

Development Software IA (DSO17AT)

Basic Programming Principles

**Compiled by
HG Erasmus
CM Pretorius**

**Adapted by: MG Meintjes, TS Raphiri , T
Malebane & SF Kgoete**

**© COPYRIGHT: Tshwane University of Technology (2020)
Private Bag X680
PRETORIA
0001**

All rights reserved. Apart from any reasonable quotations for the purposes of research criticism or review as permitted under the Copyright Act, no part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy and recording, without permission in writing from the publisher.

Chapter 1

HOW TO UNDERSTAND PROBLEMS AND SOLVE THEM USING A COMPUTER

1.1 Introduction

In this chapter you will learn how to analyse a problem and plan a solution using a five-step problem-solving approach.

1.2 Problem Solving

As humans we can solve different types of problems, often not consciously going through any formal steps. We simply execute a number of steps to come to the solution. For instance, you arrive home after your first day at school with a number of new books to that you have to cover with brown paper and plastic. You automatically go through the process to solve this problem:

- Take the first book
- Cut the brown paper to the correct size
- Cover the book
- Write the correct label for this subject
- Paste the label onto the book
- Cut the plastic to the correct size
- Cover the book with plastic
- Repeat this process until all the books are covered

This is an everyday problem, but here we are going to learn how to solve a problem using a computer. The computer is just a tool that can only do what you as the programmer instructs it to do in order to solve the problem! You will have to write down and feed the computer with all the steps that must be followed to solve the problem. Because the computer is only a dumb machine, the instructions will have to be very precise, unambiguous and in the correct sequence. So it is a person, called a programmer, who must first work out the steps to solve the problem and then translate those steps into a computer language before the computer can execute the steps to produce the solution. These detail steps that solves a problem is called an **algorithm**.

You must properly design an effective solution before going over to the coding of the solution in a specific programming language. Take time to plan the logic (reason out the steps of the program) before you begin coding. Once you have planned the logic, work through it systematically to make sure it solves the problem correctly.

1.3 Data Processing

The following problem can be solved by using a computer.

Problem: Determine the total amount payable if the price of an item and the number of those items to be bought is known.

Algorithm to solve the problem:

1. display "What is the price of the item"?
2. enter the price
3. display "How many of these items is bought"?
4. enter the number of items
5. calculate Total amount payable: price X number of items
6. display the amount payable on the screen of the computer.

When studying this algorithm we note quite a number of things:

1. Sequence of steps:
 - Is it possible to exchange steps 2 and 5?
 - What about exchanging steps 1 & 2 with steps 3 & 4?
 - Is it possible to display the answer on the screen of the computer before it has been calculated?
2. We see that there is a conversation between the user (e.g. you or the person using the computer) and the computer. The computer "talks" to the user in steps 1, 3 and 6, while the user replies in steps 2 and 4.

When the user "talks" to the computer, it is called **input**. The user supplies something to the computer.

Input identified in the algorithm in steps 2 and 4:

- Price
- Number of items

When the computer "talks" to the user, it is called **output**. The computer provides the user with questions or answers as in the above algorithm.

Output identified:

- The questions in steps 1 and 3
- The amount payable displayed in step 6 (result)

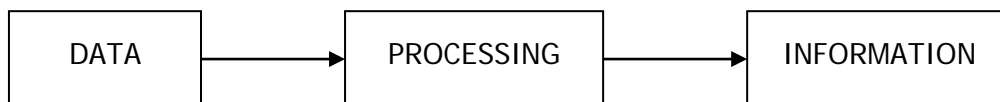
3. Step 5 is not input or output. A calculation is done to provide an answer. This step is called **processing**.

The following diagram shows the relationship between input, output and processing:



In terms of programming, we can also look at the INPUT as ***data***. ***Data*** is information in raw or unorganized form (such as alphabets, numbers, or symbols) that are input, stored, and processed by a computer, to produce OUTPUT as usable ***information***.

Data can now be seen as raw facts, while ***information*** is processed, useful data, as seen in the following diagram:



Exercises:

Study the following problems and identify the input and output and then write an algorithm in pseudocode to solve the problem:

1. Get the length and width of a suburban stand to be fenced. Determine and show the length of fencing needed to put around the stand.
2. Thandi borrowed a number of books from the local library, but has forgotten to return the books on time. She will supply you with the number of books as well as the number of weeks that her books are late. She has to pay a fine of R1.50 per week per book. Show on the screen of the computer how much she will have to pay when she returns the books.

1.4 Understanding the Problem

When a student starts reading these notes and wants to learn how to solve a problem, it is important to understand what the problem is.

So, the first steps are to

- read the problem carefully
- understand what the problem entails
- and only then, commence to solve the problem.

Let us start by looking at a problem:

Question: How many birth dates do you have?

When reading this question it sounds very simple and you will not waste time to read it for a second time, because you know your age is 18 years, so the answer is 18. Very simple!

Read the question once more carefully and you will see that the question does not ask you "How many times have you celebrated your birthday?", but refers to the number of times you were born! The only correct answer is 1.

Next question: Do you know what the time is?

As soon as you have heard this question, you will look on your watch and reply: "It is 11 o'clock". If you are not wearing a watch, you may answer that you do not know what the time is because you are not wearing a watch.

But, the question did not ask "What is the time?", it only wanted to know if you know what the time is!! The only answer to this question is either a "Yes" or a "No".

Next question: Name the months in a year that have 28 days.

Another easy question with a straight answer - of course the answer is February!

What about January? Doesn't it have 28 days as well? You have not read the question carefully and did not think about it. You just jumped to a conclusion before spending enough time and effort to not only read the question, but to understand it as well. The answer to this question is all 12 months of a year have 28 days and now you can name them.

You have to understand that a computer cannot think for itself, but you, the programmer must instruct the computer how to solve the problem. As you can see, it is of the utmost importance that the programmer must be able to **read** and **understand** the problem *before* instructing the computer.

Reading a Problem

1. Identify redundant information:

Consider the following problem:

The FRESH GREENGROCER sells apples at 85 cents each, pears at 65 cents each and oranges at 90 cents each. Tumi has R15 and wants to buy 10 apples and 5 pears. Calculate the amount due.

To be able to calculate the amount due, she only needs to know how many apples must be bought at what price and how many pears at what price. The information about the oranges and the amount Tumi has available is certainly not needed.

Strikeout all the redundant information and rewrite the problem:

The FRESH GREENGROCER sells apples at 85 cents each, pears at 65 cents each ~~and oranges at 90 cents each~~. Tumi ~~has R15 and~~ wants to buy 10 apples and 5 pears. Calculate the amount due.

Without all the unnecessary information the problem becomes:

The FRESH GREENGROCER sells apples at 85 cents each and pears at 65 cents each. Tumi wants to buy 10 apples and 5 pears. Calculate the amount due.

You will find that it is much easier to read and understand what is needed.

The second example:

The STATIONARY SHOPPE has 50 blue pens, 85 red pens, 47 green pens and 32 black pens in stock. The price of blue and red pens is R2.40 each, a black pen costs R2.35 and the price of a green pen is R2.20. John buys 50% of the blue pens and the same number of black pens. John has R150 available. Calculate how much change he would receive.

Referring to the previous example, we need to know how much money John has in his pocket to be able to calculate the change he will receive. In this question only blue and black pens are sold. So, we do not need the number and price of the red and green pens.

Only focus on the facts needed to solve the problem and cross out the unnecessary words:

The STATIONARY SHOPPE has 50 blue pens, ~~85 red pens, 47 green pens~~ and 32 black pens in stock. The price of blue ~~and red pens~~ is R2.40 each, a black pen costs R2.35 ~~and the price of a green pen is R2.20~~. John buys 50% of the blue pens and the same number of black pens. John has R150 available. Calculate how much change he would receive.

Without all the unnecessary information to confuse you, the problem becomes:

The STATIONARY SHOPPE has 50 blue and 32 black pens in stock. The price of a blue pen is R2.40 and a black pen costs R2.35. John buys 50% of the blue pens and the same number of black pens. John has R150 available. Calculate how much change he would receive.

Exercises

Identify redundant information that is not needed to solve the problem in each of the following cases and rewrite the problem including only the necessary details:

1. Tshepo who is 16 years old sold all but 7 of his chickens to a street vendor. The street vendor sells the chickens for R18.50 each. The vendor had 10 chickens before the transaction. After the transaction, she has one-and-a-half the number of chickens she had before the transaction. How many chickens did Tshepo have before he sold the chickens to the vendor?

2. A factory employs 150 workers, who starts working in the morning at 07:00 and stops working at 16:00 every day. The sun rises at 06:00 in the morning and sets at 18:30 in the evening. Every worker has a lunch break of 1½ hours and 2 breaks of 15 minutes each for tea. How many hours does a worker work per week? (One week = 6 working days).

After you have rewritten the questions, solve the problems by calculating the answers.

2. Identify insufficient information

We are going to study the following problem to determine if we have information to enable us to obtain an answer:

Peter sells oranges, pears, guavas and apples. Last week, which was the last week of the month, he sold 50 apples, 100 pears, 80 oranges and some guavas. This week after everybody received salaries, he sold twice as many items. How many did he sell?

After we have read the question it is clear that very little unnecessary information is supplied i.e. in which weeks the sales occurred.

We have to calculate the number of items sold this week. But, first of all, in order to calculate this week's sales we must know how many items were sold last week.

Let's try to add the different numbers:

$$50 + 100 + 80 + ? = ?$$

Because the last value is missing, the answer cannot be calculated!!!

Before we can proceed, we need to ask Peter how many guavas he sold last week.

Although we are not solving problems using the computer, we are already going to use specific words to "ask" the question and to "obtain" or "get" an answer.

Instead of asking the question, we are going to **display** (show) a message to indicate what the question is.

To obtain an answer the word **enter** is used.

So, to solve the problem:

1. Display the question to obtain the number of guavas sold last week.
2. Enter the number of guavas
3. Calculate the total number by adding the four numbers to determine the total number sold last week.
4. Multiply last week's total by 2 to obtain this week's total.

Problem solved!!

Another example:

The building of the head office of the Investment Company consists of 48 storeys and was built in 1999. Every floor contains at least 20 offices and a staff room. Some of the floors also have a seminar room. Lightning strikes the building at least 25 times per year. Calculate the average number of times the building was struck by lightning from 1999 up to the end of last year.

Firstly, remove all the redundant information:

We need not know how many storeys in the building. The number of offices and other rooms do not play any role in solving the problem.

The building of the head office of the Investment Company ~~consists of 48 storeys~~ was built in 1999. ~~Every floor contains at least 20 offices and a staff room. Some of the floors also have a seminar room.~~ Lightning strikes the building at least 25 times per year. Calculate the average number of times the building was struck by lightning from 1999 up to the end of last year.

Rewrite the problem without the extra information:

The building of the head office of the Investment Company was built in 1999. Lightning strikes the building at least 25 times per year. Calculate the average number of times the building was struck by lightning from 1999 up to the end of last year.

Additional information needed to solve the problem:

1. What year was *last year*?
2. We also have a problem with the number of times lightning strikes the building. In the problem statement it is mentioned that the building was struck at least 25 times per year, but we need the exact number of times per year in order to calculate the average!

Steps to solve the problem:

1. Display the question to obtain the date of last year.
2. Enter the year of last year.
3. For every year from 1999, display the question to obtain the exact number of times lightning struck the building during that year
4. Enter the number of times per year for every year
5. Calculate the total number of times lightning struck the building (add together all numbers entered in step number 4)
6. Calculate total number of years: Last year - 1999
7. Calculate answer: total number of times / total number of years

Exercises

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

1. Lesego, who is in charge of all the stock in the company, needs to buy a number of brooms to replenish the stock on hand. Her office is situated on the ground floor of the building, next to the storeroom. The price of one broom is R25.95. How much money will she have to spend?
2. An employee works 5 days per week giving a total of 40 hours per week. He receives a daily wage. All the employees have 15 working days paid leave, as well as all the public holidays. You may assume that a year contains 52.14 weeks. How much is the annual income of an employee?
3. Learn-To-Program Computer College has a computer lab available for students to practice their programming. The lab opens at 08:00 in the morning and closes at 17:00 in the afternoon from Mon to Fri. Students are anxious to work the maximum time available, and are willing to start earlier in the morning and work later in the afternoons. How much time will be available for every student to practice during the week?

1.5 The Problem-Solving Approach

For us as programmers to solve a problem in the most efficient way, we need to follow a systematic process to come up with a solution that is precise, unambiguous, in the correct sequence and gives the correct result in all cases and for all possibilities. The steps to follow are:

- Analyse the problem
- Identify alternative ways to solve the problem
- Select the most effective way to solve the problem
- List all the steps to solve the problem
- Evaluate the algorithm for accuracy

1.5.1 Analyse the problem

The programming process starts with you receiving a specification of the problem to be solved. It is very important that you fully understand the problem before attempting to solve it. Time must be spent working through the problem a number of times to make sure that you understand what is expected from you. The available input (data) must be identified, as well as the expected output (information). If the specification lacks clarity or vital information, extra questions must be asked.

For complex problems, the task should be divided into smaller subtasks. This will make the size and complexity of every part of the task easier to understand and to handle.

1.5.2 Identify alternative ways to solve the problem

There are usually many alternative ways to solve a problem and you, as the programmer, must select the one that is more effective than the others. Look at the problem from different angles and consider different ways to solve it. For instance, if you need to travel from Johannesburg to Cape Town there are many ways to reach your destination: you can travel by road, rail or air; you could go via Kimberley or Bloemfontein.

1.5.3 Select the most effective way to solve the problem

In our example above, when considering the route over Kimberley or Bloemfontein, you need to identify the advantages and disadvantages of each route, with many factors affecting your choice: i.e. difference in distance and state of the road on the route (e.g. road works or potholes).

The same applies to solving a problem on the computer. If you can write a solution in 20 statements or in 50 statements, of course the one with the 20 statements will be the most effective one.

1.5.4 List all the steps to solve the problem (Write the algorithm)

After deciding on the most effective way to solve the problem, you must present all the steps to solving the problem in a precise way. This will not be in the syntax of any programming language. You are still planning how the processing must be done, in a structured and unambiguous way, only concentrating on the logic and not on any programming language. You must be able to adapt your plan to any programming language. There are various methods to write an algorithm, but we will use *pseudocode* to specify the logic of the program.

1.5.5 Evaluate the algorithm for accuracy

The next step is to check your algorithm and logic for accuracy. This is called desk checking. Work through the algorithm with various input values, exactly as the computer would, to test whether the algorithm produces the correct output. This step is of utmost importance. Use different input values to test all possibilities. If you identify any logic errors, you can correct them at this early stage. This will prevent a lot of problems when you start coding in the programming language.

1.6 Write an Algorithm

The steps to solve a problem, that we call an *algorithm*, are written in *pseudocode*, in our case, in English.

Definition of an Algorithm:

"An algorithm is a recipe: it lists steps involved in accomplishing a task. It can be defined in programming terms as a set of detailed, unambiguous and ordered instructions developed to describe the process necessary to produce the desired output from given input. The algorithm is written in simple English."

Leslie Anne Robertson – Simple Program Design

An example of an algorithm written in pseudocode in everyday life may be the following:

Example 1:

Problem: How to prepare for class in the morning:

Solution in pseudocode:

```
Wake up from the alarm
Switch off the alarm
Get out of bed
Shower
Dress
Prepare breakfast
Eat breakfast
Brush teeth
Pack bag
Walk to class
```

Have a close look at the algorithm and you will see that it is possible to exchange some of the instructions, but not all of them. You will get the same result if you prepared the breakfast and eat it before you shower and dress.

Let us look at another option: Is it possible to shower before getting out of bed? Is it possible to get out of bed before you have woken up? Is it possible to eat your breakfast before you have prepared it?

Example 2:

Problem: How to add up a list of prices on a pocket calculator:

Solution in pseudocode:

```
Turn on the calculator
Clear the calculator
Repeat the following instructions
    Key in the Rand amount
    Key in the decimal point (.)
    Key in the cents amount
    Press the addition (+) key
Until all prices have been entered
Write down the total price
Turn off the calculator
```

In this second example it is clear that none of the steps can be swapped. All these steps must be executed in the sequence that they appear, or we will not get the desired result.

It is clear that the steps must be in a specific order to solve a problem.

Exercise:

Write algorithms for the following everyday life situation:

1. How to make putu-pap.
2. How to buy a number of items from a grocery store.

Rules regarding an algorithm:

- Must be precise (exact, accurate, correct)
- Must be unambiguous (not open to more than one interpretation, not of double meaning)
- Must be in the correct sequence
- Must give the correct result in all cases and for all possibilities

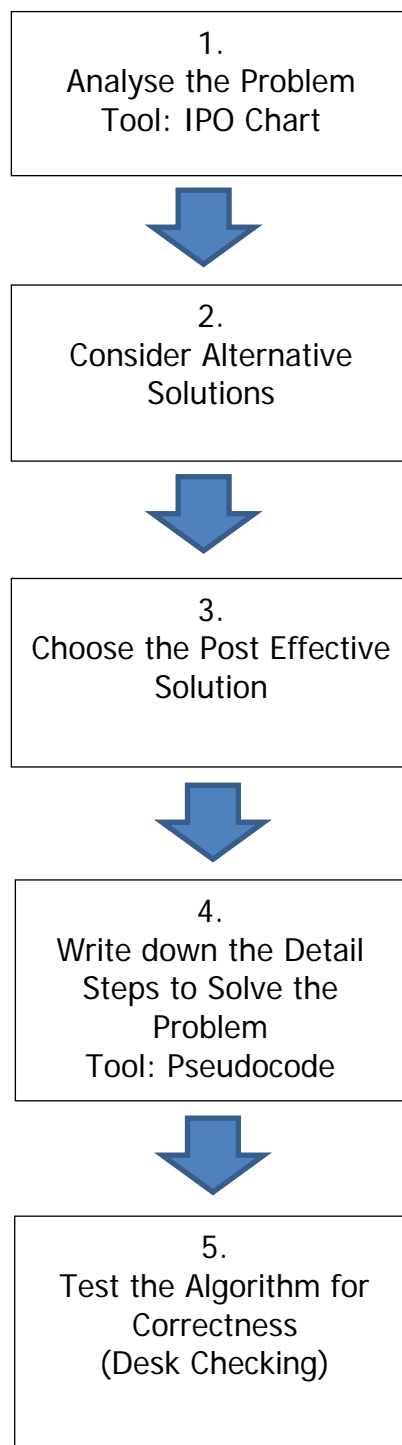
Test the previous algorithms to see if all the rules applied.

1.7 The Entire Programming Process

Once you have completed the algorithm as outlined above, follow the following steps to code the program in a programming language, convert it into machine code, and finally produce the required output:

- Code the program from the logic planned in the algorithm into a given programming language (in our case C++), to produce the source code. Each programming language will use words from the human language; it will have rules for grammar, syntax and punctuation, and sometimes it will read a bit like a human language.
- Translate the program into machine language by means of a compiler. The compiler will convert the source code into machine language, which is the only language that a computer can understand. Once translated, a program can be executed or run.
- Test the program for correctness with input data.
- Put the program into production.

Design Phase:





Implementation Phase – All programming languages:

1.
Write the Steps of the
Algorithm according to
the Syntax of the
Programming Language
(C++) - Coding



2.
Enter the Code using the
Editor of the
Programming Language
(Source File)



3.
Compile (transform the
code into 0's and 1's
(Machine Language)) the
Source Code to give an
Executable File (Object
File)



4.
Test the Program



5.
Implement the Program

1.8 C++ NOTES

1.8.1 COMMONLY USED PROGRAMMING TERMS:

- A **computer** is a machine that is used to solve problems using data. A computer only does what it has been instructed to do.
- A structured combination of data and instructions used to operate a computer to solve a problem are called **computer program** or **software**. When computers follow the instructions written in a program, we say it *executes* the program.
- Set of instructions to operate a computer can be written using a **programming language**. All computers have a programming language that they understand, commonly referred to as **machine language**, which is expressed binary system (i.e. 0s and 1s). This language is very difficult for human to write or to interpret.
- Programming languages are classified by levels. We have **Low-level programming** languages which are close to the computer hardware. Instructions written in low-level languages only execute on the type of computer there were created in. Examples of low-level languages include machine code and assembly.

E.g. of instructions written in assembly language to calculate the sum of two numbers:

```
mov a , 5  
mov b , 10  
add a , b
```

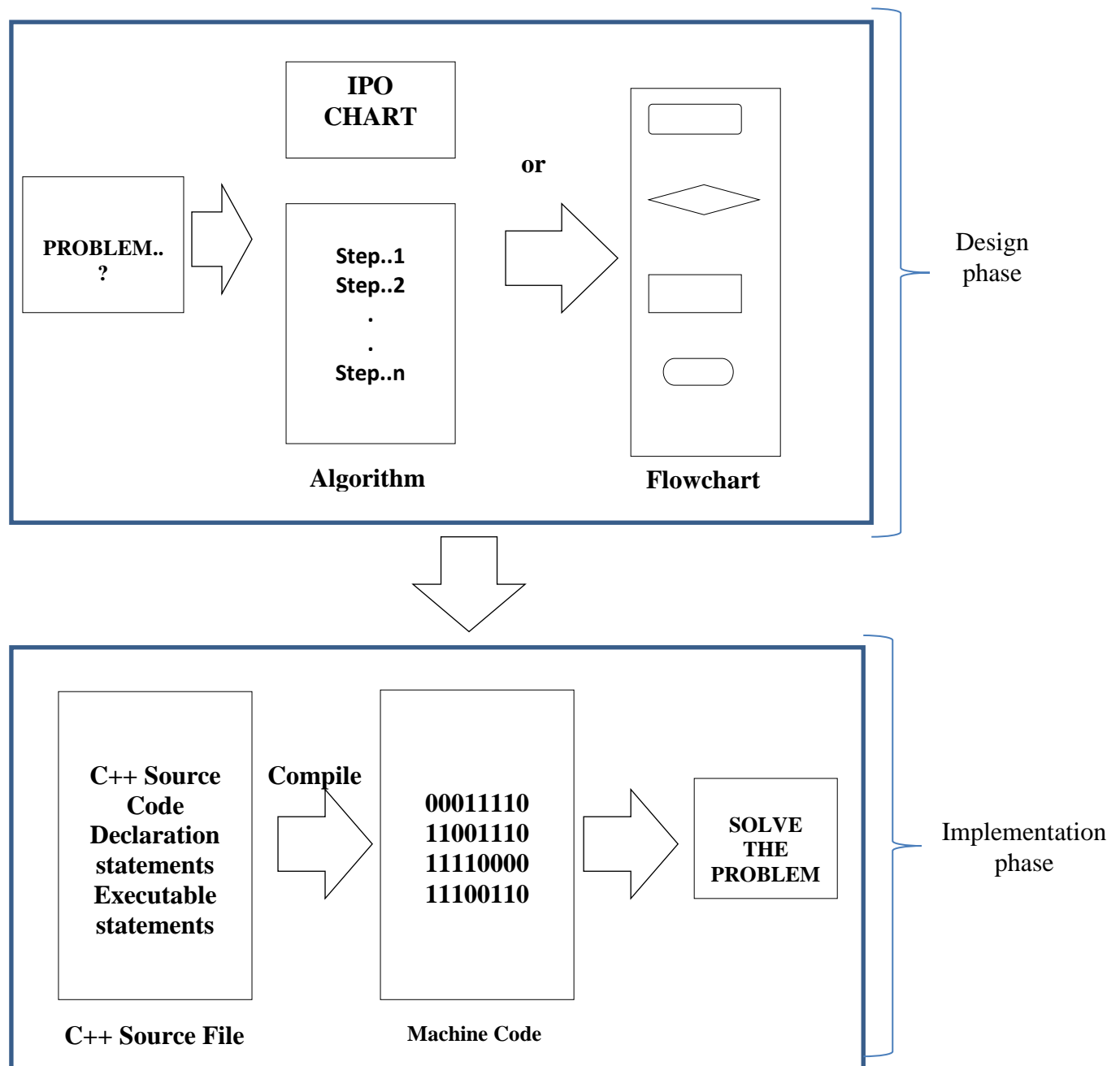
- **High-level programming** languages are easier to learn because there use English like statements. Mathematical operations are also included in high level languages. These languages require more translation before the computer will understand them. Examples of high-level languages include C++, Java, visual basic and so on.

