test your results.

REPEAT THE PROCESS

Not all solutions solve the problem. It is also possible for solutions to:

- Fail to solve the problem.
- Only partially solve the problem.
- Create new problems.

If your evaluation reveals any remaining problems, you can repeat the problem-solving process, focusing on the new problems.

UNIT

9.3 Analysing user interfaces

According to the principles of rapid application development, you should start with the user interface. By the time you have placed all the elements, boxes and buttons for your application, the only thing left to do is to code the events. Since there are only a limited number of buttons and interactive elements, there can only be a limited number of events, which means you can systematically create small parts of your program, without worrying too much about the big picture.

How do you take the first step and build a user interface? A good way to do this is to analyse existing user interfaces, especially for applications that are similar to your application, and use their user interfaces as a starting point. These user interfaces can be analysed in four steps:

- Take a picture of the screens of the user interface
- Break the screens down into their components
- Record how the user interacts with the application
- Draw a map showing how the user can navigate between screens

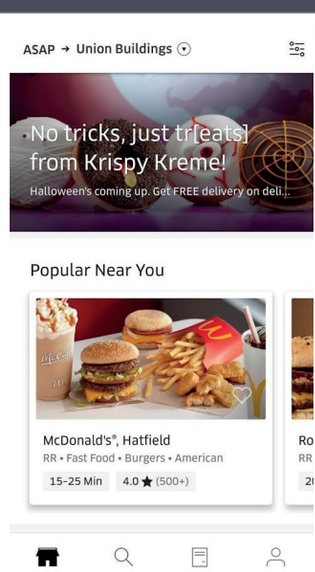
By analysing existing user interfaces, you will be surprised to see how many small features application developers add to make their programs easier to use. You will also realise that many UIs that look extremely complex are actually created using very simple components.

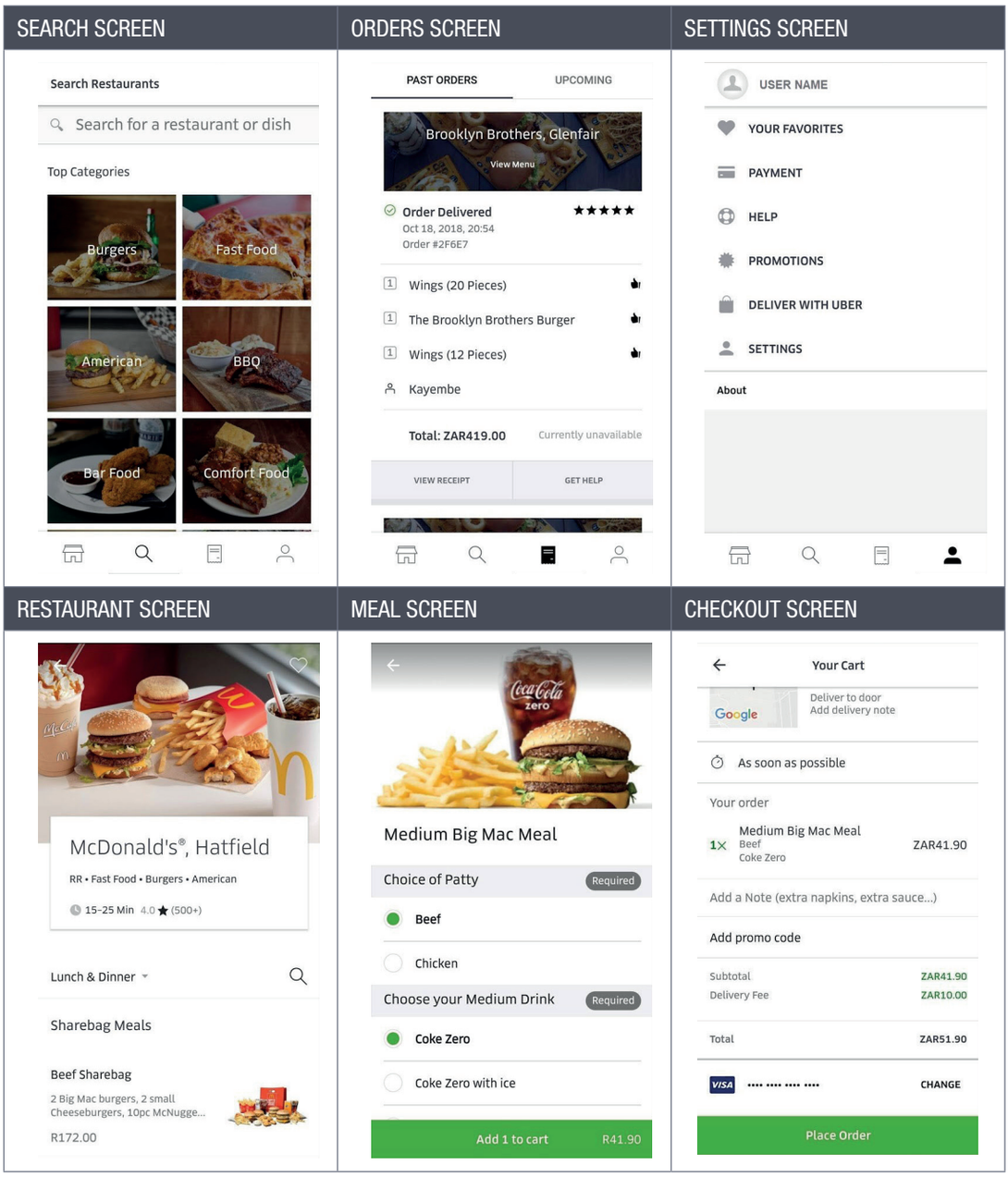
In the next section, you will use the four steps listed above to analyse an existing user interface.