E.g. of instruction written in high-level language to calculate the sum of two numbers:

```
c = a + b;
```

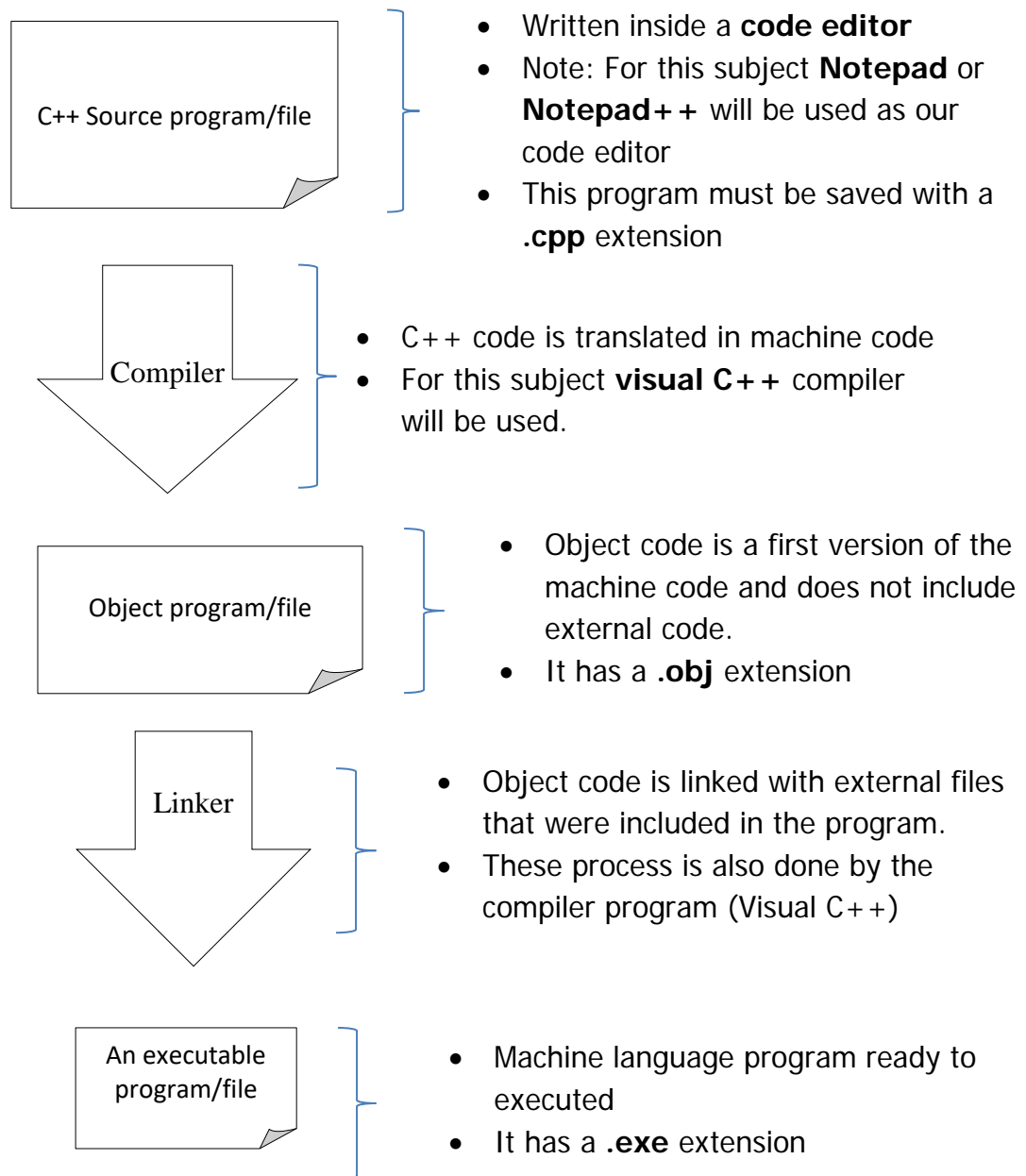
1.8.2 PROGRAMMING TRANSLATION

The first block represents the design phase of the program development life cycle, which is clearly explained in the theory notes.



In the implementation phase block, the code written in C++ must be translated into binary code which is understood by computers before the problem can be solved. A program used to translate the source code into a machine code is called a **compiler**.

1.8.3 CREATING AN EXECUTABLE C++ PROGRAM



1.8.4 COMPILE A C++ CONSOLE APPLICATION USING COMMAND PROMPT

Note: Before you attempt steps below make sure you have either **Microsoft Visual Studio 2010/2013** installed on your machine.

The following steps show how to compile a Visual C++ program using command prompt:

STEP 1: OPEN VISUAL STUDIO 2013 INTERFACE

Open **Visual Studio 2013** by clicking on **Start**, All Programs then clicking on **Visual Studio 2013** as shown below

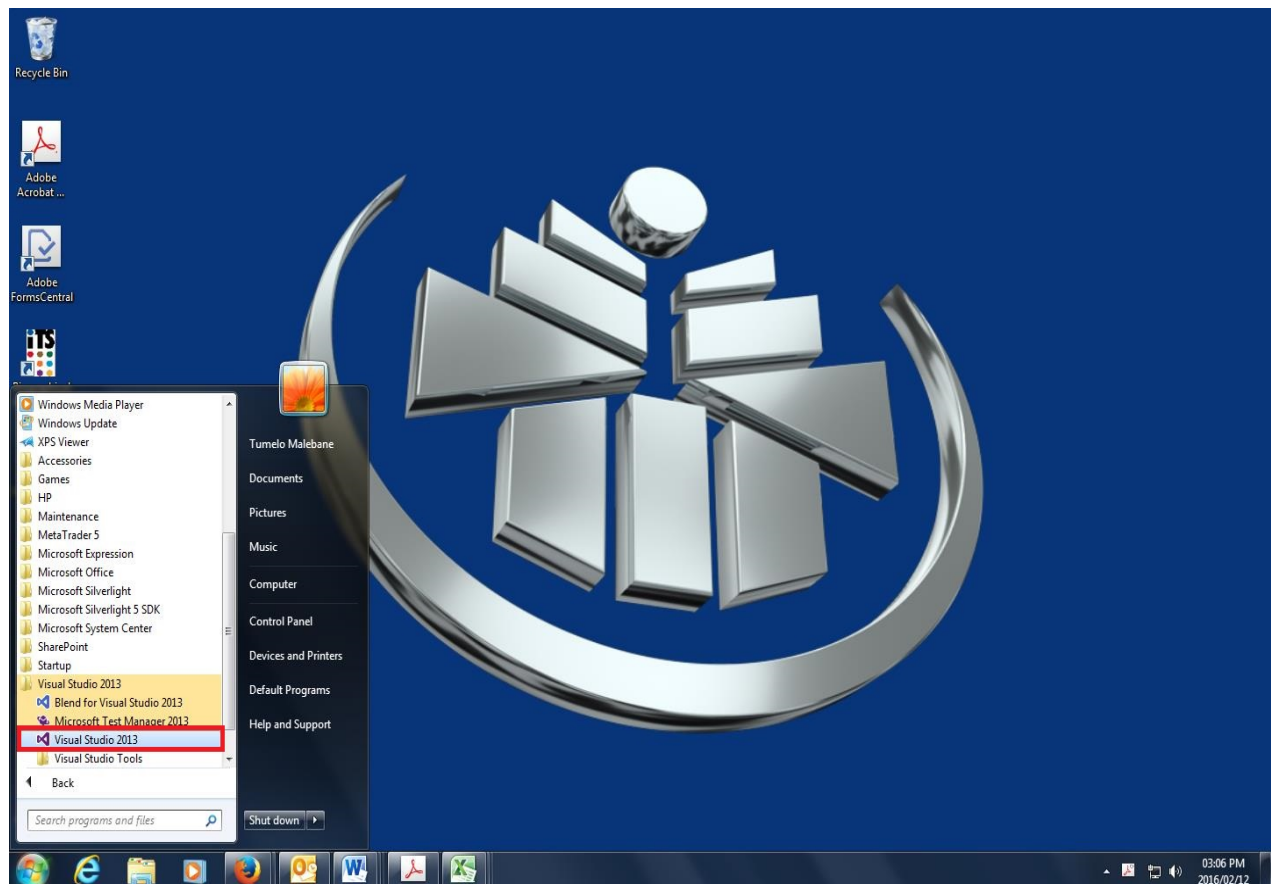


Figure 1-1: Opening Visual Studio 2013 interface

After completing **step 1**, the following window will appear:

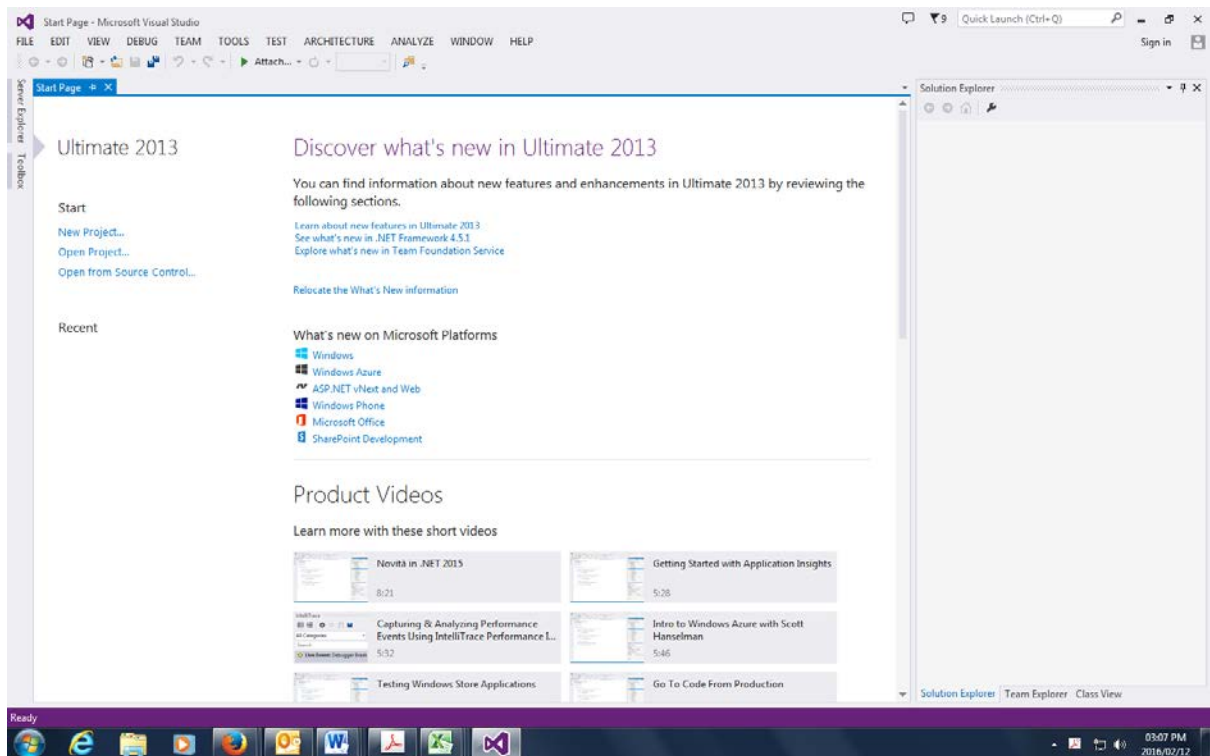


Figure 1-2: Visual Studio 2013 welcome window

STEP 2: Create your project

Click on the **FILE** menu, and then click on **New** to open up a sub menu, select **Project**

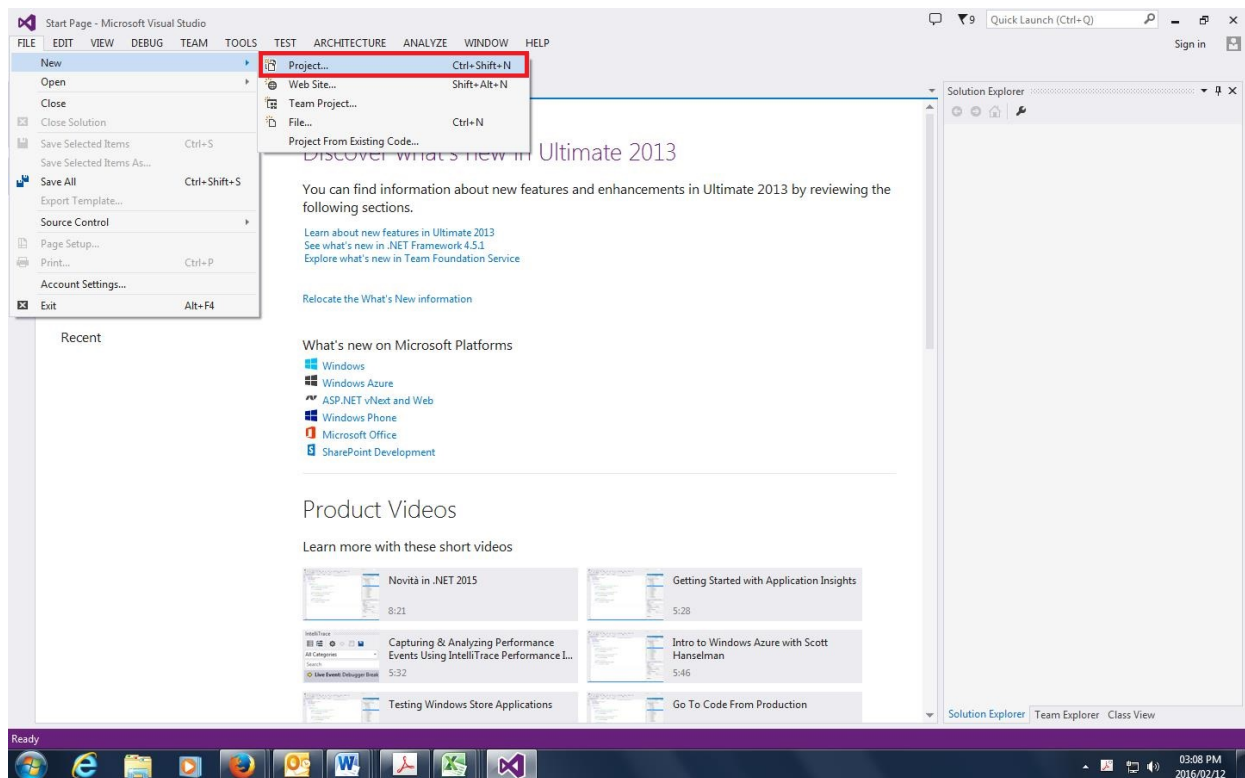


Figure 2-1: Creating a project

After completing **step 2**, the following window will appear:

Ensure that **Visual C++** has been selected on the list of languages and then select **Win32 Console Application** on the right hand side list. On this window you will also provide the name for your project/application and specify the location on which it should be saved. It is always required to save all work/projects on the D: drive, this will safeguard your project and make sure that it is still available even on the event that your machine unexpectedly shuts down. Once all information has been provided, click on **OK**

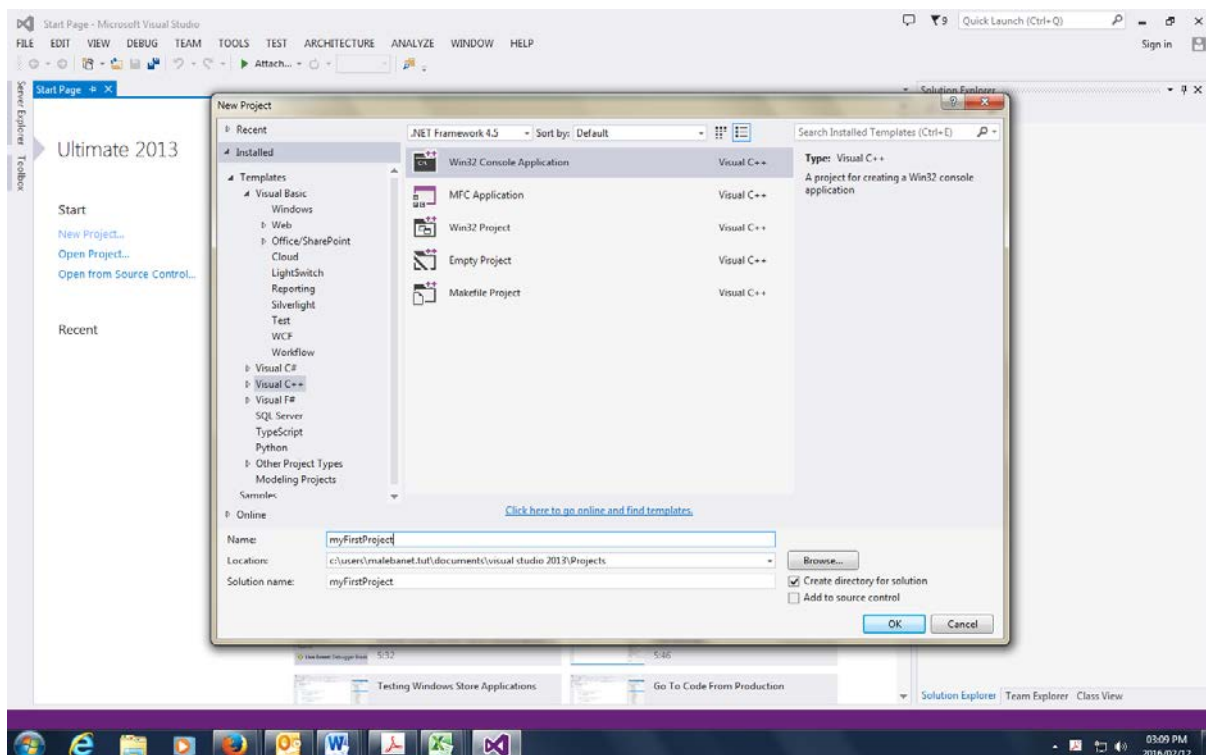


Figure 2-2: Creating a project

STEP 3: Finalise project creation

Click **Finish** on this window to complete this process.

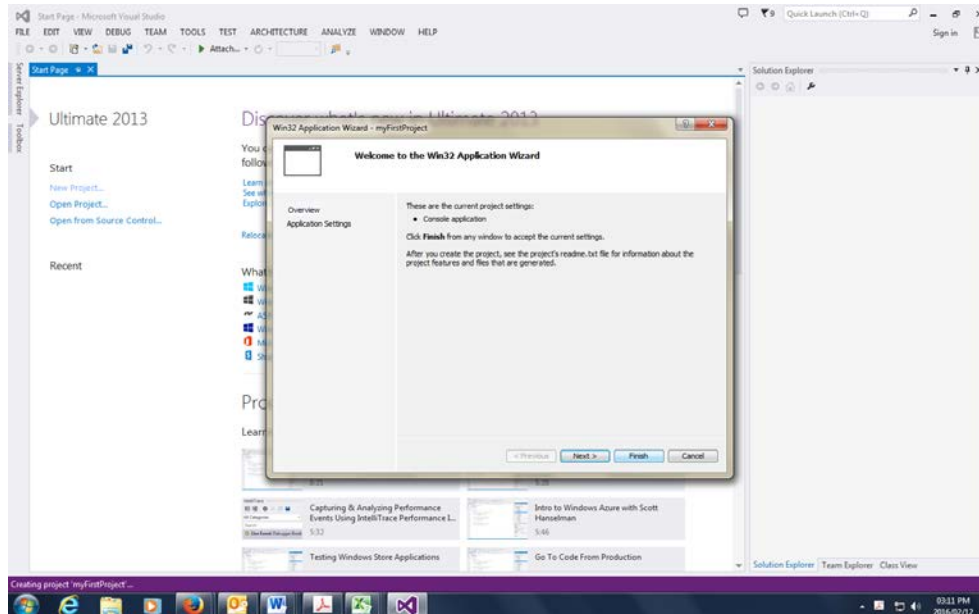


Figure 3-1: Creating a project

After completing **step 3**, the following window will appear:

This part of the IDE is called an **editor** and this is where all your source code will be typed in and then later compiled and executed. A basic structure has been provided to you and you will then provide additional libraries and source code as needed.

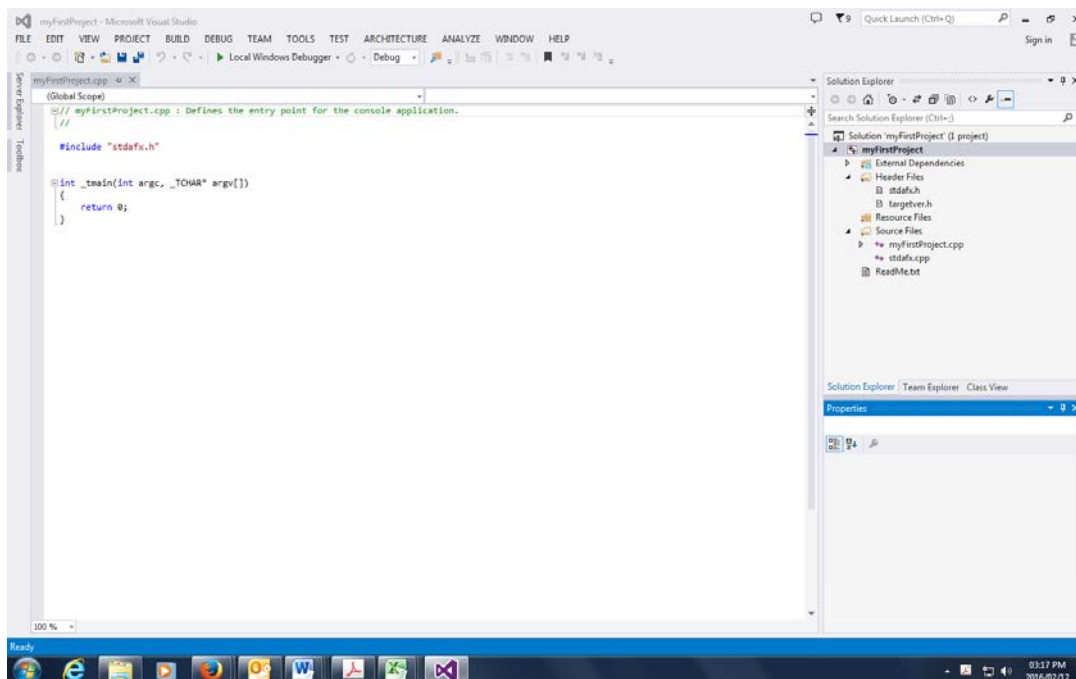


Figure 3-2: Text editor with basic code

STEP 4: Add C++ code to your source file

Type in the highlighted lines of code into your editor, once done; your application must look exactly as shown below in **Figure 4**. All this typed in code will be saved into your source file which will have an extension of **.cpp**

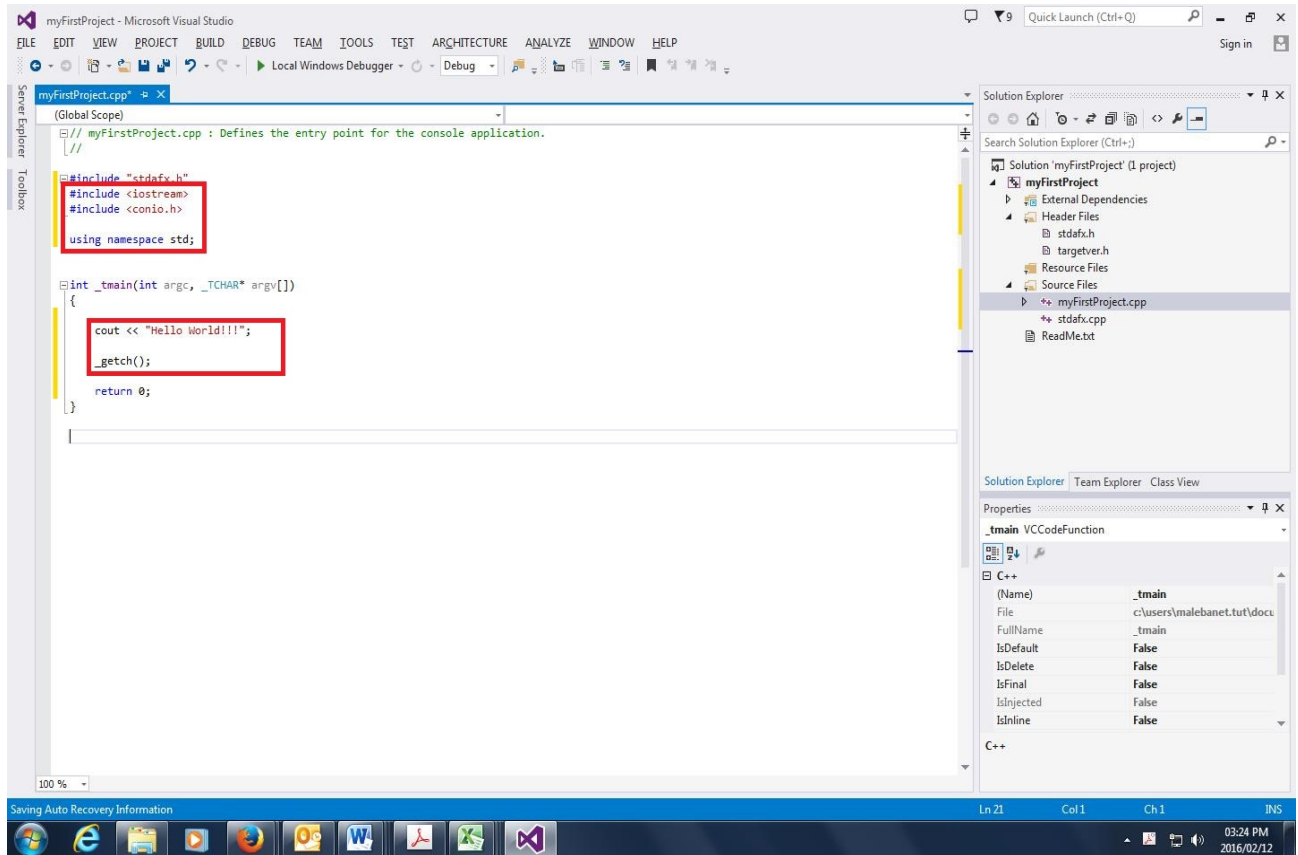


Figure 4: Adding code

STEP 5: Compiling your code

Click on the **BUILD** menu and then select **Build Solution**. This process allows your code to be checked for any errors (syntax) and in the event that none are found, your code will then be compiled and converted to byte code. If your code does have syntax errors, they will be highlighted in the editor and your code will not compile until all of them have been fixed.