STEP 1: RECORD THE SCREENS

For this example, we will analyse the smartphone user interface of the application UberEats. UberEats allows users to order food from nearby restaurants and have it delivered to their homes.

The first step in analysing the application is to record the different screens used by UberEats. The images below show nine screens from the UberEats app.

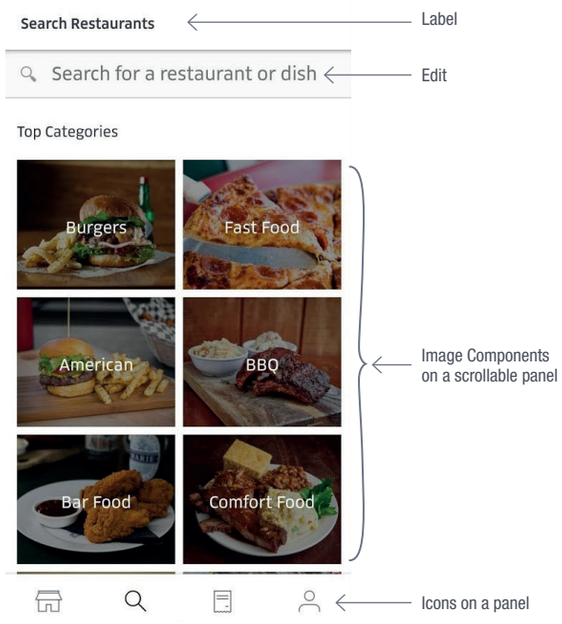
SPLASH SCREEN	LOGIN SCREEN	HOME SCREEN
		



By analysing these screens, you can learn how they are created.

STEP 2: BREAK THE SCREENS DOWN

In most cases, it is not worth breaking all the screens into their individual components. Instead, you may choose one or two screens to analyse. For this example, we will analyse the Search screen, as a similar screen can be used for many different applications.



Broken down like this, the screen is not too difficult to recreate. In fact, you know how to create almost all the elements on the screen already! Creating a similar screen for your own user interface is therefore simply a matter of placing all the components in the correct positions.

STEP 3: RECORD THE INTERACTION

The third step is to record how the program works and what happens when you click on the different buttons. While it is difficult to say what precisely is happening in the code, you can try to guess how the main elements work.

For example, looking at the *Search* screen from the previous step, there are three ways for users to interact with this screen:

- They can enter a search query at the top of the screen
- They can click on one of the category images
- They can navigate to a different screen using the panel at the bottom.

For the search textbox, you will need to create a search algorithm.

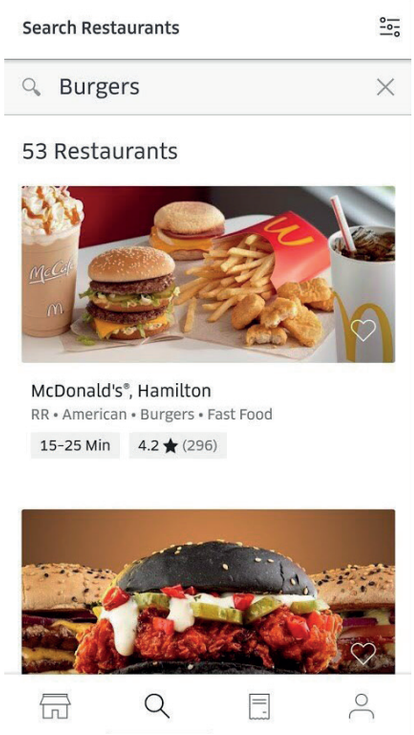


Figure 9.3: *The Search Results page*

What happens when users click on one of the categories? In UberEats, the program simply searches for the category name, returning the same results as if the user entered the category name. This means the code for these buttons is easy to create, since it can reuse the search algorithm code created already.

When the users click on one of the navigation buttons at the bottom of the screen, the program simply swaps to the selected screen. Once these screens have been created, coding the buttons so that they swap to the correct screen is easy!

Did you know

You will learn more about working with databases in Grade 11.

STEP 4: MAP THE NAVIGATION

Once you have recorded how the interaction works on the important pages, you can map how users swap between different screens.

The UberEats application is a complex application with more than 20 different screens and almost all the screens linking to the other screens. However, a simplified view of the screens can be drawn as shown below.