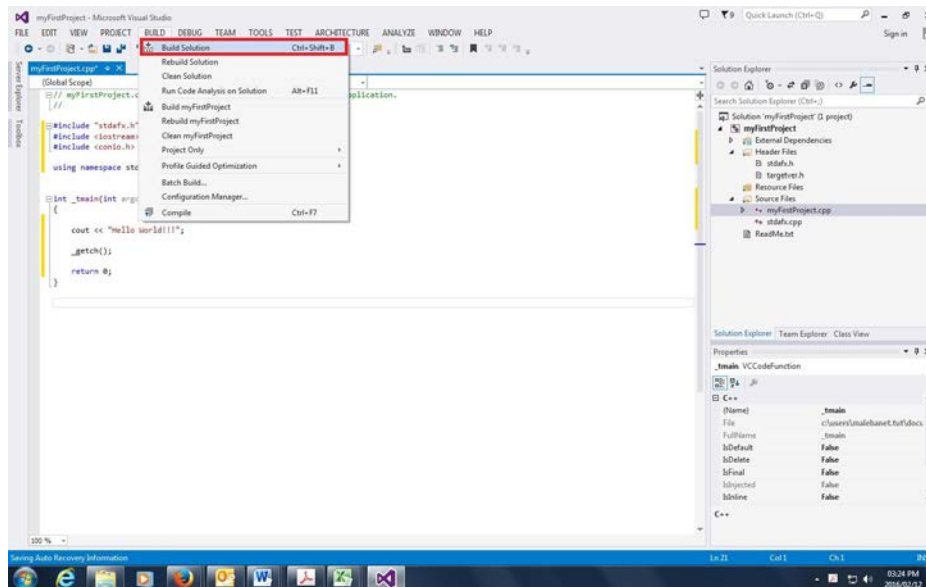


Figure 5-1: Compiling code

After completing **step 5**, the following window will appear:

It is always important to inspect the window as highlighted in red below; they show vital information after every attempt to compile/build your application. This code has been compiled with no errors and will indicate that the build was successful.

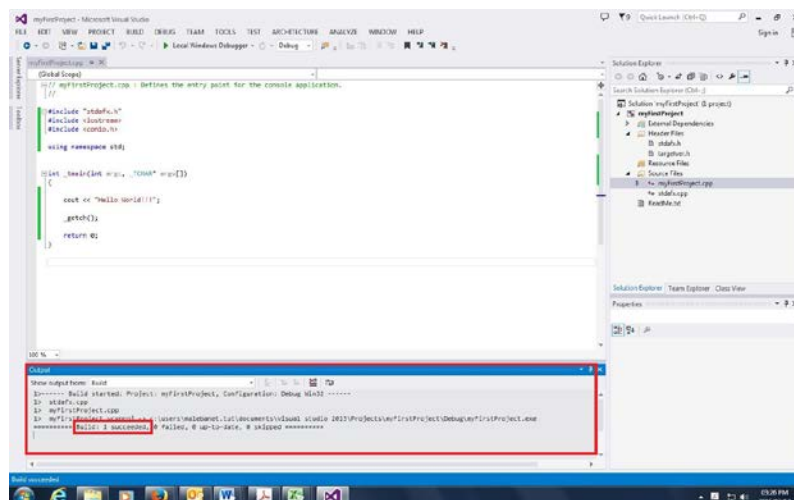


Figure 5-2: Compiling code

Save your work regularly by clicking on the button showing "*2 blue disks*". This will save all files associated with your application

STEP 6: Execute/run your program

Click on the highlighted button to execute your application. This will create **.obj** and **.exe** files that form part of your application. The result of the application will be shown through the **.exe** file.

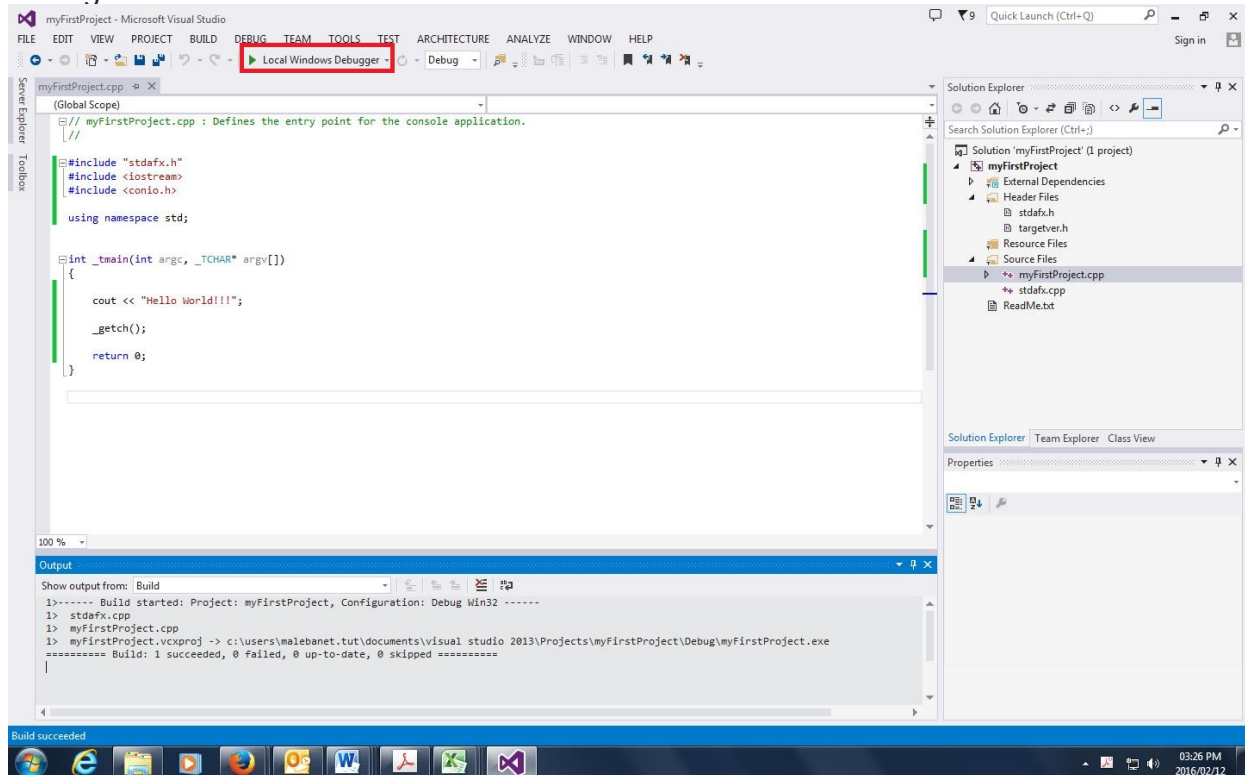


Figure 6-1: Executing an application

After completing **step 6**, the following window will appear:

This black screen will show the **output** or result of your source code (C++) code that you typed into the editor and compiled. When done viewing the result, simply press the **ENTER** key on your keyboard to go back to your editor.

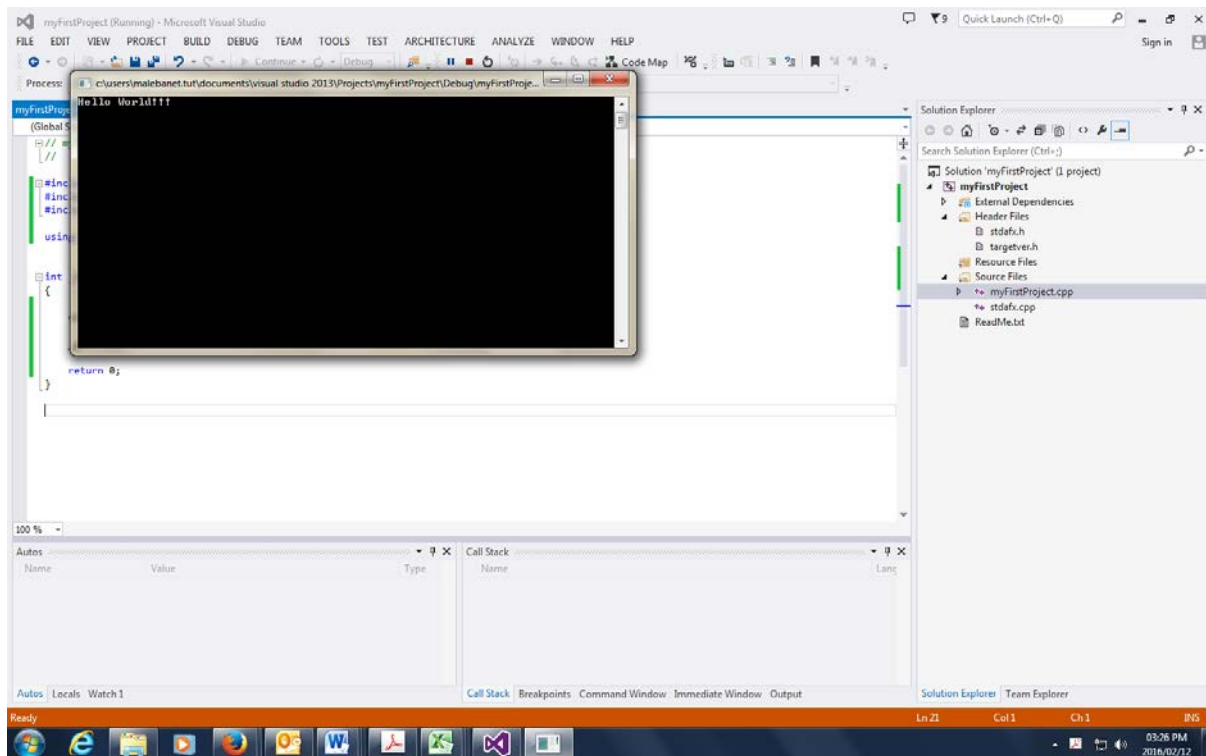


Figure 6-2: Black screen showing output

1.9 SUMMARY

In addition to these notes, study chapter 1, **page 14** to **page 27**.

Do the following exercises:

- **Exercises 1.1** Page 11 – 12 (Theory exercises):
 - do exercise 1 a, b, c, d , e, f, g, m, n, o, p and q
 - do exercise 2 – 8
- **Exercises 1.2** Page 17 (Theory exercises):
 - Do only exercise no 1
- **Exercises 1.3** Page 21 – 22 (Practical exercises)
 - Do all the exercises (1 - 5)
- **Exercises 1.4** Page 21 – 22 (Practical exercises)
 - Do all the exercises (1 - 4)
- **Exercises For Section 1.7** Page 33 – 34 (Please to not write any code to answer this exercises, only tools learned in the design phase of the program development life cycle can be used)
 - Do all the exercises (1 - 9)

Chapter 2

General Concepts and Arithmetic

2.1 Introduction

In this chapter you will learn how to go about solving a programming problem. After determining the input and output for the problem, you will learn how to develop the process to solve the problem. The tools to use at the different stages of the design process will be discussed. Then we will solve a simple problem according to the five-step plan by applying all the tools we have learned to use.

2.2 Variables

A variable is the name given to an area in the memory of the computer designed to store a particular data item. It need not contain a value and if it does, this value may change according to circumstances. But, a variable may only contain one value at a specific time.

A variable may be compared to the scoreboard of a team that has only one score at a specific time.

By using variables, a programmer can instruct the computer in an exact way what to do. If the programmer refers to a variable name, it refers to a specific memory area and it is able to get hold of the value in that memory area. The programmer can then use it to do a calculation or display it on the screen.

Example:

hoursTravellingTime

8

On a different route:

hoursTravellingTime

12

The variable called *hoursTravellingTime* may only have one value at a given time. A variable can also be explained by studying the following:

Pirates and Kaizer Chiefs are playing a game of Soccer: Before the game starts the scoreboard shows the value of 2 variables i.e. scorePirates and scoreChiefs as follows:

scorePirates	scoreChiefs
0	0

After 10 minutes Pirates scores a goal and the scoreboard changes as follows:

scorePirates	scoreChiefs
1	0

After another 15 minutes Chiefs scores a goal and the scoreboard changes as follows:

scorePirates	scoreChiefs
1	1

Just 5 minutes before the final whistle Chiefs scores another goal and once more the scoreboard changes to the final score:

scorePirates	scoreChiefs
1	2

It is clear that the value of the variables called scorePirates and scoreChiefs changes due to specific circumstances but can only contain one value at a one specific time. By looking at the scoreboard at the end, there is no way that the previous score (previous values) can be seen.

Every variable must have a name according to certain rules:

1. The name must be unique. Two variables in the same program may not have the same name.
2. It must be descriptive e.g. hours, wage.
3. It may not be more than one word (must form a unit), e.g. hourly rate is incorrect, rather use hourlyRate. When we join 2 words, the first word starts with a lower case letter while the second and following words start with uppercase letters e.g. nameOfBook, numberSchoolChildren. It may also be joined by the underscore (_) character, e.g. hourly_rate.
4. It may contain letter and numbers, but not only numbers e.g. class2 is valid, but 123 is invalid.
5. It must start with a letter of the alphabet e.g. 2class is invalid.
6. It may not contain any special characters e.g. &, #, @, etc. (except for _).
7. It may not be too long.
8. The name may not be a reserved word in the programming language (e.g. int).

Examples:

Description	Variable name
Name of employee	empName
Number of students	numStudents
Name of book	bookName
Price of Item	itemPrice

Exercises:

1. Do the following variable names comply to the rules of variable names? If not, supply a reason:

Variable Name	Valid / Invalid?	Reason
5thGrade		
member of club		
def		
deptCode		
theAddressOfTheCompanyInJhb		
grade&mark		
test_mark		
7654		

2. Supply appropriate names for the following variables:

Description	Variable name
Name of employee	
Price of car	
Address of company	
Size of area	
Salary of worker	
Type of car	
Total number of sales	
Number of books	

2.3 Constants

Constants can be one of two types: Either a literal value, such as 5 or 0.6, or it can be a data item with a name and a value that never changes. In the latter case the same rules that apply to the naming of a variable also apply to the naming of a constant.

Examples of *constants as a data item* are the following:

- There are always 7 days in a week e.g. *noDaysWeek* should contain 7.
- A day has 24 hours e.g. *dayHours* should contain 24.
- The percentage used for sales tax or VAT is 14% (at the moment) e.g. the value of *salesTax* is 0.14.

These constants are usually assigned a value at the beginning of the algorithm and it never changes like the value of a variable.

Examples of *constants as a literal value* are as follows:

- `total = 0`. The 0 is then the literal constant.
- `display "The number of students is = ", numStuds`. The literal constant is `The number of students is =`. The string in inverted commas is a constant, and will be displayed exactly as it written, but without the quotes.

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

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

The constant (literal / fixed) part of the display statement will be the legend that has the value

`Percentage obtained by student:`

The variable *percentage* will contain the calculated percentage (e.g. 55) obtained by the student. The variable name *percentage* is not contained in quotes.

The statement

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

will produce the following output if *percentage* has a value of 50:

`Percentage obtained by student: 50`

2.4 Data types

Every data item - variable or constant as a data item - has a specific data type. Data types can be divided into numeric data items and non-numeric data items:

1. Numeric data items

Numeric data items can be used in calculations. They are divided into two types:

Integers: Represents positive or negative whole numbers and zero, and has no decimal point. Examples of integers are the number of students or the quantity in stock.

Examples: 5, -120, 8765, 0

Real numbers: Contains positive or negative values with a decimal part. Examples of real numbers are the length of a table in metres or centimetres, or an amount of money in rand and cents.

Examples: 15.7, -0.45, -987.32

2. Non-numeric data items

Character: Contains a single letter, number or special character, such as: A, %, \$. The value of a character is always enclosed in quotes.

Examples: "A", "*", "g", "7".

Please note that the character "7", which is declared as character, is not regarded as numeric, and may not be used in calculations. When declared as a character, it can be used as a code – for instance the group number is "7".

String: A **string variable** consists of two or more characters and must be enclosed in quotes.

Examples: "John", "The result is positive", "The percentage is"

Boolean: Represents a control flag or switch and may only contain a true or a false value. Compare to a light switch that can either be on or off. In a computer language, a **true** value is represented by a non-zero value, whereas a **false** value is represented by a zero. A **Boolean variable** is used to test if a specific condition is true or false, and is called a logical variable.

Examples: Is the person a member of a club? Has the student registered? True or false?

Only integers and real numbers may be used when doing arithmetic.

Numeric values are not included in quotes, while character and string values are included in quotes, e.g. a message may be "The name of the girl is Petunia" and a code may be "G".

Do not confuse the value of a data item with the name of a data item.

If a numeric value is included in quotes, this value is not regarded as numeric and it cannot be used in arithmetic e.g. "15" is not numeric.

Exercise:

Complete the following table by assigning a suitable variable name to every data item and specify the type:

Data item	Variable name	Data type
Number of employees		
Name of employee		
Salary of employee		
Member of a union?		
Type of animal		
Increase percentage		
Number in stock		
Code of department (X, Y or Z)		
Lecturer available?		

2.5 Arithmetic expressions

The majority of computer programs contain arithmetic. It is therefore important to study how arithmetic is used to solve problems using a computer. Arithmetic may be used to calculate the percentage mark of a student, an amount due at a till-point or the increase that an employee will receive based on a percentage.

2.5.1 Using variables in expressions and assignments

Equations are often used in arithmetic calculations, for example:

`answer = 12 + 7`

The calculation to the right of the equal sign is done first, and then the result is assigned to the variable on the left of the equal sign. In the example above this

variable is *answer*. So, after the execution of the statement, the variable *answer* will contain a value of 11.

In programming the equal sign (=) has two meanings:

1. It can be used to assign a value to a variable or to a constant:

```
total = 0  
price = 12.50
```

2. It can be used to replace the value of the variable:

```
diff = amount - 7.56  
totCount = totCount + numStuds
```

The last example, `totCount = totCount + numStuds`, will add the value of *numStuds* to the value of *totCount* to produce a new value for the variable *totCount*.

If *totCount* has a value of 47 and *numStuds* has a value of 8, then the right-hand side of the equation will be calculated first: $47 + 8$ and the new value for *totCount* will be 55. The previous value of *totCount*, which was 47, is now replaced by the new value, which is 55.

2.5.2 Keywords used in arithmetic expressions

Some of the key words that can be used in problem statements that contain mathematical calculations are the following:

Addition:

Key word (s)	Example
Increased by	Increase the total by 12
More than	John earns R50 more than Jack.
Combined	The combined points of Sam and Sally is 129
Together	Together, Thabu and Jabu worked for 18 hours
Total of	Calculate the total of the first and the second set of quantities
Sum	What is the sum of the prices of 3 items?
Added to	The 4 th test mark must be added to the final mark
Plus	12 plus 20 is 32

Subtraction:

Key word (s)	Example
Decreased by	The final mark must be decreased by 5
Minus	The amount minus the discount is the net amount
Less than	Pete earns R20 less than Simon
Difference	The difference between my length and your length is 5 cm
Subtract from	Subtract 15 from the points accumulated
Between	The difference between Pete's marks and Rose's marks is 10
Fewer than	Goodman has worked 5 hours fewer than Samuel

Multiplication:

Key word (s)	Example
Of	A quarter of 100 is 25
Times	The final point is equal to the points obtained times 1.5
Multiplied by	The wage is the hourly rate multiplied by the hours worked
Product of	Calculate the product of number of items and the price per item
By a factor of	The total number must be increased by a factor of 12%

Division:

Key word (s)	Example
Per	Calculate kilometres per litre for fuel
Out of	15 out of 60 is 25%
Quotient	The quotient of 20 and 4 is 5 i.e. 20 divided by 4 is 5.
Ratio	If the ratio of girls to boys is 2:3 and there are 25 children, how many girls are there?

2.5.3 Arithmetic operators

An **operator** is a symbol used within an expression or equation that tells the computer how to process data. The symbol joins the **operands** to be processed. For example, in the calculation $16 + 7$, 16 and 7 are *operands* and + is the *operator*.

The following operators are used to perform arithmetic operations:

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

Arithmetic operators and the rules of precedence

Integer division can only be done when dividing an integer value by another integer value because it discards (drops) the decimal part and does not round the answer, for example: $57 \backslash 5 = 11$

Modulus arithmetic is done when the user wants to know what the value of the remainder is when dividing one integer into another integer, for example: $57 \bmod 5 = 2$ (5 goes into 57 11 times, with a remainder of 2).

When we talk about the **precedence** of operators, we talk about the order in which the operators are executed when they appear in the same expression. The order of processing is indicated in the last column in the table above. The order of precedence in an expression is important. When an expression is executed, it is always done:

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

When operators with the same precedence occur in the same expression, they will be executed from left to right.