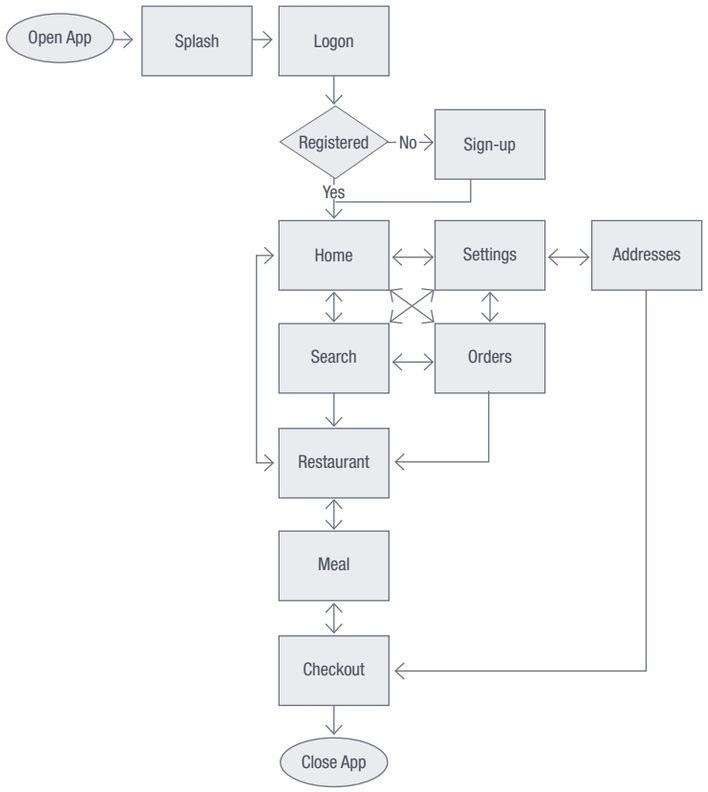
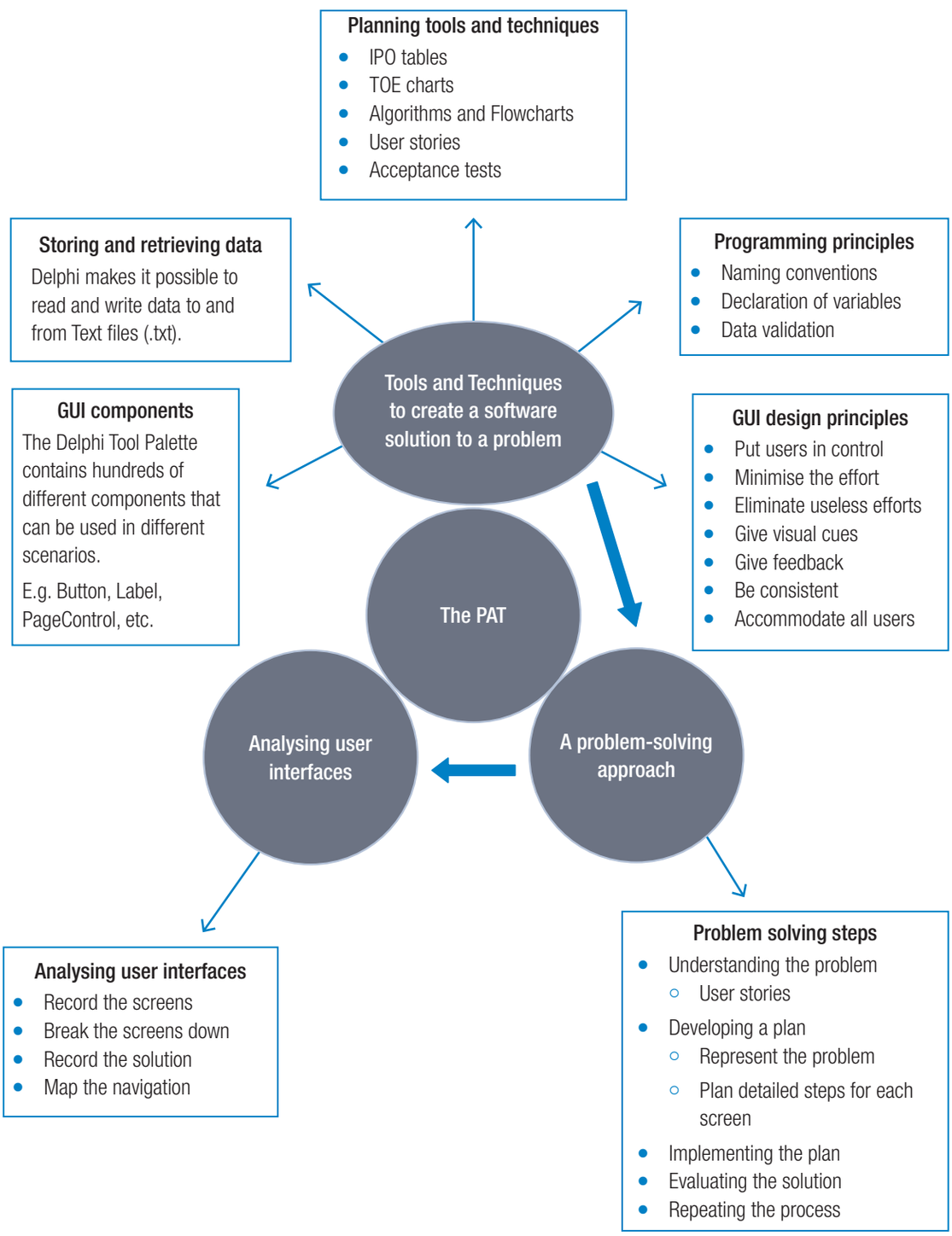


Figure 9.4: *Simplified flow chart for the navigation of UberEats*

Now that you understand how the UberEats user interface works, you can use the relevant concepts in your own programs. The more user interfaces you analyse, the more ideas you will have for your own programs!