Parentheses (brackets) are used to change the order of execution. Calculations in parentheses have a higher priority than any other of the operators. The operators within the parentheses, however, are executed in the same order of precedence.

Example of execution of an expression according to the precedence of the operators:

$$4 + 3 * 2 ^ 3 \text{ mod } (15 - 12)$$

To find the answer to the expression, we need to determine which operator to execute first. The sequence of steps, with the answer of the step underlined, is as follows:

Step 1: The expression in brackets is done first:

$$4 + 3 * 2 ^ 3 \text{ mod } \underline{3}$$

Step 2: Then the exponentiation:

$$4 + 3 * \underline{8} \text{ mod } 3$$

Step 3: Then mod:

$$4 + 3 * \underline{2}$$

Step 4: Then *:

$$4 + \underline{6}$$

Step 5: And lastly +:

$$10$$

Example: Calculate the value of the variable **k** where:

$k = a * b ^ (14 - c) \text{ mod } 4 + 3 \setminus a$, and where **a**, **b** and **c** are integers with the following values: $a = 3$, $b = 5$, $c = 11$

Substitute the values of the variables before calculating:

$$\begin{aligned} k &= 3 * 5 ^ (14 - 11) \text{ mod } 4 + 3 \setminus 3 \\ &= 3 * 5 ^ \underline{3} \text{ mod } 4 + 3 \setminus 3 \\ &= 3 * \underline{125} \text{ mod } 4 + 3 \setminus 3 \\ &= 3 * \underline{1} + 3 \setminus 3 \\ &= 3 * 1 + \underline{1} \\ &= \underline{3} + 1 \\ &= 4 \end{aligned}$$

Note that $(14 - 11)$ is executed before the other operations because it is in parentheses.

Exercises:

The following values are given:

$$W = 10, X = 5, Y = 12, Z = 2$$

Solve the following equations to calculate the value of A by substituting the given values:

1. $A = W + Y / Z - X$
2. $A = Z * 3 - W / X$
3. $A = W + Y \wedge Z \setminus 5$
4. $A = (Y - 2 * Z) + Y / 6 * W$
5. $A = X + Y - (Z * W + Y / 3) + Y * W$
6. $A = Y \setminus X + (Y + 1) \bmod X$
7. $A = Y \setminus W + 6 * Z + Y \bmod Z$
8. $A = Y + X \bmod W$

2.5.4 Setting up arithmetic equations

An arithmetic equation is also called an **assignment statement** in programming, in which variables and/or constants are assigned to a variable on the left-hand side of the equation. For example, $a = b + c$, where a must be a numeric variable that will contain the result, once executed, of the expression on the right-hand side of the equation. b and c must also be numerical values, but may be variables or constants.

Variables and/or constants on the right-hand side of the equation are always separated by operators. When writing an arithmetic equation, the mathematical rules, set out in the table of operators on page 10, must be followed. An equation contains only one variable to the left of the equation to store the result.

The following equation contains the variables X, A, B, C and D:

$$X = A^3 - 2B + \frac{2(C + D)}{2}$$

This equation must now be written in pseudocode (one-line) format, as follows:

$$X = A \wedge 3 - 2 * B + (2 * (C + D)) / 2$$

Note how the brackets are used to execute the statements in the correct order.

Exercises:

Rewrite the following equation in pseudocode format and then calculate the value of W if $K = 5$, $Y = 17$, $Z = 10$, $S = 24$

$$1. W = \frac{KZ - Y}{Y - 6} + \frac{3Z + S}{Z + 4}$$

$$2. W = (Y - Z)^{(K - 3)}$$

$$3. W = S - Z - 3K^2$$

2.5.5 Examples using variables and constants

The following examples illustrate what we have just learned. In each of the examples, the arithmetic equation to solve the problem is given, after the variables and constants have been planned. Please note the meaningful names chosen for variables and constants.

Example 1:

Thembeke works eight hours per day at a pay rate of R22 per hour. She pays R15 for the bus per day. How much money will she take home after a six-day week?

Variables (integer):	amountMoney
Constants (integers):	hours = 8
	payRate = 22
	transport = 15
	noDays = 6
Equation:	$\text{amountMoney} = (\text{hours} * \text{payRate} - \text{transport}) * \text{noDays}$

Please note that we have not included any units (in this case R) in any calculation. This also applies to other units such as metres, cm, gram, kilogram, etc.

Example 2:

Simon has 127 sweets and wants to pack them in packets of 12 each to sell at the fair at school. How many packets will he be able to pack?

Variables (integer):	fullPackets
Constants (integer):	numSweets = 127
	noPerPacket = 12
Equation:	$\text{fullPackets} = \text{numSweets} \setminus \text{noPerPacket}$

Example 3:

Thandi sold 32 loose apples at her stall today. She paid 80 cents each for the apples, and sold them for R1.20 each. How much profit did she make on the apples today?

Variables (real number):	profit
Constants (real numbers):	buyPrice = 0.80
	sellPrice = 1.20
(integer):	numApples = 32
Equation:	$\text{profit} = (\text{sellPrice} - \text{buyPrice}) * \text{numApples}$

Please note that the constant **buyPrice** has been converted from a price in cents to a price in rand, so that the two prices can be subtracted from each other.

Example 4:

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

Variables (integer):	noOfItems
(real numbers):	firstAmount, amountDue
Constants (real numbers):	discount = 7.5
	priceOfItem = 15.85
Equations: Option 1	$\text{firstAmount} = \text{noItems} * \text{priceOfItem}$ $\text{amountDue} = \text{firstAmount} - (\text{firstAmount} * \text{discount} / 100)$
Option 2	$\text{firstAmount} = \text{noItems} * \text{priceOfItem}$ $\text{amountDue} = \text{firstAmount} * 0.925$
Option 3	$\text{amountDue} = \text{noItems} * \text{priceOfItem} * 0.925$

Exercises:

1. Nancy bought a number of items for R5.60 each. She receives a discount of 5% on the amount due. Add 14% sales tax and then calculate the final amount due.
2. Tony scores points in two competitions. The first competition is out of 34 points, and the second out of 125 points. Calculate the average percentage scored in the two competitions.
3. Julius needs to know the total length of line in metres he has to buy for three washing lines. The three lengths are available in centimetres.
4. Lesego has baked a batch of biscuits. She puts 15 biscuits in a packet and Sipho will receive the remaining biscuits after she has packed them. Write two equations: How many packets of biscuits were packed by Lesego and how many biscuits will Sipho get?

2.6 DATA TYPES IN C++

In C++ data types specify:

- Types of data or values allowed, e.g. text for string type, whole numbers for int data type, etc.
- Set of operations that are permitted, e.g. only numeric data types can be used with arithmetic operations
- Range of values that can be stored, e.g. only true (1) or false (0) can be permitted in a Boolean data types
- The size of storage memory, e.g. in C++, a char data type which only permit a single character occupies 1 byte

C++ Provide two (2) categories of data types, namely:

2.6.1 Primitive types

These are built-in data types that form part of the C++ reserved keywords. C++ treats all primitive types as numerical data types. A table below includes commonly used built-in data type:

Data type	Description	Size (byte)	Range	Value
bool	Boolean value. It can take one of two values: true (1) or false (0).	1	0 or 1	true
char	Character	1	Any of the 256 characters	'9'
int	Whole number	4	-2147483648 to 2147483647	101955
float	Floating point number.	4	Floating-point number of up to 7 digits	0.14f
double	Double precision floating point number.	8	Floating-point number of up to 15 digits	0.14

The char data type is used to store a single character. Refer to page 40 – 41 for additional explanation on the char data type and the escape character. C++ provides short or long int and long double as primitive data types. These are fully explained in table 2.4 and 2.5 on page 44 and 45 respectively.

The integer data types, except bool, may include a signed or unsigned specifier. Signed types can represent both positive and negative values, whereas unsigned types can only represent positive values (and zero).

C++ provides the operator `sizeof()` to check the storage size for each data type. The program below illustrates how to use the `sizeof()`.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    cout<<"Storage size for char = "<<sizeof(char)<<" bytes.\n";
    cout<<"Storage size for int = "<<sizeof(int)<<" bytes.\n";
    cout<<"Storage size for float = "<<sizeof(float)<<" bytes.\n";
    cout<<"Storage size for double = "<<sizeof(double)<<" bytes.\n";

    _getch();
    return 0;
}
```

Figure 2.1 Source code

```
Storage size for char = 1 bytes.
Storage size for int = 4 bytes.
Storage size for float = 4 bytes.
Storage size for double = 8 bytes.
```

Figure 2.2 Output

Try this: Create a program that will verify the storage size (in bytes) of `bool`, `short int`, `long int`, and `long double`.

2.6.2 Class data type

This is a user-defined type or programmer-created data type. For this subject only the **string data** type will be used. Text is only permitted in a string data type. However, a string header file must be included using **`#include <string>`** before a string can be used, see figure 2.3. The string data type is defined or created inside these header files.

```
#include <iostream>
#include <conio.h>
#include <string>

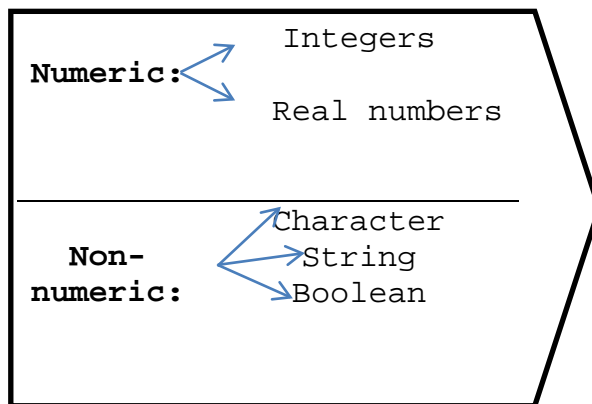
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    ,
```

Figure 2.3

Figure 2.4 illustrate how to translate data types in pseudocode to types in C++.

DESIGN PHASE (PSEUDOCODE)



IMPLEMENTATION PHASE (C++)

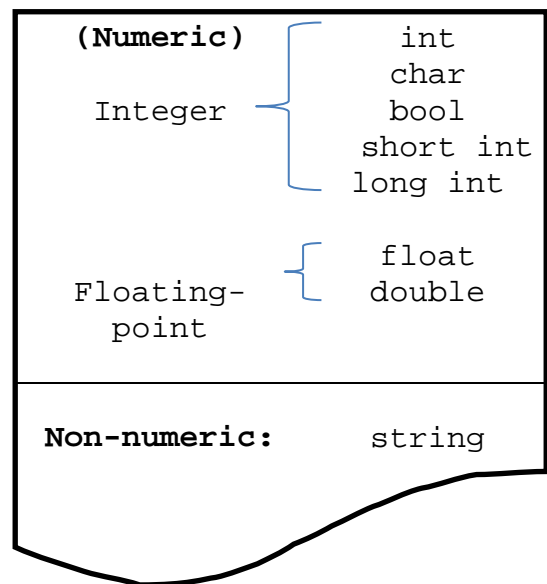


Figure 2.4

2.6.3 Declaration of variables

A Programmer declares/defines a variable by providing the name to a memory location that can be assigned a value. A value contained in a variable can be modified during the execution of the program. Every variable must be declared before use to avoid syntax errors. General syntax of declaring/defining a variable in C++:

```
Data type <NameOfVariable>;
```

E.g.:

```
int length, width ; // this is also refer to as multiple declarations
float salary;
bool registered;
string firstName;
```

Variables of the same data type can be declared using a single statement. Variables on the left-hand side of the assignment operator (=) can be initialized with one of the following:

A constant e.g.:

```
registered = true;
width = 7;
firstName = "Thabo";
```

Or

The content of another variable, e.g.:

length = width;

Or

An arithmetic or relational expression – relational expression will be discussed in learning unit 4

Salary = hrsWorked * payrate;
Area = 3.14f * radius * radius;

Or

A value entered from the keyboard using cin – this will be discussed in learning unit 3

2.6.4 Arithmetic Operators in C++

C++ supports the following binary arithmetic operators:

Operator	Description
/	Division
*	Multiplication
%	Modulus
+	Addition
-	Subtraction

Refer to page 51 to 55 of chapter 2 in the prescribed textbook for a detailed explanation of arithmetic operators and their precedence and associativity.

Let's try to re-write the example on page 12 in C++ format:

$X = A^3 - 2B + \frac{2(C+D)}{2}$	
Equation in pseudocode (one-line) format	Equation in C++ format
$X = A \wedge 3 - 2 * B + (2 * (C + D)) / 2$ <i>* Exponentiation operator (^) allowed</i>	$X = A * A * A - 2 * B + (2 * (C + D)) / 2;$ <i>* Exponentiation operator (^) and Integer division (\) are not supported in C++</i>

2.6.5 STEP BY STEP:

The source code on page 19 is for a program that will calculate the average of two tests (test1 & test2) and store it into variable finalMark. The student name and final mark must be displayed. Assign your name to the student name variable and assign any integer number to test1 and test2.

1. Open Visual Studio and type in the following source code:

```
#include <iostream> // to use cout and endl
#include <string>    //to use string data type
#include <conio.h>   //to use _getch() - to hold the screen
    Using namespace std;
int main()
{
    /*declare all variable before use, and
    If you see an error message that says:
    "Undeclared Identifier", go back to your source
    code and verify all variables the compiler
    highlighted are correctly declared*/

    //variables declaration
    int test1 , test2;
    float finalMark;
    string studentName;

    //Initialize your variables
    test1 = 57;
    test2 = 78;

    finalMark = (test1 + test2) / 2.0;

    studentName = "Mashudu";

    //display the student name and final mark
    cout << studentName
           <<" obtained a final mark of "
           <<finalMark << endl;

    //C++ consider every white space as a space bar

    _getch();
    return 0;
}
```

2. Compile and run your program on Visual Studio Command Prompt 2013

2.7 SUMMARY

In addition to these notes, study **Chapter 2, page 37 – 71** inside the prescribed textbook.

Do the following exercises:

- **Exercises 2.1** Page 47 – 48 (Theoretical & Practical exercises)
 - Do all the exercises (1 – 12)
- **Exercises 2.2** Page 55 – 56 (Theoretical & Practical exercises)
 - Do all the exercises (1 - 11)
- **Exercises 2.3** Page 66 – 70 (Theoretical & Practical exercises)
 - Do all the exercises (1 - 17)

Chapter 3

Writing Algorithms and Programs in Sequential Steps

3.1 Introduction

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

When using the first of the three structures, the **sequence structure**, all statements are executed in the sequence in which they appear. Every step is executed once, in sequential order. No step is left out, and no step is repeated. Once all the steps are executed, the problem is solved. Should the problem not be solved, then it indicates an error made in the logic of the program. It may be that a step is written incorrectly, or is in the incorrect place in the algorithm, or a step is omitted.

3.2 Program Planning

As was stated before, we cannot simply start writing a program in a programming language unless we first understand the problem. We also need to plan how to solve the problem by identifying the input and output, and then plan the processing logic by using an algorithm.

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

3.2.1 Examples

3.2.1.1 Example 1:

Problem:

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

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

- the available data (input), and
- what is needed from the program (output)

Draw the following table to identify input and output variables:

	Description	Type	Name of variable
Input:	name of student	string	name
	age of student	integer	age
Output:	name of student	string	name
	age of student	integer	age

To do better and more comprehensive planning to enable the programmer to write a complete algorithm without errors, an **IPO chart** (Input, Processing, Output) is completed. The input and output columns contain only the names of the variables as identified in the previous step, and the processing column provides an overview of the processing steps.

I	P	O
name age	Prompt to read input fields Enter name and age Show name and age on the screen	name age

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

Algorithm:

ShowDetails
~ This algorithm will enter the name and age of a student and display it on the screen
~ Enter input values
display "Provide the name of the student: " ~ display on a new line
enter name
display "Provide the age of the student: " ~ display on a new line
enter age
~ Show the details on the screen with an appropriate legend
display "The name of the student is ", name ~ display on a new line
display "The age of the student is ", age ~ display on a new line
end

The steps in the algorithm will be executed in the sequence that they are written, one after the other, and will provide the following output:

Provide the name of the student: John
Provide the age of the student: 18
The name of the student is John
The age of the student is 18

The input data that was used in this case was the name *John* and the age *18*. Other test data may be used to test the algorithm.

An algorithm has the following properties:

- Every algorithm has a descriptive name that describes the function or purpose of the algorithm. This algorithm is called *showDetails*.
- The last statement (step) is always **end**.
- The statements or instructions between the name of the algorithm and the end statement are called the **body** of the algorithm.
- ~ (tilde) indicates that a comment, which explains the code, will follow. A comment may be on a separate line or may follow in the same line as an instruction. A comment may explain difficult code, or explain why you are

including a specific instruction, or it may be any other comment you wish to include. A comment may be included anywhere in an algorithm.

- Display means that a message or the value of a variable are displayed on the screen.
- Enter indicates that data must be entered via the keyboard.
- Calculations may also be done using (an) equations(s) as in example 2 where meter and centimeter will get new values.

3.2.1.2 Example 2:

Problem:

Enter the distance between two trees in kilometers. Calculate and display the distance in meters and then calculate and display the distance in centimeters.

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

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

Note that the IPO chart does not contain detailed processing steps. It indicates **what** needs to be done, not **how** it should be done.

Algorithm:

```

CovertDistance
~   This algorithm will convert km to meter and centimeter
~   Enter input value
    display "Please provide the distance in km: "    ~ display on a new line
    enter kilometer
~   Do the conversions
    meter = kilometer * 1000
    centimeter = meter * 100
~   Display the output
    display kilometer, " km is equivalent to ", meter, " m and to ", centimeter, " cm"
    ~ display all on one line
end

```

The input data to test the algorithm will be 3 kilometers.

Output:

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

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

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

New output:

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

3.2.1.3 Example 3:

Problem:

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

	Description	Type	Name of variable
Input:	length of court	real	length
	width of court	real	width
Output:	perimeter of court	real	perim
	area of court	real	area

I	P	O
length width	Prompt for input fields Enter input fields Calculate perimeter and area Display output fields	perim area

Algorithm:

```
CalcTennisCourt
~   This algorithm will calculate perimeter and area of tennis court
~   Enter input values
    display "Please enter length of tennis court: " ~ display on a new line
    enter length
    display "Please enter width of tennis court: " ~ display on a new line
    enter width
```



```

~      Do calculations and display
      perim = 2 * (length + width)
      area = length * width
      display "The perimeter of the tennis court is ", perim, " meters"
      display "The area of the tennis court is ", area, " square meters"
end

```

The input data to test the algorithm are length = 30 and width = 10.

Output:

```

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

```

Take note: The algorithm can also accept and display real values. The length could have been 30.5 and the width 10.2 which would have resulted in the answer 81.4 meters and 311.10 square meters.

3.2.1.4 Example 4:

Problem:

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

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

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

Algorithm:

```

CalculateAverage
~      Prompt for and enter two integers
      display "Enter first number "           ~ display on a new line
      enter num1
      display "Enter second number "         ~ display on a new line
      enter num2
~      Calculate average
      average = (num1 + num2)/2
~      Show average on screen
      display "The average of the 2 numbers is ", average ~ display on a new line

```

end

If the two numbers used as input data are 24 and 40, then the output will be as follows:

Output:

```
Enter first number 24
Enter second number 40
The average of the 2 numbers is 32
```

3.2.1.5 Example 5:

Problem:

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

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

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

Algorithm:

```
CalculateWeeklyWage
~   Calculate and display the weekly wage for Jason
    display "Provide the number of hours Jason worked "           ~ display on a new line
    enter hours
    display "Provide rate of pay "                                 ~ display on a new line
    enter rate
~   Do calculations and display
    wage = hours * rate - 8
    display "After deductions Jason earned R", wage " for the week" ~ display on a new line
end
```

The input data used to test this algorithm: hours worked = 24 and rate of pay = R12.

Output:

```
Provide the number of hours Jason worked 24
Provide rate of pay 12
After deductions Jason earned R280 for the week
```

3.2.1.6 Example 6:

Problem:

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

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

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

Algorithm:

CalculateMoney		
display "Provide Tebogo's monthly salary: "		~ display on a new line
enter salary		
~ Do calculations and display		
room = 450		
remainder = salary – room		
food = remainder * 0.5		
clothes = remainder * 0.2		
transport = remainder * 0.15		
charity = remainder * 0.05		
pocket = remainder – (food + clothes + transport + charity)		
~ Display results		
display "Tebogo divided her salary as follows:"		~ display on a new line
display "Room	R", room	~ display on a new line
display "Food	R", food	~ display on a new line
display "Clothes	R", clothes	~ display on a new line
display "Transport	R", transport	~ display on a new line
display "Charity	R", charity	~ display on a new line
display "Pocket money	R", pocket	~ display on a new line
end		

The algorithm is tested with a monthly salary for Tebogo of R2 000. (Please note that the units (R) is not included in the calculation, and the space – a thousand separator – also).

Output:

```
Provide Tebogo's monthly salary: 2000
Tebogo divided her salary as follows:
Room           R450.00
Food           R775.00
Clothes        R310.00
Transport       R232.50
Charity         R77.50
Pocket money    R155.00
```

3.3 Predicting the output of an algorithm

It is often necessary to determine the exact outcome of a set of instructions in an algorithm. Usually these statements include tricky calculations and therefore the programmer needs to ensure that the calculations have been done correctly. The programmer must also show what results to expect. For this we use a method, which is explained at the hand of two examples:

3.3.1 Example 1

```
firstExample
    A = 3
    B = 7
    A = B - A + 2
    C = A ^ 2 + B - 5 * 2
    Display "The value of C is ", C
end
```

Solution:

Draw a table that contains a column for each variable used in the calculation:

A	B	C

After the first statement is executed, the value of A will be 3:

A	B	C
3		

After the next statement is executed, the value of B will be 7:

A	B	C
3		
	7	

Now the first calculation must be done to determine the new value of A:

$A = B - A + 2$
 Substitute the current values of A and B:
 $A = 7 - 3 + 2$
 Therefore $A = 6$
 A gets a new value of 6, replacing the previous value of 3

Strike through the 3 in the table in order to indicate that A now has a new value.

A	B	C
3		
	7	
6		

Now the second calculation must be done to determine the value of C:

$C = A^2 + B - 5 * 2$
 Substitute the current values of A and B:
 $C = 6^2 + 7 - 5 * 2$
 Therefore C gets a value of 33

A	B	C
3		
	7	
6		
		33

The last statement:

`display "The value of C is ", C`
 will first display the content of the legend followed by the value of the variable as indicated below:

The value of C is 33

This will be the answer to the question because it is the only output displayed by the algorithm. The table assisted us only in calculating the values of the variables while the statements were executed.

3.3.2 Example 2

Use the same set of instructions as in Example 1, but change the display statement to:

```
display "The value of A plus B is ", A + B
```

Then the output will now be:

The value of A plus B is 13

The value of A plus B ($6 + 7 = 13$, as can be seen in the table), will be calculated before the result is displayed.

3.3.3 Example 3

Use the same set of instructions again, but change the display statement to:

```
display "A = ", A, " B = ", B, " C = ", C
```

Then the output will now be:

A = 6 B = 7 C = 33

Please note that the output contains only the results that are displayed on the screen of the computer. **It does not contain any of the calculations.**

Study the display statements closely to see how spaces are inserted between other output characters.

Note that if the display statement does not indicate that the output must be displayed on a new line, it must continue on the previous line e.g.