PAT PREPARATION

YOUR SOFTWARE DEVELOPMENT PROJECT



ANNEXURE A

Installing the Delphi IDE

Different programming languages use different applications to create programs. Some languages can be programmed using a simple text file, others can be programmed from your web browser, while a third group of languages require an integrated development environment (or IDE) to create code. Delphi falls in the third group and in this book, we will use an IDE called Embarcadero RAD Studio to create programs. This means that, just like you would open Microsoft Excel to create a new spreadsheet, you will open Embarcadero RAD Studio whenever you want to create a Delphi application.

Embarcadero RAD Studio helps you to create new applications more easily by:

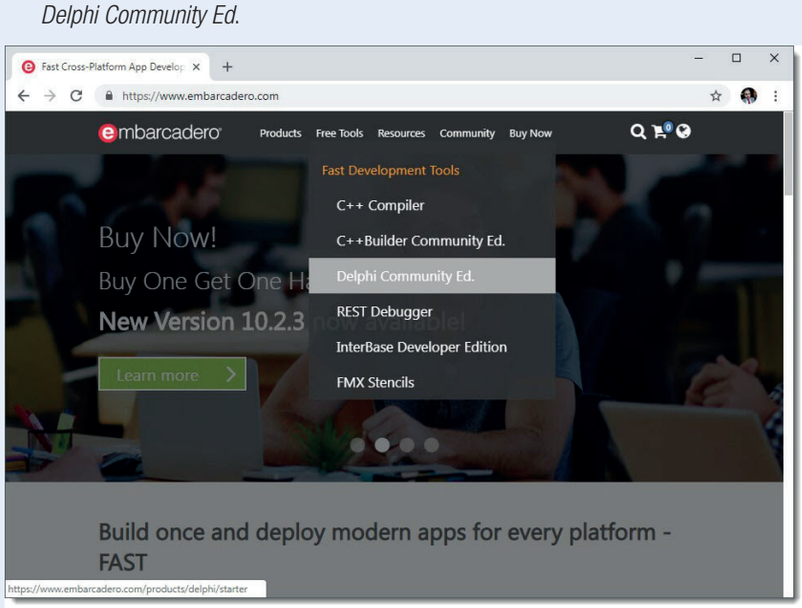
- Converting (or **compiling**) the lines of code into a usable program
- Giving you the tools needed to create a user interface
- Helping you to refer to the correct names and properties
- Highlight **syntax** elements to make the code easier to read
- Highlighting and fixing code mistakes
- Allowing you to **debug** your program

This saves you time and effort when creating programs and allows you to quickly create new programs. The next example will show you how to install RAD Studio on your computer.

Example Downloading the Delphi IDE

To download the Delphi IDE:

1. Open your computer's web browser.
2. Open Embarcadero's website at www.embarcadero.com
3. At the top of the window, hover over the *Free Tools* menu and select *Delphi Community Ed.*



4. Click on the [Get Community Edition Free] button.

New words

compiling – to convert (a program) into a machine-code or lower-level form in which the program can be executed

syntax – the language rules used by the programming language

debug – to identify and remove errors from computer hardware or software

Take note

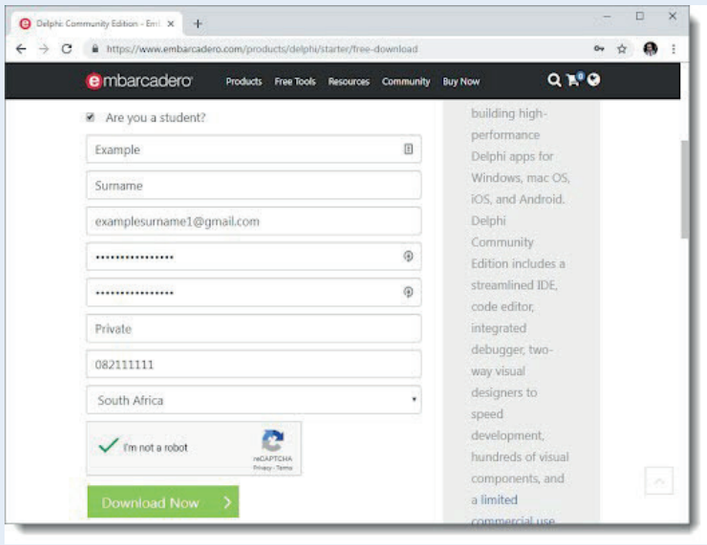
The exact layout of the IDE may change over time, so follow the prompts on the screen if the layout is different to what is being shown here.

Did you know

You must use a valid email address when registering. A unique serial number will be sent to your email address, and you will not be able to install the RAD Studio without this serial number.

Example Downloading the Delphi IDE *continued*

5. Click on the [Register Here] button to create an Embarcadero account.
6. Fill in the questionnaire with your personal details. If you do not have your own email account, follow the QR code in the margin to learn how to create one.



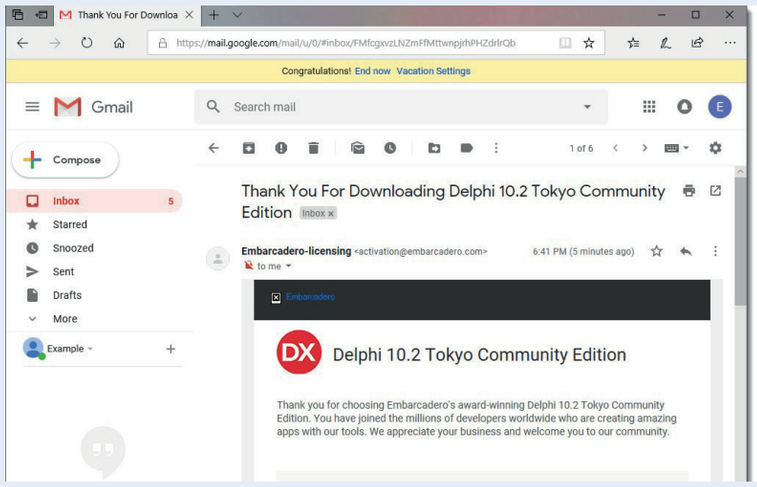
7. Click on the [Download Now] button to download the RAD Studio install file.

Congratulations, you have just downloaded the RAD Studio installer! You are now ready to begin installing RAD Studio.

Example Installing the Delphi IDE

To install the Delphi IDE:

1. Open the RAD Studio installer you downloaded in the previous example.
2. Tick the checkbox to agree with the RAD Studio license agreement and privacy policy then click *Next*.
3. Select the option I already have a product serial number then click Install.
4. While the program installs, open your email application and open the email from *Embarcadero-licensing*.

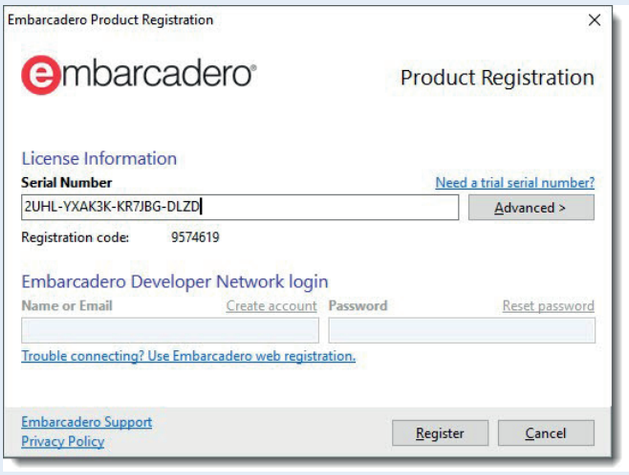


Take note

The exact layout may change over time, so follow the prompts on the screen if the layout is different to what is being shown here.

Example Installing the Delphi IDE *continued*

5. Select the serial number from the email and press CTRL + C to copy it to your clipboard.
6. When the *Embarcadero Product Registration* window opens, select the *Serial Number* textbox and press CTRL + V on your keyboard. You should now see the serial number in the textbox.



7. Click on the [Register] button to continue the installation.
8. In the *RX RAD Studio Platform Selection* window, select *Delphi Windows 32-bit Community* and *Delphi Android Community* and click *Continue*.

9. Click on the [Install] button to begin downloading and installing the application. Once the installation is complete, you can start using RAD Studio!