```
display "The answer is "           ~ on a new line
display " the sum of 2 numbers"
```

The following will be displayed:

The answer is the sum of 2 numbers

Design exercises:

5. Thandeka bought a number of widgets to re-sell at a profit. Enter the number of widgets and the cost price of one widget. Unfortunately, 5% of the widgets broke, but she sold all the other widgets at cost price + 25%. Calculate and show the total profit on the computer screen. You may assume that there will not be a loss.

6. **Harry's Rent-A-Car** is a budget car rental company who wants you to write a program that will calculate and display the amount payable by a customer when renting a car. The basic cost is R450 per day, plus a specific amount per kilometer driven, in cents, to be entered by the owner of the company. Vat of 14% must be added. The number of days rented, the initial kilometer reading and final kilometer reading must also be entered.
7. The basic price for a phone call of three minutes is 50 cents. Enter the duration of a phone call in minutes. For every minute above the basic three minutes, an additional amount of 15 cents per minute is charged. Add 14% VAT. Calculate and display the price of the phone call. You may assume that the call will be more than 3 minutes.
8. The **Elgin Packing Company** is responsible for packing apples in boxes. They need a program to determine for them how many boxes will be needed to pack the apples of a specific farm. The apples are packed on polystyrene layers. Each layer takes 25 apples. Six layers can fit into a box. Enter the number of apples from a farm, and determine the number of polystyrene layers and boxes needed to pack the apples for that farm. Only full boxes are distributed. Also display the number of apples left over, that could not be packed.

3.4 GETTING STARTED IN C++

3.4.1 ASSIGNMENT STATEMENT (=)

So far we have learned how to declare a variable, so let's look at how to assign a value to it. C++ provides us with two ways to do this:

- Using the C++ assignment statement
- Using the input (read) statement

C++ assignment statement takes the following syntax:

```
Variable = expression;
```

The value of the expression in an assignment statement should match the data type of the variable. Fortunately, C++ provides coercion on primitive/built-in data types, i.e. the expression on the right side is evaluated, and its value is converted to the data type of the variable on the left side before assigning a value.

A variable is said to be initialized if at first a value is placed in the variable. Suppose you have the following variable declarations:

```
int firstNum, secondNum;  
double sales;  
float price;  
char code;  
string subject;
```

Now let's look at possible assignment statement for each variable:

```
firstNum = 15;      //assignment statement assigns the integer 15 to variable firstNum
secondNum = 15 + 16 / 8;
//the expression on the right side will be evaluated first before the value is assigned to a variable named
secondNum on the left side.
sales = 15000 * 0.5;
//the expression on the right side will be evaluated first before the value is assigned to a variable named
sales on the left side

price = 299.99f;
//the expression on the right side will be converted into a float data type before it is assigned to a
variable on the left side.
code = 'C';        //assignment statement assigns character C to variable code. Note that character
values are enclosed in single quotes.
subject = "Development Software 1A";
//assignment statement assigns the string "Development Software 1A" to the variable called subject.
String values are enclosed in double quotes.
```

Now let's look on how we can use the above assignment statements in a complete C++ program:

```
#include <iostream>
#include <string>
using namespace std;
```

```
int main()
{
    int firstNum, secondNum;
    double sales;
    float price = 299.99;
    char code;

    //plus a Boolean variable
    bool member;
    string subject;

    firstNum = 15;
    cout << "The first number = " << firstNum << endl;

    secondNum = 15 + 16 / 8;
    cout << "The second number = " << secondNum << endl;

    sales = 15000 * 0.5;
    cout << "Sales = R" << sales << endl;

    code = 'C';
    cout << "Code = " << code << endl;
```

Program Output:

```
The first number = 15
The second number = 17
Sales = R7500
Code = C
Membership = 1
Development Software 1A
Price = R299.99
```


<pre> member = true; cout << "Membership = " << member << endl; subject = "Development Software 1A\n"; cout << subject; price = 299.99f; cout<<"Price = R"<<price<<endl; return 0; } </pre>	
--	--

Only a variable is allowed on the left side of the equal sign in the assignment expression.

A variable that contains an initial value can be used on both sides of the equals sign.
Example: sum = sum + 5;

In the above assignment expression, the value of sum + 5 on the right side will first be calculated and thereafter stored into sum to replace the previous value in sum.

3.4.2 FORMATTING NUMERIC OUTPUT

In a C++ program, real numbers are displayed in either fixed-point or exponential (e) notation, depending on the size of the number. Smaller real numbers—those containing six or fewer digits to the left of the decimal point—usually are displayed in fixed-point notation. C++ contains the following stream manipulators which are found in the *#include <iomanip>* directive to format real number in the display statement.

Stream manipulator	Action/description
setw(n)	Set the field width to n The n can only be an integer value If the current field width is more than the n then this manipulator is ignored.
fixed	Display a decimal-point and uses a default of six digits after the decimal point If the real number contains less than six decimal digits, the fixed manipulator pads the number with zeros until it has six decimal places.
setprecision(n)	Set the total number of digits to be displayed after a decimal point The n can only be an integer value The setprecision remains in effect either until it encounters another setprecision manipulator or until the end of the program This manipulator does not modify the value stored in a variable

Now let's look at an example that includes all three stream manipulators discussed in the table above:

<pre>#include <iostream> #include <iomanip> using namespace std; int main() { double salary = 0.0; double rate = 199.99; int hourWorked = 55; salary = hourWorked * rate; cout << "New salary: R" << setw(7) << fixed << setprecision(3)<<salary << endl; system("pause"); return 0; }</pre>	<p>Program Output:</p> <p>New salary: R10999.450</p> <p><i>*The setw manipulator will be ignored</i></p>
---	--

Also check examples on how to use the setw and effects of manipulators on page 95 – 97 inside the prescribed textbook.

- Program 3.5
- Program 3.6 and
- Table 3.2

3.4.3 TYPE CONVERSIONS (CASTS)

So far we have seen that the computer will either promote or demote the value on the right side of the assignment statement to match the data type of the variable on the left side. This is known as implicit type conversions. C++ provides an explicit type conversion or a type cast to allow the programmer to convert data from one data type to another during compile time.

Syntax:

datatype (expression)

Now let's look at an example for type conversion:

```
int x = 3, y = 2;
float z;
z = (float (x) / y); //the value of variable x is converted into float before dividing it
                    //with the value of variable y and assigns 1.5 in to variable z
```

3.4.4 INTERACTIVE KEYBOARD INPUT IN C++

A program performs three basic operations:

- It gets data (from keyboard, scanners, etc.),
- It manipulates the data (through CPU), and
- It outputs the results (screen or printer).

C++ provide two user-defined variables, namely, `cout` (pronounced "see-out"), which stands for common output, and `cin` (pronounced "see-in"), which stands for common input. Both variables are declared in the **`iostream`**. We have already seen how the `cout` variable works.

This is how the `cin` works:

- It tells the computer to pause the program's execution while the user enters one or more characters from the keyboard.
- Then it temporarily stores the characters as they are typed.
- Then it extracts characters using extraction operator (`>>`) from the `cin` variable and sends them "in" to the programmer's declared variable.

Syntax of using `cin`:

```
cin >> variableName;

//read in multiple data using a single cin is permitted in C++

cin >> variableName1 >> variableName2;
```

To develop a user friendly program, the prompt must be displayed using the `cout` variable to describe what must be entered before the data is read by `cin`.

Now let's modify the program we created earlier to include the `cin`:

```
/* This program read in the hours worked and pay rate and calculate and display
salary */
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double salary;
    double rate;
    int hourWorked;

    cout << "Please provide the hours worked: ";
    cin >> hourWorked;
    cout << "Please provide the pay per hour: ";
    cin >> rate;
    salary = hourWorked * rate;

    cout << "New salary: R" << setw(7) << fixed
         << setprecision(3) << salary << endl;

    system("pause");
    return 0;
}
```

Program Output:

```
Please provide the hours worked: 45
Please provide the pay per hour:
175.80
New salary: R 7911.000
```

**The `setw` manipulator will be ignored*

Open notepad or notepad++ and type examples on how to use the interactive input on page 118 – 120 inside the prescribed textbook.

- Program 3.12
- Program 3.13

3.4.5 SYMBOLIC CONSTANTS

A constant stores a value that does not change throughout the execution of the program. For good programming practice, a programmer must declare constants for values that repeatedly appear in the same program to allow easy maintenance of the program. For example, if any of the constants' value has to change in the future, the programmer will only change it once in the declaration statement.

C++ syntax for declaring constants variable:

```
const <datatype> CONSTANTNAME = value;
```

Examples:

```
const int TOTAL_MINUTE = 60;  
const float TAX = 0.14f;
```

*For good programming practice when declaring constants variables all letters must be in uppercase.

Now let's do the implementation of the algorithm in Example 1 and 2 (on pages 2 - 5) in C++:

Example 1

Algorithm

ShowDetails

```
~   This algorithm will enter the name and age of a student and display it on the screen
~   Enter input values
    display "Provide the name of the student: "    ~ display on a new line
    enter name
    display "Provide the age of the student: "      ~ display on a new line
    enter age
~   Show the details on the screen with an appropriate legend
    display "The name of the student is ", name    ~ display on a new line
    display "The age of the student is ", age      ~ display on a new line
```

end

Corresponding C++ Code

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    //This program will enter the name and age of a student and display it on the screen
    string name;
    int age;
    //Enter input values

    cout << " Provide the name of the student: "<< endl;
    cin >> name;

    cout << " Provide the age of the student: " << endl;
    cin >> age;

    //Show the details on the screen with an appropriate legend
    cout << " The name of the student is " << name << endl;
    cout << " The age of the student is " << age << endl;
    return 0;
}
```

Example 2

Algorithm

CovertDistance

```
~   This algorithm will convert km to meter and centimeter
~   Enter input values
    display "Please provide the distance in km: "    ~ display on a new line
    enter kilometer
~   Do the conversions
    meter = kilometer * 1000
    centimeter = meter * 100
~   Display the output
    display kilometer, " km is equivalent to ", meter, " m and to ",
        centimeter, " cm"    ~ display all on one line
```

end

Corresponding C++ Code

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    //This program convert km to meter and centimeter
    int kilometer;
    int meter;
    int centimeter;

    //Enter input value
    cout << " Please provide the distance in km: "<< endl;
    cin >> kilometer;

    // Do the conversions
    meter = kilometer * 1000
    centimeter = meter * 100

    // Display the output
    cout << kilometer <<" km is equivalent to " << meter
        << " m and to " << centimeter <<" cm" << endl;

    return 0;
}
```

Now we have seen how easy it is to transfer a clear and accurate algorithm to C++. Please re-write Example 3 – 6 in C++ and test your program with values which were used in the design.

3.5 SUMMARY

In addition to these notes, study the following pages on **Chapter 3** inside the prescribed textbook.

- **Page 79 – 85** (Assignment operators)
- **Page 93 – 106** (Only study stream manipulators discussed in the notes)
- **Page 111 – 134** (Casts – Common programming errors)

Do the following exercises:

- **Exercises 3.1** Page 90 - 93
 - Do all the exercises (1 – 16)
- **Exercises 3.2** Page 103 – 106
 - Do only exercises (1, 5 – 8)
- **Exercises 3.3** Page 112 – 116
 - Do only exercises (2 (a – i) and 5)
- **Exercises 3.4** Page 122 – 127
 - Do only exercises (1 – 8)
- **Exercises 3.5** Page 129 – 131
 - Do all the exercises (1 – 5)

Chapter 4

The Selection Control Structure

4.1 Introduction

Any program can be written by using a combination of three basic control structures: sequence, selection and iteration. In this chapter we discuss the second of the three basic control structures, the **selection (decision) control structure**.

The complexity of a problem may be of such a nature that it cannot be solved by making use only of sequential statements. An algorithm may therefore include the possibility to provide for different options that could change the flow of the statements in the algorithm. To enable the programmer to include this type of logic in an algorithm, a selection (decision) control structure, which is based on relational and logical operators, must be used.

4.2 Relational operations

Computers have the ability to compare two values. Expressions that compare operands (values or variables) are called relational expressions. The outcome or result of such a comparison always yields a Boolean value – that is, true or false.

Examples:

6 > 4? result: true
12=7? result: false

Description	Pseudocode Symbol	Example
Equal to	=	number = 5
Greater than	>	quantity > 35
Less than	<	temp < minTemp
Not equal to	<>	code <> "sd"
Greater or equal to	>=	mark >= 50
Less or equal to	<=	points <= 500

All these **relational operators** have equal priority, but the arithmetic operators have a higher priority than the relational operators.

Examples:

Determine whether the following expression is true or false:

1. $b \leq a - 3$ where $b = 5$ and $a = 7$

Substitute:

$$5 \leq 7 - 3$$

Answer to calculation:

$$5 \leq 4$$

Outcome of relational comparison:

false

2. $m + 7 > n - 3 * p$ where $m = 4$, $n = 12$ and $p = 5$

Substitute:

$$4 + 7 > 12 - 3 * 5$$

Answer to first calculation: (* higher priority than +)

$$4 + 7 > 12 - 15$$

Answer to second calculation: (from left to right)

$$11 > 12 - 15$$

Answer to third calculation:

$$11 > -3$$

Outcome of relational comparison:

true

Exercises:

Evaluate the following expressions:

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

Note

In computing and mathematics, the term “evaluate” is often used to mean “find the result” – for example, the equation **evaluates** to zero or another value, or it evaluates to true or false.

4.3 Logical operations

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

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

Example:

To illustrate the OR, suppose a person needs either an ID document or a driver's license, but not both, to enter an event (outcome to be true). However, to illustrate the AND, if the person needs to provide an ID document as well as a letter of approval before entering the event, both documents must be provided before the person may enter.

The order of precedence is indicated in the last column – that is, processing is done in the following order:

- NOT
- AND
- OR

If two of the same operator appears in the same expression, it will be executed from left to right.

When different operators are combined in one expression, all the mathematical operators will be processed first, then the relational operators and finally the logical operators.

The order can be changed by using parenthesis. For example, if an OR must be processed before an AND, parenthesis can be inserted.

Examples

Evaluate the following expressions:

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

Substitute:

TRUE OR FALSE AND NOT TRUE
 TRUE OR FALSE AND FALSE
TRUE OR FALSE
TRUE

2. $M > 7 \text{ AND } D < S \wedge 2$ where $M = 7, D = 8, S = 4$

Substitute:

$$7 > 7 \text{ AND } 8 < 4 \wedge 2$$

$$7 > 7 \text{ AND } 8 < 16$$

$$\text{FALSE AND } 8 < 16$$

$$\text{FALSE AND TRUE}$$

$$\text{FALSE}$$

3. $D < (E + 7) \text{ AND } G \text{ OR } (A * C) \geq (D - E + C)$
 where $A = 5, C = 12, D = -15, E = -17, G = \text{FALSE}$

Substitute:

$$-15 < (-17 + 7) \text{ AND } \text{FALSE OR } (5 * 12) \geq (-15 - -17 + 12)$$

$$-15 < -10 \text{ AND } \text{FALSE OR } (5 * 12) \geq (-15 - -17 + 12)$$

$$-15 < -10 \text{ AND } \text{FALSE OR } 60 \geq (-15 - -17 + 12)$$

$$-15 < -10 \text{ AND } \text{FALSE OR } 60 \geq (-15 + 17 + 12)$$

$$-15 < -10 \text{ AND } \text{FALSE OR } 60 \geq (2 + 12)$$

$$-15 < -10 \text{ AND } \text{FALSE OR } 60 \geq 14$$

$$\text{TRUE AND FALSE OR } 60 \geq 14$$

$$\text{TRUE AND FALSE OR TRUE}$$

$$\text{FALSE OR TRUE}$$

$$\text{TRUE}$$

Exercises:

Evaluate the following expressions where $A = 5, B = 7, C = -4, D = 12,$
 $E = \text{TRUE}$ and $F = \text{FALSE}$

1. $A * 3 \geq B - 14 \setminus 3 \text{ AND } F \text{ OR } C \leq D + 3$
2. $\text{NOT } F \text{ OR } E \text{ AND } D + C = A$
3. $(C - D) * 2 <> A + B \text{ OR } E \text{ AND } F \text{ OR } B > D + B \wedge 2$
4. $D = C \text{ MOD } 5 \text{ AND } \text{NOT } F \text{ OR } \text{NOT } (A + D \leq A - D)$

4.4 The simple if-statement

Let us explain this option by using the following example:

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

Pseudocode:

```

~ Test if the employee worked overtime
if hours > 45 then
    wage = wage + 300
endif
  
```

Explanation of the above two lines:

```
if hours > 45 then
```

 This statement tests if the hours worked is more than 45.

```
wage = wage + 300
```

 If the test is true (the employee has worked more than 45 hours), the previous wage is replaced by an increase of R300 on the previous value.

```
endif
```

 This line ends the if-structure and the execution of statements will continue after the endif

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

The syntax of the if-statement in an algorithm:

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

Example:

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

Please note:

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

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

```
if NOT condition then
    statement 1
    statement 2
    .
    .
    statement n
endif
```

```

Example:   if mark NOT < 50 then
            display "Student have passed"
        endif

```

4.5 Testing an algorithm

After an algorithm has been written, the programmer may not assume that it is error-free and produces the correct output in a manner that is easy to understand and read. It is important to test the algorithm for correctness and readability.

In order to accomplish the desires and correct result, the programmer must compose a set of test data to test the algorithm. The test data must include all possibilities of correct and incorrect data. The test data must be compiled in such a way that they resemble the possible real data. At this stage we are going to use simple test data, but more complicated programs will need more test data to ensure an error-free algorithm.

The algorithm can be tested by using a trace table (desk checking). Later the same test data will be used to test the actual program.

Study the following program:

Problem:

Lerato bakes cakes that she sells per slice. She cuts 10 slices per cake. She calculates the amount spent on the ingredients, electricity and packaging material. She adds 30% to the amount spent and then she calculates the price of a slice of cake. She always sells the slice in full rand amounts. For example, if the calculated price is R3.25, then the selling price is R4. In other words, cents are always rounded up to the next rand.

	Description	Type	Name of variable
Input:	cost of cake	real	expense
Intermediate:	cake selling price	real	cakePrice
	price per slice	real	slicePriceDecimal
Output:	price per slice	integer	slicePriceInteger

I	P	O
expense	Prompt for expense Enter expense Calculate slice price Display slice price	slicePriceInteger

Algorithm:

```
CalcCakePrice
~   Calculate the selling price of a slice of cake
    display "Provide the cost of the cake"
    enter expense

    ~ Add profit and calculate the price per slice
    cakePrice = expense + expense * 0.3    ~ intermediate variable
    slicePriceInteger = cakePrice \ 10
    slicePriceDecimal = cakePrice / 10

    if slicePriceInteger <> slicePriceDecimal then
        slicePriceInteger = slicePriceInteger + 1
    endif

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

Test the algorithm by using a manual calculation:

Use as test data:

Expense R18.60

Increase by 30% R24.18

Gross price per slice R2.418

Integer division results in R2 per slice of cake

Because there is a remainder, R1 will be added to the result to give the selling price of the slice of cake.

Selling price per slice R3

Trace table:

Instruction	expense	cakePrice	slicePriceInteger	slicePriceDecimal	Outcome of if	output
display						Provide the cost of the cake
enter	18.60					
calculate		24.18				
calculate			2			
calculate				2.418		
if					TRUE	
calculate			3			
display						The price of a slice of cake is R3

The instructions are listed in the leftmost column, followed by the variables used, the outcome of the if-statement and finally the output, as shown in the following table:

Output:

Provide the price of the cake 18.60 The price of a slice of cake is R3

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

4.6 Examples of simple if-statements

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

Example 1

Problem:

John bought a number of pencils at a given price. The user must be asked to enter values for these variables. If he buys 25 or more pencils, he receives a discount of 7.5%. Calculate and display the amount due by John.

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

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

Algorithm:

```
CalcAmount
~   Calculate the amount due for pencils
    display "Provide the number of pencils bought" ~ Display on a new line
    enter number
    display "Provide the price of one pencil" ~ Display on a new line
    enter price

    ~ Processing
    amount = number * price

    ~ Test if John bought 25 or more pencils
    if number >= 25 then
        amount = amount - amount * 0.075
    ~ The calculation could also be: amount = amount * 0.925
    endif

    ~ Display the output
    display "The amount due is R", amount ~ Display on a new line
end
```

Test the program twice using:

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

Output:

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

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

Example 2

Problem:

Jenny wants to buy a packet of chips for every child in the playgroup. The salesperson at the shop tells her how many packets are in stock. Enter the number of children, the number of packets and the selling price of a packet. If the shop has a packet for every child, calculate and display the amount due on the screen. If there are not enough packets, Jenny is not going to buy anything and no message is displayed.

	Description	Type	Name of variable
Input:	number of children	integer	numChildren
	number of packets in stock	integer	numPackets
	selling price	real	sellingPrice
Output:	amount due	real	amntDue

I	P	O
numChildren numPackets	Prompt to read input Enter input values Calculate amntDue Display amount	amntDue

Algorithm:

```

CalcAmntDue
~   Calculate the amount due for chips
    display "Provide the number of children"      ~ Display on a new line
    enter numChildren
    display "Provide the number of packets"      ~ Display on a new line
    enter numPackets
    display "Provide the selling price of a packet" ~ Display on a new line
    enter sellingPrice

    ~ Processing
    if numPackets >= numChildren then
        amntDue = sellingPrice * numChildren
        display "The amount due is R", amntDue
    endif
end

```

Test the program twice using:

numChildren = 12, numPackets = 10, sellingPrice = R5.70
 numChildren = 8, numPackets = 15, sellingPrice = R6.50

Output:

```

Provide the number of children 12
Provide the number of packets 10
Provide the selling price of a packet 5.70

```

No calculation is done, and nothing is displayed as output, as 12 is greater than (or equal to) 10.

Provide the number of children 8
Provide the number of packets 15
Provide the selling price of a packet 6.50
The amount due is R52

Design exercises

1. A grocery store wants to increase all items in the store by 5%, except those in the perishable department, department 3, that must be increased by 7%. Enter the price of an item and the department number it falls in. Calculate the new price of the item and display it on the screen.
2. A number of people go to a Parlotones Concert. The price of a ticket is R550, but if a group of more than 35 wants to buy tickets, the price is only R500 per ticket. Enter the number of people in the group. Calculate and display the total amount for all the tickets.
3. Determine how much Lizzie has to pay for a number of T-shirts. Enter the number of T-shirts bought and the price of a T-shirt. Calculate the amount due. If this amount is more the R500, Lizzie gets a discount of 8%. Display the amount due on the screen.
4. Henry rents a bakkie to move stock to a store. The basic cost is R300 per day. He also pays a specific amount per kilometer travelled. If the number of kilometers exceeds 400, het gets a discount of 5% on the amount for the kilometers travelled. Enter the number of kilometers travelled, as well as the number of days that he rented the trailer, and calculate the amount due by him.

4.7 THE IF STATEMENT IN C++

We have seen how a computer processes a C++ program with only sequential control structure statements, so now let's see how a selection control structure is processed.

In C++ a simple If-statement (also called a single-alternative or one-way selection structure) can be written using the following syntax:

```
if (expression)
    statement;
```

Simple If-statement that executes a single statement

Please note:

- The selection structure must begin with the reserved word **if**
- This must be followed by **parentheses** around the expression
- The **expression** is sometimes called a **decision maker** because it decides whether to execute the statement that follows it.

The expression is a Boolean expression. If the end value of the expression is **true (1)** then the statement that follows the expression will execute, however if the value of the expression is **false (0)** then the computer skips the first statement and execute the next one. **Curly brackets** can be used whenever there is more than one statement to be executed based on the value of the expression, in such case the syntax will be as follows:

```
if (expression)
{
    Statement 1;
    Statement 2;
    :
    Statement n;
}
```

Simple If-statement that executes multiple statements

Any variable declared within a block has meaning only between its declaration and the closing braces of the block.

Relational Operators in C++

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

All these C++ relational operators are binary operators, and are always evaluated to either true (1) or false (0).

Please note:

In C++, the symbol ==, which consists of two equal signs, is called the equality operator. Recall that the symbol = is called the assignment operator. Remember that the equality operator, ==, determines whether two expressions are equal, whereas the assignment operator, =, assigns the value of an expression to a variable.

Example of if-statements in C++

Consider the following variables:

```
int month = 4;
double newPrice = 95.99;
double oldPrice = 89.50;
char letter = 'm';
string name = "Mary";
bool isStudent = false;
```

if (month < 6) cout<< "New Born\n";	if (4 < 6) if (true) Then the cout-statement will execute
if (newPrice < oldPrice * 1.1) cout<<"The new price was increased by " <<"less than 10%\n";	if (95.99 < 89.50 * 1.1) if (95.99 < 98.45) if (true) Then the cout-statement will execute
if (letter > 'j') cout<<"The letter comes after j\n";	if ('m' > 'j') if (true) Then the cout-statement will execute
if (name != "Mpho") cout<<"You are not Mpho\n";	If ("Mary" != "Mpho") If (true) Then the cout-statement will execute
if (isStudent) cout<<"You are still a student\n";	If (false) Then the cout-statement will not execute

COMPARING CHARACTERS

C++ uses the ASCII values, as indicated in the following table, when comparing characters.

ASCII Table															
Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(72	48	110	H	104	68	150	h
9	9	11		41	29	51)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Now let's look at the previous example

```
char letter = 'm';  
if (letter > 'j')  
    cout<<"The letter comes after j\n";
```

Therefore:

```
if ('m' > 'j')
```

The ASCII value of 'm' is 109, while the ASCII value of 'j' is 106

```
if (109 > 106 )
```

```
if (false)
```

Other Examples:

```
'a' < 'A' = false
```

```
'R' > 'T' = false
```

```
'd' >= 'c' = true
```

COMPARING STRINGS

C++ compares strings character by character using ASCII values.

Example:

```
string name = "Mary";  
if (name != "Mpho")  
    cout<<"You are not Mpho\n";
```

Therefore:

```
if ('M','a','r','y' != 'M','p','h','o')
```

ASCII values

```
if (77,97,'r','y' != 77, 112, 'h', 'o')
```

- "Mary" is not equals to "Mpho",
- "Mary" is less than "Mpho"

C++ LOGICAL OPERATORS

Representation of logical operators in C++:

Operator	Description
!	NOT
&&	AND
	OR

The operator ! is unary, so it has only one operand. The operators && and || are both binary operators.

Now let's implement example 1 and 2 on pages 9 – 11 in C++:

EXAMPLE 1

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    int number;
    double price, amount;

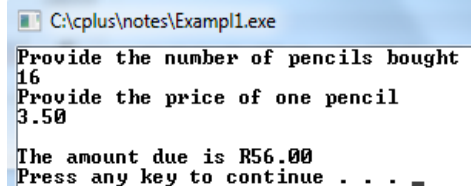
    //Calculate the amount due for pencils
    cout<<"Provide the number of pencils bought\n";
    cin>>number;
    cout<<"Provide the price of one pencil\n";
    cin>>price;

    //Processing
    amount = number * price;

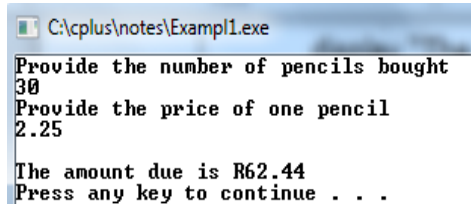
    //Test if John bought 25 or more pencils
    if (number >= 25)
        amount = amount - amount * 0.075;
    /*The calculation could also be:
    amount = amount * 0.925    */

    cout<<fixed<<setprecision(2);
    cout<<"\nThe amount due is R"<<amount<<endl;
    system("pause");
    return 0;
}
```

Sample output:



```
C:\cplus\notes\Examp1.exe
Provide the number of pencils bought
16
Provide the price of one pencil
3.50
The amount due is R56.00
Press any key to continue . . . _
```



```
C:\cplus\notes\Examp1.exe
Provide the number of pencils bought
30
Provide the price of one pencil
2.25
The amount due is R62.44
Press any key to continue . . .
```

EXAMPLE 2

```
#include <iostream>
#include <iomanip>

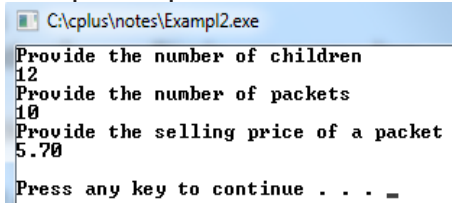
using namespace std;

int main()
{
    int numChildren, numPackets;
    float sellingPrice, amntDue;

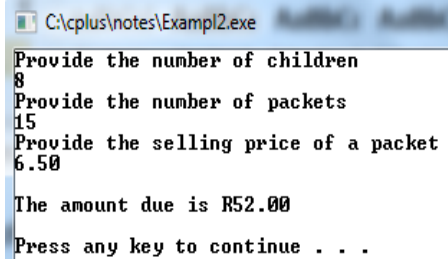
    //Calculate the amount due for chips
    cout<<"Provide the number of children\n";
    cin>>numChildren;
    cout<<"Provide the number of packets\n";
    cin>>numPackets;
    cout<<"Provide the selling price of a packet\n";
    cin>>sellingPrice;

    //Processing
    if (numPackets >= numChildren)
    {
        amntDue = sellingPrice * numChildren;
        cout<<fixed<<setprecision(2);
        cout<<"\nThe amount due is R"
            <<amntDue<<endl;
    }
    cout<<endl;
    system("pause");
    return 0;
}
```

Sample output



```
C:\cplus\notes\Examp12.exe
Provide the number of children
12
Provide the number of packets
10
Provide the selling price of a packet
5.70
Press any key to continue . . . _
```



```
C:\cplus\notes\Examp12.exe
Provide the number of children
8
Provide the number of packets
15
Provide the selling price of a packet
6.50
The amount due is R52.00
Press any key to continue . . .
```

4.8 The if-then-else statement

With the simple-if statement, the statement or statements following the if-header will be executed when the condition is TRUE, but nothing will be executed if the condition is not TRUE, but FALSE. It is also possible to have a statement or statements executed when the condition is FALSE, by adding an else-clause to the if statement:

```
if condition then
    statement(s)           ~ executed when true
else
    other statement(s)     ~ executed when false
endif
```

Example:

Suppose all the employees in a company receives an increase 8%, but the employees from Department C receive an increase of 10%, as they performed better than the others. We can accomplish this by writing the following if-statement that contains an else-clause:

if dept = "C" then	
increase = salary * 0.1	~ 10% increase applies
else	
increase = salary * 0.08	~ 8% increase applies
endif	

Note the indentation of the lines. This enhances the readability of the program.

4.9 Examples of if-then-else statements

Example 1

Problem

Abel and Fred take part in the final round of a competition. One rule of the competition is that judges may not assign the same marks to two competitors. Enter the number of points that each competitor obtains, compare the points and determine who the winner is. Display the name of the winner, as well as the winner's name on the screen.

Planning:

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

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

Algorithm:

```

DetermineWinner
~   Determine the winner of the competition
    display "Provide the points for Abel"           ~ Display on a new line
    enter pointsA
    display "Provide the points for Fred"           ~ Display on a new line
    enter pointsF

    ~ Find the winner
    if pointsA > pointsF then
        pointW = pointsA
        winner = "Abel"
    else
        pointW = pointsF
        winner = "Fred"
    endif

    ~ Display the result on a new line
    display "The winner is", winner, " with ", pointW, " points"
end

```

Test the program

Abel had 127 points and Fred had 134 points.

Output:

```

Provide the points for Abel 127
Provide the points for Fred 134
The winner is Fred with 134 points

```

Abel had 152 points and Fred had 144 points.

Output:

```

Provide the points for Abel 152
Provide the points for Fred 144
The winner is Abel with 152 points

```

Example 2

Problem

A customer bought some items at a store. Enter the amount payable. If the amount is R550 or more, a discount of 7% applies, but if it is less only a discount of 4.5% applies. Calculate and display the discount amount, as well as the final amount payable.

Planning:

	Description	Type	Name of variable
Input:	Amount payable	real	amntPayable
Output:	Discount	real	discount
	Final amount payable	real	finAmntPayable

I	P	O
amntPayable	Prompt for amount payable Enter amount payable Determine discount Determine final amount payable Display discount and final amount payable	discount finAmntPayable

Algorithm:

```
DetermineDiscount
~ Determine the discount and final amount payable
  display "Provide the amount payable "      ~ Display on a new line
  enter amntPayable

  ~ Caculate the discount
  if amntPayable >= 550 then
    discount = amntPayable * 0.07
  else
    discount = amntPayable * 0.045
  endif

  ~ Calculate the final amount payable
  finAmntPayable = amntPayable – discount

  ~ Display the results, each on a new line
  display "The discount applicable is R", discount
  display "The final amount payable is R", finAmntPayable
end
```

Test the program

Amount payable is R635.50

Output:

```
Provide the amount payable 635.50
The discount applicable is R44.49
The final amount payable is R591.01
```

Amount payable is R380.75

Output:

```
Provide the amount payable 380.75
The discount applicable is R17.30
The final amount payable is R363.45
```

Exercise:

1. In each of the following cases, do the planning and write an algorithm to solve the following problems:
 - 1.1 The ABC Cell Phone Company charges customers a basic rate of R60 per month to send text messages. The first 100 messages per month, an additional 5c per message is charged. For all messages more than 100, an additional 10c is charged. Enter the number of messages texted by a customer in a month, and calculate and display the bill amount for the customer.
 - 1.2 Enter an employee's salary, and determine and display the tax payable by the employee. If the employee earns less than R20 000 per month, the percentage tax is 18%, and for all other salaries the percentage tax is 27%.
2. Write down only the exact and correct output that will be displayed after each of the following group of statements were processed:

```
x = 4
y = x + 2
display "Here is the output"           ~ display on a new line
x = x * 2 - y / 3
y = x * 2
if x >= 8 then
    display "The value of x = ", x      ~ On new line
else
    x = y - 3
    display "The value of x = ", x      ~ On a new line
endif
w = x + y * 0.5
display x, " ", y, " ", w              ~ display on a new line
```

4.10 IF-ELSE STATEMENT IN C++

In C++ the If-Else statement (also called Two-way selection structure) can be written using the following syntax. Should more than one statement be executed when the condition is either true or false, the statements must be included in curly braces {}.

```
if (expression)
    statement1;
else
    statement2;
```

Now let's implement example 1 and 2 of 4.6 in C++:

Example 1

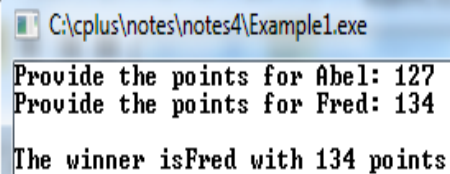
```
#include <iostream>
#include <conio.h>
#include <string>

using namespace std;

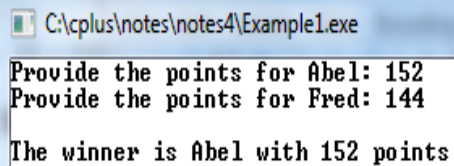
int main()
{
    //Determine the winner of the competition
    int pointsA, pointsF;
    int pointW = 0;
    string winner="";

    cout << "Provide the points for Abel: ";
    cin >> pointsA;
    cout << "Provide the points for Fred: ";
    cin >> pointsF;

    // Find the winner
    if (pointsA > pointsF)
    {
        pointW = pointsA;
        winner = "Abel";
    }
    else
    {
        pointW = pointsF;
        winner = "Fred";
    }
    //Display the result
    cout << "\nThe winner is " << winner
         << " with " << pointW << " points\n";
    _getch();
    return 0;
}
```



```
C:\cplus\notes\notes4\Example1.exe
Provide the points for Abel: 127
Provide the points for Fred: 134
The winner is Fred with 134 points
```



```
C:\cplus\notes\notes4\Example1.exe
Provide the points for Abel: 152
Provide the points for Fred: 144
The winner is Abel with 152 points
```

Example 2

```
#include <iostream>
#include <conio.h>
#include <iomanip>

using namespace std;

int main()
{
    /*Determine the discount and final
    amount payable*/
    double amntPayable, finAmntPayable;
    double discount = 0.0;

    cout << "Provide the amount payable: ";
    cin >> amntPayable;

    //Calculate the discount
    if (amntPayable >= 550)
        discount = amntPayable * 0.07;
    else
        discount = amntPayable * 0.045;

    //Calculate the final amount payable
    finAmntPayable = amntPayable - discount;

    // Display the result
    cout << fixed << setprecision(2);
    cout << "The discount applicable is R"
        << discount << endl;
    cout << "The final amount payable is R"
        << finAmntPayable << endl;

    _getch();
    return 0;
}
```

C:\cplus\notes\notes4\Example2.exe

```
Provide the amount payable: 635.50
The discount applicable is R44.49
The final amount payable is R591.01
```

C:\cplus\notes\notes4\Example2.exe

```
Provide the amount payable: 380.75
The discount applicable is R17.13
The final amount payable is R363.62
```

4.11 Compound if-statements

Programmers often find that more than one condition must be tested before the result can be true. A **compound if-statement**, based on logical operators can then be used.

Example

If a person is 18 years or older, and has registered as a voter, this person may vote in an election.

```

if age >= 18 AND isRegisteredVoter then
    display "You may vote"
else
    display "You may not vote"
endif

```

The word NOT can also be used, **but with caution**, as the beginner programmer might just achieve the opposite result than what is expected. NOT has a higher precedence than AND and OR, which means it will be evaluated first. Try to avoid using NOT.

```

if age < 18 OR NOT isRegisteredVoter then
    display "You may not vote"
else
    display "You may vote"
endif

```

Example 1

All people entering the Rand Easter show pays a normal entrance fee, except for children under 12 and pensioners (60 and over), who only pays half of the normal entrance fee. Enter the normal entrance fee and the age in full years of the person, and calculate and display the actual entrance fee payable.

Planning:

	Description	Type	Name of variable
Input:	Entrance fee	real	entranceFee
	Age	integer	age
Output:	Actual entrance fee	real	actEntranceFee

I	P	O
entranceFee age	Prompt for input Enter input Determine the actual entrance fee Display actual entrance fee	actEntranceFee

Algorithm:

```
DetermineActualEntranceFee
~   Determine the actual entrance fee
    display "Provide the normal entrance fee "           ~ Display on a new line
    enter entranceFee
    display "Provide the age of the person in whole years " ~ On a new line
    enter age

    ~ Calculate the actual fee
    if age < 12 OR age >= 60 then
        actEntranceFee = entranceFee / 2
    else
        actEntranceFee = entranceFee
    endif

    ~ Display the result on a new line
    display "This person must pay an entrance fee of R", actEntranceFee
end
```

Test the program

Normal entrance fee of R150 and age 9

Output:

```
Provide the normal entrance fee 150
Provide the age of the person in whole years 9
This person must pay an entrance fee of R75
```

Normal entrance fee of R80.70 and age 41

Output:

```
Provide the normal entrance fee 80.70
Provide the age of the person in whole years 41
This person must pay an entrance fee of R80.70
```

Exercises

Do the planning and write an algorithm to solve the following problem:

An athlete may participate in the Comrades Marathon if he/she is a member of a club, and has taken part in the Two Oceans Marathon or the Round the Dam Marathon. Enter the athlete's name, an indication if he/she is a member of a club, an indication if he/she ran in the Two Oceans Marathon and an indication of whether he/she ran in the Round the Dam Marathon. Display the athletes name and an appropriate message displaying whether he/she may participate in the Comrades Marathon.

4.12 EXAMPLE OF THE IF-ELSE STATEMENT WITH LOGICAL OPERATORS IN C++

Example 1 – See algorithm on page 25

```
#include <iostream>
#include <conio.h>
#include <iomanip>

using namespace std;

int main()
{
    //Determine the actual entrance fee
    double entranceFee;
    int age;

    double actEntranceFee = 0.0;
    cout << "Provide the normal entrance fee: ";
    cin >> entranceFee;
    cout << "Provide the age of the person in whole years: ";
    cin >> age;

    //Calculate the actual fee
    if(age < 12 || age >= 60)
        actEntranceFee = entranceFee / 2;
    else
        actEntranceFee = entranceFee;

    //Display the result
    cout << fixed << setprecision(2);
    cout << "This person must pay an entrance fee of R"
        << actEntranceFee << endl;

    _getch();
    return 0;
}
```

C:\cplus\notes\notes4\Example3.exe

Provide the normal entrance fee: 150
Provide the age of the person in whole years: 9
This person must pay an entrance fee of R75.00

C:\cplus\notes\notes4\Example3.exe

Provide the normal entrance fee: 80.70
Provide the age of the person in whole years: 41
This person must pay an entrance fee of R80.70

4.13 Nested if-statements

It often happens that we need to test more than one condition to enable us to take the correct action. This causes us to have to make use of a nested-if: i.e. an if-statement within another if-statement.

Example 1:

Members of a club pay a basic membership fee per month, and then an additional fee depending on the sport they participate in:

Sport	Sport Code	Additional Amount
Tennis	T	160.00
Squash	S	140.00
Rugby	R	150.00

Write an algorithm to enter the basic membership fee, the name of the member, as well as the code of the sport that he/she participates in. Then display a message containing the name of the member, the name of the sport, as well as the membership fee, e.g. Peter takes part in Squash and has a membership fee of R550.50

Planning:

	Description	Type	Name of variable
Input:	Member name	string	memberName
	Basic membership fee	real	basicFee
	Sport code	character	sportCode
Output:	Member name	string	memberName
	Sport name	string	sportName
	Membership fee	real	membershipFee

I	P	O
memberName basicFee sportCode	Prompt for input Enter input Determine the actual entrance fee Display actual entrance fee	actEntranceFee memberName sportName membershipFee

One possible solution, with simple if-statements, would be:

Algorithm:

```
DetermineMembershipFee
~   get the input
    display "Please provide the name of the member"           ~ on new line
    enter memberName
    display "Please provide the basic membership fee"         ~ on a new line
    enter basicFee
    display "Please provide the sport code (T, S or R)"        ~ on new line
    enter sportType
~   determine the membership fee
    if sportType = "T" then                                   ~ test for sport code T
        membershipFee = basicFee + 160.00
        sportName = "Tennis"
    endif
    if sportType = "S" then                                   ~ test for sport code S
        membershipFee = basicFee + 140.00
        sportName = "Squash"
    endif
    if sportType = "R" then                                   ~ test for sport code R
        membershipFee = basicFee + 150.00
        sportName = "Rugby"
    endif
    display memberName, " takes part in ", sportName         ~ on a new line
    display " and has a membership fee of R", membershipFee
end
```

The minimum number of if-statements that will be executed is 3.

A more effective way to write this algorithm is as follows by making use of nested if-statements:

```
DetermineMembershipFee
~   get the input
    display "Please provide the name of the member"           ~ on new line
    enter memberName
    display "Please provide the basic membership fee"         ~ on a new line
    enter basicFee
    display "Please provide the sport code (T, S or R)"        ~ on new line
    enter sportType
~   determine the membership fee
    if sportType = "T" then                                   ~ test for sport code T
        membershipFee = basicFee + 160.00
        sportName = "Tennis"
    else
        if sportType = "S" then                               ~ test for sport code S
            membershipFee = basicFee + 140.00
            sportName = "Squash"
        else
            if sportType = "R" then                             ~ test for sport code R
                membershipFee = basicFee + 150.00
                sportName = "Rugby"
            endif
        endif
    endif
    display memberName, " takes part in ", sportName         ~ on a new line
    display " and has a membership fee of R", membershipFee
end
```

The 2nd algorithm is more effective than the 1st algorithm, as the minimum number of if-statements that will be executed is 1.

Please note that we have assumed that a correct sportCode has been entered. It is however possible that an incorrect sportCode could have been entered, but we did not attend to that possibility in this question.

This type of if-statement is called a **nested if statement**.

Consider the following if-statement:

```
if condition1 then
    statement1
else
    statement2
endif
```

statement1 and/or statement2 may itself be if-then-else statement, depending on the program specification. If this is the case, the general format of a nested if-statement will be as follows:

```
if condition1 then
    if condition2 then
        statement1
    else
        statement2
    endif
else
    if condition3 then
        statement3
    else
        statement4
    endif
endif
```

Note how indenting is done. Any of the above statements may once again be an if-statement.

Example 2

A program is required by a company to read an employee number, department code indicating to which department the employee belongs, the pay rate and the number of hours worked in a week. Then the employee's weekly pay must be calculated, and the employee number and weekly pay must be displayed.

For all departments, if more than 40 hours are worked, payment for overtime hours worked is calculated at time-and-a-half, except for those in department "CS". These employees' overtime hours are calculated as time-and-a-quarter.

Planning:

	Description	Type	Name of variable
Input:	Employee number	string	empNum
	Department code	string	deptCode
	Pay rate	real	payRate
	Number of hours worked	real	hoursWorked
Output:	Employee number	string	empNum
	Weekly pay	real	weeklyPay

I	P	O
empNum deptCode payRate hoursWorked	Prompt for input Enter input Calculate weekly pay Display output	empNum weeklyPay

Algorithm:

```
ComputeEmpPay
~   Get the input
    display "Please provide the employee number " ~ on new line
    enter empNum
    display "Please provide the department code " ~ on new line
    enter deptCode
    display "Please provide the hourly pay rate " ~ on new line
    enter payRate
    display "Please provide the number of hours worked" ~ on new line
    enter hoursWorked

    if    hoursWorked <= 40 then
        weeklyPay = payRate * hoursWorked
    else
        overTimeHours = hoursWorked - 40
        if    deptCode = "CS" then
            overTimePay = overTimeHours * payRate * 1.25
        else
            overTimePay = overTimeHours * payRate * 1.5
        endif
        weeklyPay = (payRate * 40) + overTimePay
    endif
    display empNum, " gets a weekly pay of R", weeklyPay
end
```

Test the program three times using:

1. empNum = 21435, deptCode = "FH", payRate = 30, hoursWorked = 35
2. empNum = 30124, deptCode = "CS", payRate = 27, hoursWorked = 42
3. empNum = 72145, deptCode = "DA", payRate = 40, hoursWorked = 52

Trace table 1:

Instruction	Emp Num	dept Code	Pay Rate	Hours Worked	Over Time Hours	Over Time Pay	Weekly Pay	Outcome of if	output
display									Please provide the employee number
enter	21435								
									Please provide the department code
enter		FH							
display									Please provide the hourly pay rate
enter			30						
display									Please provide the number of hours worked
enter				35					
if								TRUE	
calculate							1050.00		
display									21435 gets a weekly pay of R1050.00

Trace table 2:

Instruction	Emp Num	dept Code	Pay Rate	Hours Worked	Over Time Hours	Over Time Pay	Weekly Pay	Outcome of if	output
display									Please provide the employee number
enter	30124								
									Please provide the department code
enter		CS							
display									Please provide the hourly pay rate

enter			27						
display									Please provide the number of hours worked
enter				42					
if								FALSE	
calculate					2				
if								TRUE	
calculate						67.5			
calculate							1147.50		
display									30124 gets a weekly pay of R1147.50

Trace table 3:

Instruction	Emp Num	dept Code	Pay Rate	Hours Worked	Over Time Hours	Over Time Pay	Weekly Pay	Outcome of if	output
display									Please provide the employee number
enter	72145								
									Please provide the department code
enter		DA							
display									Please provide the hourly pay rate
enter			40						
display									Please provide the number of hours worked
enter				52					
if								FALSE	
calculate					12				
								FALSE	
						720			
calculate							2320.00		
display									72145 gets a weekly pay of R2320.00

Example 3

The gender and age of a person determines how many minutes the athlete will get as a head-start in a race. Every female of 50 years and older gets a 30 minute head-start, whereas all other females gets a head-start of 15 minutes. Males of 60 years and older gets a head-start of 20 minutes, whereas all other males gets no head-start.

Enter the competitor number, the gender (M for male and F for female), as well as the age of the athlete. Then determine the number of head-start minutes, and display the competitor number and the head-start minutes.

Planning:

	Description	Type	Name of variable
Input:	Competitor number	string	compNum
	Gender	character	gender
	Age	integer	age
Output:	Competitor number	string	compNum
	Head-start minutes	integer	headStartMin

I	P	O
compNum gender age	Prompt for input Enter input Determine head-start minutes Display output	compNum headStartMin

Algorithm:

```
DetermineHeadStartMinutes
~   Get the input
    display "Please provide the competitor number "           ~ on new line
    enter compNum
    display "Please provide the gender of the competitor (M for male, F for female) "
                                                                ~ on a new line
    enter gender
    display "Please provide the age of the competitor "         ~ on new line
    enter age
    if gender = "F" then
        if age >= 50 then
            headStartMin = 30
        else
            headStartMin = 15
    endif
```

```

else
    if gender = "M" then
        if age >= 60 then
            headStartMin = 20
        else
            headStartMin = 0
        endif
    else
        headStartMin = 0
        display "Invalid gender entered"
    endif
endif

display compNum, " gets a ", headStartMin, " minute head-start"
end

```

Test the program four times using:

1. compNum = 4545, gender = "M", age = 65

Output:

Provide the competitor number 4545
 Provide the gender of the competitor (M for male, F for female) M
 Provide the age of the competitor 65
 4545 gets a 20 minute head-start

2. compNum = 2154, gender = "F", age = 20

Output:

Provide the competitor number 2154
 Provide the gender of the competitor (M for male, F for female) F
 Provide the age of the competitor 20
 2154 gets a 15 minute head-start

3. compNum = 7465, gender = "F", age = 57

Output:

Provide the competitor number 7465
 Provide the gender of the competitor (M for male, F for female) F
 Provide the age of the competitor 57
 7465 gets a 30 minute head-start

4. compNum = 8978, gender = "M", age = 42

Output:

Provide the competitor number 8978
 Provide the gender of the competitor (M for male, F for female) M
 Provide the age of the competitor 42
 8978 gets a 0 minute head-start

4.14 C++ Nested IF (Multiple If-Statements)

Some problems require the implementation of more than two alternatives. Now let's implement example 1 to 3 on pages 29 - 36 in C++:

Example 1

```
#include <iostream>
#include <string>
#include <conio.h>
    using namespace std;
int main()
{
    //get the input
    string memberName;
    double basicFee;
    char sportType;
    double membershipFee = 0.0;
    string sportName = "";

    cout << "Please provide the name of the member: ";
    cin >> memberName;
    cout << "Please provide the basic membership fee: ";
    cin >> basicFee;
    cout << "Please provide the sport code (T, S or R): ";
    cin >> sportType;
    //determine the membership fee
    if (sportType == 'T')    //test for sport code T
    {
        membershipFee = basicFee + 160.00;
        sportName = "Tennis";
    }
    else if (sportType == 'S')    // test for sport code S
    {
        membershipFee = basicFee + 140.00;
        sportName = "Squash";
    }
    else if (sportType == 'R')    // test for sport code R
    {
        membershipFee = basicFee + 150.00;
        sportName = "Rugby";
    }
    cout << memberName << " takes part in " << sportName
        << " and has a membership fee of R"
        << membershipFee << endl;

    _getch();
    return 0;
}
```

Example 2:

```
#include <iostream>
#include <string>
#include <conio.h>

using namespace std;

int main()
{
    //Get the input
    int empNum, hoursWorked;
    double payRate;
    int overTimeHours = 0;
    double weeklyPay = 0.0;
    double overTimePay = 0.0;
    string deptCode;

    cout << "Please provide the employee number: ";
    cin >> empNum;
    cout << "Please provide the department code: ";
    cin >> deptCode;
    cout << "Please provide the hourly pay rate: ";
    cin >> payRate;
    cout << "Please provide the number of hours worked: ";
    cin >> hoursWorked;

    if (hoursWorked <= 40)
        weeklyPay = payRate * hoursWorked;
    else
    {
        overTimeHours = hoursWorked - 40;
        if (deptCode == "CS")
            overTimePay = overTimeHours *
                payRate * 1.25;
        else
            overTimePay = overTimeHours *
                payRate * 1.5;

        weeklyPay = (payRate * 40) + overTimePay;
    }
    cout << empNum << " gets a weekly pay of R"
        << weeklyPay << endl;

    _getch();
    return 0;
}
```

```
Please provide the employee number: 21435
Please provide the department code: FH
Please provide the hourly pay rate: 30
Please provide the number of hours worked: 35
21435 gets a weekly pay of R1050
```

```
Please provide the employee number: 30124
Please provide the department code: CS
Please provide the hourly pay rate: 27
Please provide the number of hours worked: 42
30124 gets a weekly pay of R1147.5
```

```
Please provide the employee number: 72145
Please provide the department code: DA
Please provide the hourly pay rate: 40
Please provide the number of hours worked:
72145 gets a weekly pay of R2320
```

Example 3:

```
#include <iostream>
#include <string>
#include <conio.h>

using namespace std;

int main()
{
    //Get the input
    int compNum, age;
    char gender;

    int headStartMin = 0;

    cout << "Please provide the competitor number: ";
    cin >> compNum;
    cout << "Please provide the gender of the competitor (M for male, F for female): ";
    cin >> gender;
    cout << "Please provide the age of the competitor: ";
    cin >> age;

    if (gender == 'F')
        if (age >= 50)
            headStartMin = 30;
        else
            headStartMin = 15;
    else if (gender == 'M')
        if (age >= 60)
            headStartMin = 20;
        else
            headStartMin = 0;
    else
    {
        headStartMin = 0;
        cout << "\nInvalid gender entered\n";
    }
    cout << compNum << " gets a " << headStartMin
        << " minute head-start"<<endl;
    _getch();
    return 0;
}
```

```

}
C:\cplus\notes\notes4\Example6.exe
Please provide the competitor number: 2154
Please provide the gender of the competitor (M for male, F for female): F
Please provide the age of the competitor: 20
2154 gets a 15 minute head-start

C:\cplus\notes\notes4\Example6.exe
Please provide the competitor number: 7465
Please provide the gender of the competitor (M for male, F for female): F
Please provide the age of the competitor: 57
7465 gets a 30 minute head-start

C:\cplus\notes\notes4\Example6.exe
Please provide the competitor number: 8978
Please provide the gender of the competitor (M for male, F for female): M
Please provide the age of the competitor: 42
8978 gets a 0 minute head-start

```

Algorithm Exercises:

1. In each of the following cases, do the planning and write an algorithm to solve the following problems:
 - 1.1 An architect's fee is calculated as a percentage of the cost of the building. The fee is made up as follows:

On the first R350 000 of the cost of the building:	8%
On the remainder if the remainder is less than or equal to R500 000:	3%
On the remainder if the remainder is more than R500 000:	2½%

Enter the cost of the building, and calculate and display the architect's fee.

- 1.2 Design an algorithm that will prompt for and enter the price of an article and a discount code. Calculate a discount according to the discount code, and display the original price on the screen as well as the discount amount and the discounted price. The discount code and accompanying discount rate is as follows:

Discount Code	Discount Rate
H	50%
F	40%
T	33%
Q	25%
Z	0%

If the discount code is Z, the words "No discount" are to be printed on the screen. If the discount code is neither of the above, the words "Invalid discount code" must be displayed.

- 1.3 Thabo is a taxi driver and has 3 approved routes, each with the following cost.

Route A: 11.50

Route B: 9.00

Route C: 7.50

Pensioners get 10% discount and children under the age of 10 years pay only R2 per trip, regardless the route.

For one passenger, enter the route code and whether this person is a pensioner or under the age of 10. Ensure that all values entered are valid. Calculate the amount due and display the output in the format provided by the 3 examples.

Amount due for route A (Pensioner): R 10.35

Amount due for route B (Normal rate): R9.00

Amount due for route C (Child): R2.00

4.15 Data validation

In programming we have a saying: Garbage in, garbage out. On other words, if our input is not valid, the output will not be correct. We as programmers must therefore code defensively. This means that we have to try to detect all errors that the user of the program may make. Most computer errors are as a result of incorrect user input. If a program picks up the errors before it is processed, the output (information) will be accurate and reliable.

Available to the programmer are a number of **data validation** tests that we can use to code defensively. At this early stage, we will only test whether a numeric field contains a numeric value before it is processed, i.e. if the numeric variable contains a non-numeric value, it will not be processed.

Example of a numeric test in an algorithm:

```
display "Please provide the length and width of the swimming pool"
enter length
enter width
if length is numeric and width is numeric then
    area = length * width
    display "The area of the swimming pool is ", area
else
    display "Invalid input"
endif
```

4.16 The select case structure

The select case structure gives the programmer an alternative way to test the same variable for different values.

Consider the following nested if statement:

```
if colourCode = "B" then
    display "Blue paint"
else
    if colourCode = "Y" then
        display "Yellow paint"
    else
        if colourCode = "W" then
            display "White paint"
        else
            display "Invalid colour code"
        endif
    endif
endif
```

The same condition can be tested using a select case structure that is easier to read:

```
select case colourCode
    case "B"    display "Blue paint"
    case "Y"    display "Yellow paint"
    case "W"    display "White paint"
    case else   display "Invalid colour code"
endselect
```

One can also have more than one statement executed in each case:

```
select case colourCode
    case "B"    display "Blue paint"
                pricPerLitre = 55.50
    case "Y"    display "Yellow paint"
                pricPerLitre = 60.80
    case "W"    display "White paint"
                pricPerLitre = 47.30
    case else   display "Invalid colour code"
endselect
```

4.17 Examples containing the select case structure

Example 1:

Consider the problem in Example 1 on page 29 under Nested-if statements:

Members of a club pay a basic membership fee per month, and then an additional fee depending on the sport they participate in:

Sport	Sport Code	Additional Amount
Tennis	T	160.00
Squash	S	140.00
Rugby	R	150.00

Write an algorithm to enter the basic membership fee, the name of the member, as well as the code of the sport that he/she participates in. Then display a message containing the name of the member, the name of the sport, as well as the membership fee, e.g. Peter takes part in Squash and has a membership fee of R550.50

Written with a select case structure, the algorithm will be:

```

DetermineMembershipFee
~   get the input
    display "Please provide the name of the member" ~ on new line
    enter memberName
    display "Please provide the basic membership fee" ~ on new line
    enter basicFee
    display "Please provide the sport code (T, S or R)" ~ on new line
    enter sportType

~   determine the membership fee
    select case sportType
        case "T" ~ test for sport code T
            membershipFee = basicFee + 160.00
            sportName = "Tennis"
        case "S" ~ test for sport code S
            membershipFee = basicFee + 140.00
            sportName = "Squash"
        case "R" ~ test for sport code R
            membershipFee = basicFee + 150.00
            sportName = "Rugby"
    endselect

    display memberName " takes part in ", sportName ~ on new line
    display " and has a membership fee of R", membershipFee

end

```

Example 2:

Write an algorithm that will read two numbers and an integer code. The value of the integer code should be 1, 2, 3 or 4. If the value of the code is 1, calculate the sum of the two numbers. If the code is 2, compute the difference of the two numbers (first minus second). If the code is 3, compute the product of the two numbers. If the code is 4, and the second number is not zero, compute the quotient (first divided by second). If the code is not equal to 1, 2, 3 or 4, display an error message.

DoSelectiveCalculations

~ This program enters two numbers and an integer code. Then, based on the code,
~ different calculations are done

```
display "Enter two numbers "
enter num1
enter num2

display "Enter the calculation code:"
display "    1 to get the sum of the two numbers"
display "    2 to get the difference between the two numbers"
display "    3 to get the product of the two numbers"
display "    4 to get the quotient of the two numbers"
enter calcCode
select case calcCode
    case 1      result = num1 + num2

    case 2      result = num1 - num2

    case 3      result = num1 * num2

    case 4      if num2 <> 0 then
                  result = num1 / num2
                else
                  display "Second number is 0 – result set to 0"
                  result = 0
                endif
    case else   display "Invalid calculation code – result set to 0"
                result = 0
endselect
display "The result is = ", result
end
```


Exercises:

Rewrite the algorithms written in exercises 1.2 and 1.3 of the nested-if statements, using a select case structure:

- 1.2 Design an algorithm that will prompt for and enter the price of an article and a discount code. Calculate a discount according to the discount code, and display the original price on the screen as well as the discount amount and the discounted price. The discount code and accompanying discount rate is as follows:

Discount Code	Discount Rate
H	50%
F	40%
T	33%
Q	25%
Z	0%

If the discount code is Z, the words "No discount" are to be printed on the screen. If the discount code is neither of the above, the words "Invalid discount code" must be displayed.

- 1.3 Thabo is a taxi driver and has 3 approved routes, each with the following cost.

Route A: 11.50

Route B: 9.00

Route C: 7.50

Pensioners get 10% discount and children under the age of 10 years pay only R2 per trip, regardless the route.

For one passenger, enter the route code and whether this person is a pensioner or under the age of 10. Ensure that all values entered are valid. Calculate the amount due and display the output in the format provided by the 3 examples.

Amount due for route A (Pensioner): R 10.35

Amount due for route B (Normal rate): R9.00

Amount due for route C (Child): R2.00

4.18 WRITING THE SELECT CASE IN C++ (SWITCH STRUCTURES)

There are two types of selection structures in C++:

- The first selection structure, which is implemented with if and if. . .else statements, usually requires the evaluation of a (logical) expression.
- The second selection structure, which does not require the evaluation of a logical expression, is called the switch structure. Statements are evaluated based on the value of an integral expression.

A general syntax of the switch statement is:

```
switch (expression)
{
    case value1:
        statements1;
        break;
    case value2:
        statements2;
        break;
    .
    .
    .
    case valueN:
        statementsN;
        break;
    default:
        statements;
}
```

The switch, case, break, and default are reserved words.

The switch statement executes according to the following rules:

- When the value of the expression is matched against a case value, then statements will execute until either a break statement is found or the end of the switch structure is reached.
- If the value of the expression does not match any of the case values, the statements following the default will execute.
- If the switch structure has no default and if the value of the expression does not match any of the case values, the entire switch statement is skipped.
- A break statement causes an immediate exit from the switch structure.

Now let's rewrite example 1 and 2 on page 45 and 46 with the Switch-statement:

Example 1:

```
#include <iostream>
#include <string>
#include <conio.h>
    using namespace std;

int main()
{
    string memberName;
    double basicFee;
    char sportType;
    double membershipFee = 0.0;
    string sportName = "";
    cout << "Please provide the name of the member\n";
```

```

cin >> memberName;
cout << "Please provide the basic membership fee\n";
cin >> basicFee;
cout << "Please provide the sport code (T, S or R)\n";
cin >> sportType;

//determine the membership fee
switch (sportType)
{
    case 'T':    //test for sport code T
        membershipFee = basicFee + 160.0;
        sportName = "Tennis";
        break;
    case 'S':    //test for sport code S
        membershipFee = basicFee + 140.0;
        sportName = "Squash";
        break;
    case 'R':    // test for sport code R
        membershipFee = basicFee + 150.0;
        sportName = "Rugby";
}

cout << memberName << " takes part in " << sportName
    << " and has a membership fee of R"
    << membershipFee << endl;

_getch();
return 0;
}

```

```

Please provide the name of the member
Peter
Please provide the basic membership fee
550.50
Please provide the sport code (T, S or R)
S
Peter takes part in Squash and has a membership fee of R690.5
-

```

Example 2:

```
#include <iostream>
#include <string>
#include <conio.h>

using namespace std;
int main()
{
    /* This program enters two numbers and an integer code.
    Then, based on the code,
    different calculations are done */
    int num1, num2, calcCode;
    int result = 0;

    cout << "Enter two numbers\n";
    cin >> num1 >> num2;
    cout << "Enter the calculation code:\n"
        << "    1 to get the sum of the two numbers\n"
        << "    2 to get the difference between the two numbers\n"
        << "    3 to get the product of the two numbers\n"
        << "    4 to get the quotient of the two numbers\n";
    cin >> calcCode;
    switch (calcCode)
    {
        case 1:
            result = num1 + num2;
            break;
        case 2:
            result = num1 - num2;
            break;
        case 3:
            result = num1 * num2;
            break;
        case 4:
            if (num2 != 0)
                result = num1 / num2;
            else
            {
                cout << "\nSecond number is 0 - result set to 0\n";
                result = 0;
            }
            break;
        default:
            cout << "\nInvalid calculation code - result set to 0\n";
            result = 0;
    }
    cout << "The result is = " << result << endl;
    _getch();
    return 0;
}
```

```
Enter two numbers
33
9
Enter the calculation code:
  1 to get the sum of the two numbers
  2 to get the difference between the two numbers
  3 to get the product of the two numbers
  4 to get the quotient of the two numbers
2
The result is = 24
```

```
Enter two numbers
15
7
Enter the calculation code:
  1 to get the sum of the two numbers
  2 to get the difference between the two numbers
  3 to get the product of the two numbers
  4 to get the quotient of the two numbers
3
The result is = 105
```

```
Enter two numbers
37
0
Enter the calculation code:
  1 to get the sum of the two numbers
  2 to get the difference between the two numbers
  3 to get the product of the two numbers
  4 to get the quotient of the two numbers
4
Second number is 0 - result set to 0
The result is = 0
```

```
Enter two numbers
15
5
Enter the calculation code:
  1 to get the sum of the two numbers
  2 to get the difference between the two numbers
  3 to get the product of the two numbers
  4 to get the quotient of the two numbers
5
Invalid calculation code - result set to 0
The result is = 0
```

4.19 SUMMARY

In addition to these notes, study the following pages on **Chapter 4** inside the prescribed textbook.

- Page 137 – 142
- Page 148 – 154

Do the following exercises:

- **EXERCISES 4.1** Page 142- 143
 - Do all the exercises (1 – 4)
- **EXERCISES 4.2** Page 154 – 158
 - Do all the exercises (1 - 17)
- **EXERCISES 4.3** page 162 – 167
 - Do all the exercises (1 - 12)
- **EXERCISES 4.4** page 171 – 172
 - Do all the exercises (1 - 7)

Chapter 5

The Repetition (Iteration) Control Structure

5.1 Introduction

Any program can be written by using a combination of three basic control structures: sequence, selection and iteration. In this chapter we discuss the third of the three basic control structures, the **repetition (iteration) control structure**. This is also often called a **looping structure**.

When developing an algorithm, it is very often required that a certain set of statements is executed a number of times, for instance calculating the final mark for a subject for all students in a class. It is not necessary to repeat the statements to be calculated repeatedly, as the repetition control structure is available for the programmer to use.

5.2.1 Repetition using a fixed-count loop

In some instances the programmer knows exactly how many times the statements need to be repeated, and sometimes not. If the programmer knows exactly how many times the statements must be executed, the **fixed (automatic) count loop** is used. We use the **for-next loop** to accomplish this. The syntax is as follows:

for variable1 = begin-value to end-value [step variable2]	
statement1	} ~ Body of for-next statement
statement2	
:	
statementn	
next variable1	

- variable1 is the loop control-variable (index), and is initialized to begin-value when the loop starts executing.
- The statements in the body of the loop will be executed, and then the loop control-variable, variable1, will be incremented by variable2.
- The process is repeated until variable1 has moved passed the end-value for the first time.
- Should the [step variable2] be omitted, the step will automatically be +1.
- The body of the for-loop can consist of 1 or many statements.
- The step can be a negative value. If so, begin-value must be larger than end-value.
- The variables do not have to be integers, it can also be real.

5.2.2 Examples of the for-loop

Example 1

Display all consecutive integers from 5 to 12.

```
for i = 5 to 12
    display i, " "      ~ On the same line
next i
```

The output produced will be:

5 6 7 8 9 10 11 12

Example 2

Display all consecutive integers from 5 to 12, in descending sequence.

```
for i = 12 to 5 step -1
    display i, " "      ~ On the same line
next i
```

The output produced will be:

12 11 10 9 8 7 6 5

Exercise

Predict the output for the following for-loops:

1. for x = 5 to 8.5 step 0.5
 display x, " " ~ On the same line
next x
2. for x = -4 to 7 step 3
 display x, " " ~ On the same line
next x
3. for x = 15 to -1 step -3
 display x, " " ~ On the same line
next x
4. for x = 2 to 22 step 5
 display x, " " ~ On the same line
next x

Example 3

Determine and display the sum of the first 4 positive even numbers:

sum = 0	~ Initialize sum before entering the loop
even = 2	~ The first positive even number
for k = 1 to 4	~ The loop will execute 4 times
sum = sum + even	~ Add the current even number to sum
even = even + 2	~ Get the next even number
next k	~ Get the next value of loop control-variable k
display "The sum of the first 4 even numbers is ", sum	~ display the answer once, after the loop has executed

Trace table to illustrate the execution of this for-loop:

Instruction	sum	even	k	Condition	Output
assignment	0				
assignment		2			
for			1		
for				Is 1 <= 4? Y	
calculation	2				
calculation		4			
next			2		
for				Is 2 <= 4? Y	
calculation	6				
calculation		6			
next			3		
for				Is 3 <= 4? Y	
calculation	12				
calculation		8			
next			4		
for				Is 4 <= 4? Y	
calculation	20				
calculation		10			
next			5		
for				Is 5 <= 4? N	
display					The sum of the first 4 even numbers is 20

Please note that the sum is only displayed once, after the loop has been executed. This is because we only know the final value of sum after all four the even numbers have been added (inside the loop).

This process of adding inside a loop is called **accumulation**. The variable sum is called the **accumulator**. We often use accumulation in loops to determine a total or an average. If only 1 is added to the accumulator inside the loop, the accumulator is called a **counter**.

Example 4

Display the first 5 multiples of 4 (starting with 4), followed by their sum.

MultiplesOfFour	
sum = 0	~ Initialize sum before entering the loop
multiple = 4	~ The first multiple of 4
display "The first 5 multiples of 4:"	
for k = 1 to 5	~ The loop will execute 5 times
display multiple	~ On a new line
sum = sum + multiple	~ Add the current multiple to sum
multiple = multiple + 4	~ Get the next multiple
next k	~ Get the next value of control variable k
display "The sum of the first 5 multiples of 4 is ", sum	~ display the sum once, at the end of the loop
end	

In this problem we need to display each multiple inside the loop.

The sum is accumulated inside the loop, but displayed after the loop has terminated.

Trace table:

Instruction	sum	multiple	k	Condition	Output
assignment	0				
assignment		4			
display					The first 5 multiples of 4:
for			1		
for				Is 1 <= 5? Y	
display					4
calculation	4				
calculation		8			
next			2		
for				Is 2 <= 5? Y	
display					8
calculation	12				
calculation		12			
next			3		
for				Is 3 <= 5? Y	
display					12
calculation	24				
calculation		16			
next			4		
for				Is 4 <= 5? Y	
display					16
calculation	40				
calculation		20			
next			5		
for				Is 5 <= 5? Y	
display					20
calculation	60				
calculation		24			
next			6		
for				Is 6 <= 5? N	
display					The sum of the first 5 multiples of 4 is 60

Example 5

Enter 10 integers that falls within the range of 25 and 80. Display the average of these numbers.