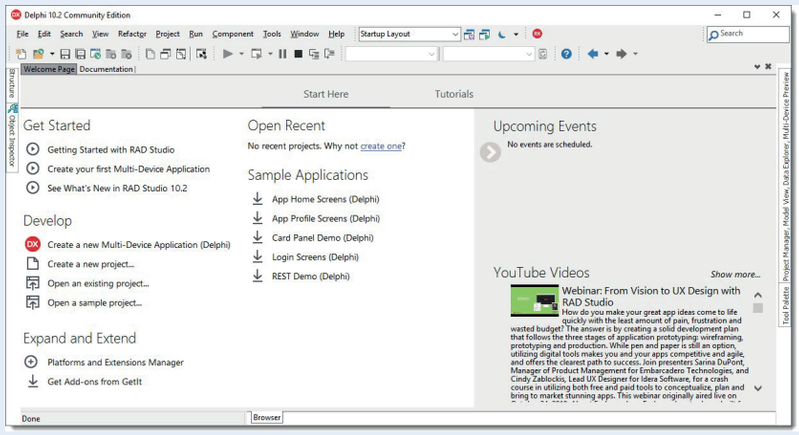
Congratulations, you have just installed Embarcadero's RAD Studio! For the rest of the year, you will use this application to code programs and games!

Example Opening the Delphi IDE

To open the Delphi IDE:

1. Open the Windows *Start Menu* and search for "RAD Studio".
2. Open the *RAD Studio* desktop application from the *Start Menu*.
3. Select whether you want to use a light theme (with a light background and dark text) or a dark theme (with a dark background and light text). You can always change your theme at a later stage.

4. Click on the [Get started!] button to open RAD Studio.



Congratulations, you are ready to start programming!

Did you know

Most programmers prefer to use dark themes when coding. However, to make the pictures easier to see, this textbook will use a light theme.

ANNEXURE B The ASCII table

In 1963, the American Standards Association published a table which linked 127 different letters and symbols to numbers. This table was called the **ASCII** table, which is short for the American Standard Code for Information Interchange.

With ASCII, the first 32 characters in the table are programming characters that cannot be shown on the screen. These include characters like a carriage return character (which shows where a new line should start) and a horizontal tab character which added some horizontal space. The full list of these 32 programming characters is given in the table below.

Table 10.1: *The programming characters*

DECIMAL NUMBER	CHARACTER	NAME
0	NUL	Null
1	SOH	Start of Heading
2	STX	Start of Text
3	ETX	End of Text
4	EOT	End of Transmission
5	ENQ	Enquiry
6	ACK	Acknowledgement
7	BEL	Bell
8	BS	Backspace
9	HT	Horizontal Tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out
15	SI	Shift In

DECIMAL NUMBER	CHARACTER	NAME
16	DLE	Data Link Escape
17	DC1	Device Control 1
18	DC2	Device Control 2
19	DC3	Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowledgement
22	SYN	Synchronous Idle
23	ETB	End of Transmission Block
24	CAN	Cancel
25	EM	End of Medium
26	SUB	Substitute
27	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator

The next 95 characters are all visible characters that you can see on the screen.

Table 10.2: *Visible characters*

DECIMAL	CHARACTER
32	SPACE
33	!
34	“
35	#
36	$
37	%
38	&
39	‘
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<

DECIMAL	CHARACTER
61	=
62	>
63	?
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y

DECIMAL	CHARACTER
90	Z
91	[
92	\
93	]
94	^
95	_
96	@
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v

DECIMAL	CHARACTER
119	w
120	x
121	y
122	z
123	{
124	|
125	|
126	~

The final 127th character is the DELETE character, which is used when something needs to be removed or deleted.

ANNEXURE C

Naming conventions of components

COMPONENT NAME	PREFIX	ICON	BRIEF DESCRIPTION
Standard Group			
Button	btn		Most used to activate an action.
Label	lbl		Commonly used to display information.
Edit	edt		Used for single line input, but also displays information.
Memo	mem		* Multiple line display organized in lines.
Panel	pnl		A container hosting other components.
List Box	lst		Multiple line display. Able to display left aligned columns.
Radio Button	rad		Toggles selection.
Radio Group	rgp		Grouped Radio buttons – only one selectable.
Combo Box	cmb		Multiple line capturing. Selection of item through drop-down.
Check Box	chk		Toggles selection.
Additional Group			
BitButton	btt		Button with icon - used to activate actions.
Image	img		Component to host pictures (bitmaps, jpgs).
Shape	shp		* Basic shape like circle, rectangle or ellipse.
Win32			
Page Control	pgc		Special page component hosting tab sheets.
System Group			
Timer	tmr		* Count down timer – initializing action as count-down reaches 0.
Samples Group			
Spin Edit	sed		Integer input component with pre-set range to select from.
Data Controls Group			
Form	frm		The initial Form – not categorised under any group.

Glossary

A

algorithm an ordered list of steps used to accomplish a task or solve a problem

algorithmic structures different structures and techniques that you can use to improve and simplify an algorithm

array an array is a sequence of data items of the same type. It is made up of a list of other variables, such as strings or integers

B

BODMAS Brackets, Orders, Division, Multiplication, Addition and Subtraction

Boolean variable contains only one of two values: TRUE or FALSE

C

compiling converting (a program) into a machine-code or lower-level form in which the program can be executed

concise giving a lot of information clearly and in a few words

conditionals tell a program to execute different actions depending on whether a condition is true or false

D

debug to identify and remove errors from computer hardware or software

double variable contains any positive and/or negative numbers (including decimal numbers)

E

event-driven an event-driven program is one that largely responds to user events or other similar input

exponentially more and more rapidly

F

factor any number that can be divided into another number (the multiple) without leaving a remainder

feedback whenever a user interacts with an application, they should receive feedback that acknowledges the interaction and informs them of what's happening

file a container in a computer system for storing information

flow chart a tool that is used to show visually how an algorithm works

H

hard coded data or parameters fixed in a program in such a way that they cannot be changed without modifying the program

hotkey one or more keys used to perform a menu function or other common functions in an application

I

implementation plan step-by-step guide, describing how to solve the problem

input validation a technique used by programmers to check the user's input before processing it

integer contains any positive or negative whole number (i.e. a number without a decimal)

integrated development environment (or IDE) a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger

iteration repetition

L

list allows you to store a large number of elements which can be accessed using an index and which must all be of the same type

logic error these errors occur when there is a logical error or design problem in your program

loop a sequence of an instruction that is continually repeated until a certain condition is reached

M

mathematical operators the symbols used to tell Delphi to add, subtract, multiply or divide two numbers

method (Delphi) a method is a function that is programmed into an object (such as a label or a string) by the creators of Delphi

mnemonic a tool to help remember facts or a large amount of information

N

noun–verb analysis a planning technique used to analyse a problem statement

O

object-oriented a software programming model constructed around objects

P

problem solving problem solving refers to the process through which a solution is found to a complex problem

problem statement a concise description of an issue to be addressed or a condition to be improved upon

R

rapid application development (RAD) a programming system that enables programmers to build working programs quickly

runtime error this occurs when you ask your program to do a task that is either impossible or is impossible under certain circumstances

S

string variable made up of a sequence of numbers, letters and symbols

syntax refers to the specific rules of a language

syntax error occurs when you break the rules of the programming language

T

toggle to swap between two stages (ON/OFF)

trace table a technique that can be used to test an algorithm. It helps you to find out if your answer is correct

U

user interface (or UI) refers to the way in which users interact with a computer

V

variable a value that can change, depending on conditions or on information passed to the program

variable scope the scope of a variable refers to the sections of a program in which that variable is available

visual cues a visual signal and reminder of something. It helps users to understand where they are in the application, what they are doing, how the application works and what is expected of them next

W

work in place algorithms that do all their calculations without taking up any additional space in memory

work out-of-place algorithms that need to make copies of the items they are working with in memory

QR Code list

You can use the QR codes on these pages to link to online content for further information on these topics.