```
DetermineAve10Numbers
    sum = 0
    for number = 1 to 10
        display "Enter a value between 25 and 80"
        enter num
        if number >= 25 AND number <= 80 then
            sum = sum + num
        else
            display "The number must be within the range 25 to 80, please re-enter"
            number = number - 1
        endif
    next number
    average = sum / 10
    display "The average of the 10 numbers is ", average
end
```

Notes:

- Because 10 numbers must be entered, this must be done inside the loop.
- Because each number must fall within a specific range, the number must be validated to make sure that it is valid. If it does not fall within the range, an error message must be displayed and the user must be asked to enter a new number. As this is a re-entering of a number, the loop control-variable must be decremented by 1, so that when the loop control-variable is incremented, it is back to the same value as when the previous incorrect value was entered.
- The average can only be determined at the end, when the sum of all 10 numbers is known.

Example 6

There are 17 departments in Ms Mabena's supermarket. She would like to know which department had the highest sales for the month. For each of the 17 departments, enter the department's name and the sales for this department this month. Then calculate and display the department name with the highest sales, along with the sales the department obtained.

	Description	Type	Variable Name
Input:	department name	string	deptName
	department sales	real	deptSales
Intermediate:	department number to control the loop	integer	dept
Output:	name of best department	string	bestName
	best sales	real	bestSales

I	P	O
For each department: deptName deptSales	Initialize values For each department: Prompt for and enter input Determine the best department Display output values	bestName bestSales

```

DetermineHighestSales
~ This algorithm will find the department with the highest sales
~ Enter department name and sales of the first department
display "Enter the name of the first department"
enter deptName
display "Enter the sales for department ", deptName
enter deptSales
bestName = deptName      ~ Initialize the best name to the first dept's name
bestSales = deptSales    ~ Initialize the best sales to the first dept's sales
for dept = 2 to 17       ~ Repeat for the rest of the departments
    display "Enter the name for department ", dept
    enter deptName
    display "Enter the sales for department ", deptName
    enter deptSales
    ~ if the sales is higher than the current best sales, then replace with this dept
    if deptSales > bestSales then
        bestSales = deptSales
        bestName = deptName
    endif
next dept
~ The final result can only be displayed when all the departments have been considered
display "The department with the best sales is ", bestName, " with sales of ", bestSales
end

```

Exercises:

1. How many times will the following loops be executed?

1.1 for count = 4 to 15

1.2 for i = 1 to 15 step 2

1.3 for x = 90 to 60 step -2

1.4 for num = 1 to 30 step 4

2. Solve the following problems by defining the variables, declaring the input and output and also write the algorithm. Use the **for** structure in every case.

- 2.1 Ten students had worked in a clothing store during the holidays. At the end of the holiday, they receive a wage according to the number of items that they sold. The rate of pay is R6.50 per item. Enter the student name and the number of items that the student sold during the holidays and display the student name and the amount earned by every student. Also calculate and display the student name of the student who sold the most items.
 - 2.2 Enter an integer indicating the number of integers that need to be entered. Enter all the rest of the integers, and calculate and display the sum and average of all the integers entered, except the first one.
 - 2.3 Enter a beginning and ending value of a set of numbers, and calculate and display, for each of the numbers, the square of the number, and a message indication whether it is divisible by 3 or not.
3. Give the exact output of what will be displayed on the screen after each of the following algorithms have been processed.

3.1 ShowResult1

```

a = 2
b = 4
for k = 4 to 7
    a = a + 3
    b = b + a
    display "a = ", a, "b = ", b           ~ on new line
next k
display "The product of a and b is ", a*b   ~ on new line
end

```

3.2 ShowResult2

```

x = 2
y = x * 4 - 2
z = x * y
for w = 0 to x
    y = y mod 3 + w + 3
    z = z + y
    display w, x, y, z                     ~ on new line
next w
display "*****"
end

```

```

3.3 ShowResult3
    k = 2
    m = 5
    n = m - k
    for j = 10 to 6 step -2
        if m > n then
            n = n * j
            display n
        else
            k = m * n
            display k
        endif
    next j
    display j
end

```

~ on new line

~ on new line

5.2.3 Nested for-statements

It is often needed to have a loop within another loop. This is called a nested loop.

Example 1

for i = 1 to 3	~ Outer loop
for j = 1 to 2	~ Inner loop
display i, " ", j	~ Statement(s) to be displayed on a new line
next j	~ End of inner loop
next i	~ End of outer loop

The output of the program is as follows:

```

1 1
1 2
2 1
2 2
3 1
3 2

```

Trace table:

Instruction	i	j	Test	Output
outer for	1			
outer for			Is 1 <= 3? Y	
inner for		1		
inner for			Is 1 <= 2? Y	
display				1 1
next j		2		
inner for			Is 2 <= 2? Y	
display				1 2
next j		3		
inner for			Is 3 <= 2? N	
next i	2			
outer for			Is 2 <= 3? Y	
inner for		1		

inner for			Is 1 <= 2? Y	
display				2 1
next j		2		
inner for			Is 2 <= 2? Y	
display				2 2
next j		3		
inner for			Is 3 <= 2? N	
next i	3			
outer for			Is 3 <= 3? Y	
inner for		1		
Inner for			Is 1 <= 2? Y	
display				3 1
next j		2		
inner for			Is 2 <= 2? Y	
display				3 2
next j		3		
inner for			Is 3 <= 2? N	
next i	4			
outer for			Is 4 <= 3? N	

One can clearly see from the trace table that, for every value of the outer loop, the program goes through all the values of the inner loop, before the control-variable for the outer loop is incremented. Once incremented, the control-variable for the inner loop will start at the initial value again.

Example 2

A company has 3 salespersons in their service, and wants a program developed that will determine and display, for each of the salespersons, their total sales for the 5 working days in a week. Enter the salesperson's name, and then one by one the sales for each day of the week. Display the salesperson's name and his/her total sales for the week. Also determine the name of the salesperson with the highest sales, as he will be salesperson of the week.

Planning

	Description	Type	Variable Name
Input:	salesperson name	string	salesName
	salesperson's daily sales	real	daySales
Intermediate:	salesperson number to control the outer loop	integer	sp
	day number to control the inner loop	integer	day
Output:	salesperson name	string	salesName
	salesperson weekly sales	real	weeklySales
	highest sales	real	highestSales
	highest name	string	highestName

I	P	O
For each salesperson: salesName For each day: daySales	Initialize values For each salesperson: Prompt for and enter salesName For each day: Prompt & enter daySales Accumulate weeklySales Display salesName & weeklySales Determine name of salesperson with highest sales Display name of salesperson with highest sales	For each salesperson: salesName weeklySales highestName

Algorithm:

```

DetermineWeeklySales
    ~ Determine and display, for 3 salespersons, their weekly sales as well as
    ~ the salesperson with the highest sales

    highestSales = 0
    highestName = " "

    for sp = 1 to 3          ~ Loop for the 3 salespersons
        display "Enter the salesperson's name "
        enter salesName
        weeklySales = 0
        display "For each day of the week, enter the sales for the day:"
        for day = 1 to 5      ~ Loop for the 5 days of the week
            display "    for day ", day
            enter daySales
            weeklySales = weeklySales + daySales
        next day

        display "The weekly sales for ", salesName, " is ", weeklySales

        if weeklySales > highestSales then          ~ Determine the highest sales
            highestSales = weeklySales
            highestName = salesName
        endif
    next sp

    display "The salesperson of the week is ", highestName
end

```

Test the program with the following test data:

	Days	1	2	3	4	5
Salesperson	1	4750.50	7230.70	2380.40	5812.55	5278.20
	2	3333.00	2168.40	5436.70	4100.50	7255.60
	3	2578.35	7891.30	6777.70	4555.55	3535.35

Exercises:

1. Do the planning and the coding to draw the following figures, using nested for statements:

1.1 *
 **

1.2 #*#*#
 #*#*
 #*#
 #*
 #

2. How many times will the display statement in the following nested for-statements be executed?

2.1 for k = 1 to 5
 for j = 2 to 10
 display k, " ", j
 next j
 next k

2.2 for k = -4 to 7 step 2
 for j = 4 to 15 step 3
 display k, " ", j
 next j
 next k

3. Do the planning and write an algorithm to solve the following problem:

TUT Gymnastic Club wants you to write a program that will calculate the winner of the Women's and Men's annual competition. There are 20 gymnasts taking part, 10 women and 10 men. The women take part in 4 events, as follows:

Event Number	Event Name
1	Uneven Bars
2	Balance Beam
3	Floor Exercise
4	Vault

The men take part in 6 events, as follows:

Event Number	Event Name
1	Floor Exercise
2	Pommel Horse
3	Still Rings
4	Vault
5	Parallel Bars
6	The High Bar

The score for each event is a score out of 10. All the events' scores must be added together to give the total score. The name of the winner of the women's and men's competition must be determined and displayed, along with each ones score.

You must write the algorithm to solve the problem by:

- For all the women:
 - Prompt and enter the name of the contestant. Then, for each event, display the event name (Use a **select case** statement to determine the event name) and ask for the contestant score for this event, for example:

```
Enter the name of contestant 1
Mary Nkosi
Enter the mark for Uneven Bars for Mary Nkosi: 6
Enter the mark for Balance Beam for Mary Nkosi: 7
Enter the mark for Floor Exercise for Mary Nkosi: 4
Enter the mark for Vault for Mary Nkosi: 5
```

- Determine the total score of the contestant, and display the score.
 - Determine the winner, and display the winner's name and score.
- Do the same as above for all the men.

5.2.4 THE FOR STATEMENT IN C++

The for-loop simplify the writing of counter-controlled loops. The general format of the for-statement is:

```
for (initial statement; loop condition; update statement)
    statement;
```

In C++, for is a reserved word. The initial statement, loop condition, and update statement enclosed within the parentheses control the body of the for statement.

The for loop executes as follows:

1. The initial statement executes once during the first execution of the loop. The initial statement usually initializes the loop control variable or index variable.
2. The loop condition is evaluated. If the loop condition evaluates to true:
 - The for loop statement/body will execute.
 - Then the update statement (the third expression in the parentheses) will execute.
3. Repeat Step 2 until the loop condition evaluates to false.

The following for loop display all consecutive integers from 5 to 12 as shown in Example 1 on page 3 of part A:

```
for (i = 5; i <= 12 ; i = i + 1)
    cout << i << " "; //this is the for loop statement
cout << endl;
```

NOTE: Because there is only one statement in this for-loop, the curly braces {} may be omitted.

The initial statement, `i = 5;`, initializes the int variable `i` to 5.

- Then, the loop condition, `i <= 12`, is evaluated. Because `5 <= 12` is true, then outputs 5 will be displayed with a space.
- The update statement, `i = i + 1;`, then executes, which sets the value of `i` to 6.

Once again, the loop condition is evaluated, which is still true, and so on. When `i` becomes 13, the loop condition evaluates to false, the for loop terminates, and then, the `cout << endl;` statement following the for loop will execute.

NOTE: Beside integer data types, C++ allows you to use fractional values for the loop control-variable (index variable) of double type (or float data type). And, if the loop condition is initially false, the loop body does not execute.

Similarly to the if statement, a for-loop can have compound statement.

```
for (initial statement; loop condition; update statement)
{
    statement 1;
    statement 2;
    .
    .
    Statement n;
}
```

Note: The initial statement, loop condition, and update statement enclosed within the parentheses of the for-loop are optional, however, the two semicolons must always be included in these parentheses. For example, **for (; i <= 12;)** is valid

```
int i = 5;
for (; i <= 12; )
{
    cout << i << " ";
    i = i + 1;
}
```

Although the initializing and update statement can be omitted from a for-statement, omitting the loop condition expression results in an infinite loop.

5.2.5 Increment and Decrement Operators

Suppose count is an int variable. The statement:

```
count = count + 1;
```

increments the value of count by 1.

To execute this assignment statement, the computer first evaluates the expression on the right, which is `count + 1`. It then assigns this value to the variable on the left, which is `count`. C++ provides the increment operator, `++`, which increases the value of a variable by 1, and the decrement operator, `--`, which decreases the value of a variable by 1. Then, using the increment operator, the previous statement will be written like this:

```
count++; or ++count
```

Both the `++` and `--` are unary operators and operators in two forms, pre and post.

Pre-increment: `++count;` add 1 first

Post-increment: `count++;` add 1 after

The following for loop display all consecutive integers from 5 to 12, in descending sequence as shown in Example 2 on page 3 of part A:

```
for (i = 12; i >= 5 ; i--)  
    cout << i << " "; //this is the for loop statement  
cout << endl;
```

Now let's try Example 3 on page 4 which determine and display the sum of the first 4 positive even numbers:

```
int main()  
{  
    int sum = 0;                // Initialize sum before entering the loop  
    int even = 2;              // The first positive even number  
    for (int k = 1; k <= 4; k++) // The loop will execute 4 times  
    {  
        sum = sum + even;      // Add the current even number to sum  
        even = even + 2;       // Get the next even number  
    }  
  
    cout << "The sum of the first 4 even numbers is " << sum << endl;  
    return 0;  
}
```

5.2.6 ASSIGNMENT VARIATIONS

Assignment expressions where the same variable is used on both sides of the assignment operator can be written by using the following **shortcut assignment operators**:

+=	add and assign
-=	subtract and assign
*=	multiple and assign
/=	divide and assign
%=	mod and assign

The expression, `sum = sum + even;` can be written as follows:

```
sum += even;    and  
even = even + 2; //is the same as even += 2;
```

Now let's try Example 4 on page 5 which display the first 5 multiples of 4 (starting with 4), followed by their sum.

```
int main()
{
    int k;
    int sum = 0;                //Initialize sum before entering the loop
    int multiple = 4;           // The first multiple of 4
    cout << "The first 5 multiples of 4:\n";
    for (k = 1; k <= 5; k++)    // The loop will execute 5 times
    {
        cout << multiple << endl;
        sum += multiple;        // Add the current multiple to sum
        multiple += 4;           //Get the next multiple
    }
    cout << "The sum of the first 5 multiples of 4 is " << sum << endl;
    return 0;
}
```

5.2.7 Interactive for-loops in C++

Using the cin object inside a for-loop creates an interactive for loop.

Now let's try Example 5 on page 6 which enters 10 integers that falls within the range of 25 and 80. Display the average of these numbers:

```
int main()
{
    int number, num;
    int sum = 0;
    for (number = 1; number <= 10; number++)
    {
        cout << "Enter a value between 25 and 80\n";
        cin >> num;
        if (number >= 25 && number <= 80 )
            sum += num;
        else
        {
            cout << "The number must be within the range 25 to 80, please re-enter\n";
            number--;
        }
    }
    average = sum / 10;
    cout << "The average of the 10 numbers is " << average << endl;
    return 0;
}
```

The following program is an implementation of example 6 on page 6 and 7 in C++:

```
int main()
{
    //Find the department with the highest sales
    //Enter department name and sales of the first department
    string deptName;
    double deptSales;
    cout << "Enter the name of the first department\n";
    cin >> deptName;
    cout << "Enter the sales for department " << deptName << endl;
    cin >> deptSales;

    string bestName = deptName; //Initialize the best name to the first dept's name
    double bestSales = deptSales; //Initialize the best sales to the first dept's sales
    for (int dept = 2; dept <= 17; dept++) // Repeat for the rest of the departments
    {
        cout << "Enter the name for department " << dept << endl;
        cin >> deptName;
        cout << "Enter the sales for department " << deptName << endl;
        cin >> deptSales;
        //if the sales is higher than the current best sales, then replace with this dept
        if (deptSales > bestSales)
        {
            bestSales = deptSales;
            bestName = deptName;
        }
    }
    //The final result can only be displayed when all the departments have been considered
    cout << "The department with the best sales is " << bestName
        << " with sales of " << bestSales << endl;
    return 0;
}
```

5.2.8 NESTED FOR LOOPS IN C++

Using a loop within another loop is called a **nested loop**.

Now let's try Example 1 on page 9 which displays the first 5 multiples of 4 (starting with 4), followed by their sum.

```
for (int i = 1; i <= 3; i++)
{
    for (int j = 1; j <= 2; j++)
        cout << i << " " << j << endl;
    cout << endl;
}
```

The first loop, controlled by the value of *i*, is called the **outer loop**. The second loop, controlled by the value of *j*, is called the **inner loop**.

The following program is an implementation of example 2 on page 10 – 11 in C++:

```
int main()
{
    //Determine and display, for 3 salespersons, their weekly sales as well as
    //the salesperson with the highest sales

    double highestSales = 0.0;
    string highestName = " ", salesName;
    double weeklySales, daySales;

    for (int sp = 1; sp <= 3; sp++)    // Loop for the 3 salespersons
    {
        cout << "Enter the salesperson\'s name\n ";
        cin >> salesName;
        weeklySales = 0.0;
        cout << "For each day of the week, enter the sales for the day:\n";
        for (int day = 1; day <= 5; day++)    // Loop for the 5 days of the week
        {
            cout << "    for day "<< day<<endl;
            cin >> daySales;
            weeklySales += daySales;
        }

        cout << "The weekly sales for " << salesName << " is "
            << weeklySales <<endl;

        if (weeklySales > highestSales)    // Determine the highest sales
        {
            highestSales = weeklySales;
            highestName = salesName;
        }
    }
    cout << "The salesperson of the week is " << highestName <<endl;
    return 0;
}
```

5.2.9 ADDITIONAL C++ READING AND EXERCISES

In addition to these notes, study the following pages on **Chapter 5** inside the prescribed textbook.

- **Page 85 – 90 (Assignment Variations)**
- **Page 201 – 216 (for loop statement)**

Do the following exercises:

- **EXERCISES 5.3** Page 212- 216
 - Do all the exercises (1 – 18)

5.3 Repetition using the Do-loop

When we know exactly how many times a loop must be executed, a for-loop is used. Very often we do not know how many times a loop is to execute, and then we cannot use the for-loop. In these cases we would use the do-loop.

In order to illustrate the concept of a loop where we don't know how many times the loop must be executed, we can use the analogy of people waiting in a queue to buy a ticket. If there is exactly 25 people in the queue, we can let the loop repeat 25 times, but more people may join the queue, and then we don't know how many times the loop is to execute. A different condition may then terminate the execution e.g.:

- All the tickets are sold;
- The ticket office closes at 16h00;
- All the people in the queue have bought their tickets;
- The ticket office has not been opened.

When making provision for all these conditions, we really don't know when the loop will stop. The algorithm needs to test these conditions to enable it to stop, otherwise we get an endless loop.

There are two types of do-loops. When designing a solution that has a number of statements to be repeated, the programmer must decide which one to use.

5.3.1 The Pre-test loop (do-while statement)

In this statement the condition is tested before any statement in the body of the loop is executed, and is therefore called a **pre-test loop**.

The body of the loop will execute as long as the **condition is true**. When the condition becomes false, the loop is exited. If the condition is false to start with, the body of the loop will never be executed.

The condition can also be a compound condition.

General format of the do-while statement:

```
do while condition
    statement1
    statement2
    .
    .
    statementn
loop
```


For the ticket example, the structure of the loop could be:

```
do while time <= 1600 AND noTicketsLeft > 0
    statement1
    statement2
    .
    .
    statementn
loop
```

5.3.2 The Post-test loop (do-loop-until statement)

This do-loop is called a post-test loop, because the condition is tested for the first time after the body of the loop has been executed. Therefore the body of the loop will always execute at least once.

This loop is executed as long as the **condition is false**, when it becomes true, it will be exited.

General format of the do-loop-until statement:

```
do
    statement1
    statement2
    .
    .
    statementn
loop until condition
```

As an example, say that there is only 100 tickets available, but there are more than 100 people in the queue, then the loop will be:

```
do
    statement1
    statement2
    .
    .
    statementn
loop until noTicketsLeft = 0
```

5.3.3 Terminating the execution of a do-loop

For both the pre-test and post-test loops, there must be statement(s) that enables the condition tested in the loop control statement to change. If it is omitted, the loop will be an endless loop.

In our ticket example, the following statement, in bold, could lead to the loop terminating:

```

noTickets = 100
do
    display "How many tickets do you want to buy?"
    enter noTicketsToBuy
    if noTicketsToBuy <= noTickets then
        noTickets = noTickets – noTicketsToBuy
    else
        display "Only ", noTickets, " are available"
    endif
loop until noTicketsLeft = 0

```

Another way to terminate a loop is to make use of a **sentinel**. A sentinel is a suitable value, decided on by the programmer, to indicate the end of the processing. For instance, the programmer can decide, when entering a list of marks, to terminate when a value of -1 is entered for the mark:

```

display "Enter a mark (-1 to end)"
enter mark
do while mark <> -1
    statement(s)
    :
    display "Enter a mark (-1 to end)"
    enter mark
loop

```

It is very important to note that, for a pre-test loop, the variable that is tested in the condition (the loop control-variable) needs to get a value before entering the loop, as the condition is tested before entering the loop. In our example, mark must receive a value, so that it can be tested against -1 in the condition.

A next value will have to be entered at the bottom of the loop, just before the loop clause, as the loop clause will return control to the condition that is then tested again.

The user of the program also needs to be informed, in the prompt, of what value to enter to terminate the loop.

5.3.4 Examples of do-while loops

Example 1

This algorithm enters an even number, and a maximum sum value. If an odd number was entered, ask the user to re-enter, until an even number has been entered. Then calculate and display the number of even numbers that were accumulated, starting from the first even number entered, and terminate the loop when the sum is greater than the second entered number.

Planning:

	Description	Type	Variable Name
Input:	Starting even number	integer	evenNum
	Maximum sum	integer	maxSum
Intermediate:	Sum	integer	sum
Output:	Count of even numbers	integer	count

I	P	O
evenNum maxSum	Initialize sum Prompt for & enter first even number Validate number Prompt for & enter the maximum sum For each even number: Determine the sum Count the number of even numbers Display count values	count

Algorithm:

CalcNumEvenNumbers

```
sum = 0  
count = 0
```

```
display "Enter an even number"  
enter evenNum  
do while evenNum mod 2 = 1  
    display "You have entered an odd number, please re-enter"  
    enter evenNum  
loop
```

```
display "Enter the maximum sum"  
enter maxSum
```

```
do while sum <= maxSum  
    count = count + 1  
    sum = sum + evenNum  
    evenNum = evenNum + 2  
loop
```

```
display count " number of even numbers were added together where the sum is ", maxSum,  
" or less"
```

end

First we have to ensure that we understand what is to be done before the loop, inside the loop and after the loop:

- The 3 statements before the first loop will initialize the variables to be used in the second loop.
- The first loop ensures that the number entered is indeed an even number, and it will keep on executing as long as an odd number was entered. Also note that the even number is entered before the loop, so that it can be tested for the first time in this pre-test loop. The even number is prompted for and re-entered inside this loop so that the condition can be tested with the new value, and eventually can become false in order to exit the loop.
- The maximum sum is entered before the second loop, so that the sum can be tested against its value for the first time.
- The statement **sum = sum + evenNum** will eventually cause the condition of the loop to become false, and the loop to be exited.

Example 2

Mr. Mabena teaches Programming 1. His students write 5 tests each semester. The average of the 5 tests gives the predicate. The average of the predicate and the exam mark gives the final mark. For each student in his class, enter the student number, 5 test marks and the exam mark (as a percentage), calculate and display the predicate mark and final mark. Then also determine and display the student number of the student with the highest final mark, with his/her mark. Also calculate and display the average final mark the students have obtained. Use the student number as the sentinel, and terminate when -1 for it is entered.

Planning:

	Description	Type	Variable Name
Input:	Student number	integer	studNum
	Test mark	real	testMark
	Exam mark	real	examMark
Intermediate:	Sum of test marks	real	sumMarks
	Sum of final marks	real	sumFinMarks
	Student count	real	studCount
	Variable to control the tests	integer	tst
Output:	Predicate mark	real	predMark
	Final mark	real	finMark
	Student number of the student with the highest mark	integer	studNumHighest
	Highest mark	real	highestMark
	Student average	real	studAve

I	P	O
For each student: studNum For each test: testMark examMark	Initialize variables For each student: Prompt for & enter student number For each test: Prompt for & enter test mark Accumulate sum Calculate the predicate Prompt for & enter the exam mark Calculate the final mark Display the predicate & final mark Determine the student number with the highest mark Accumulate all final marks Count the number of students Display the student number & mark of the student with the highest mark Calculate the student average Display the student average	For each student: predMark finMark studNumHighest highestMark studAve

Algorithm:

DoMarkCalculations

```

sumFinMarks = 0
studCount = 0
studNumHighest = 0
highestMark = 0

display "Please enter the student number (-1 to end)"
enter studNum
do while studNum <> -1
    sumMarks = 0
    display "For ", studNum, " please enter:"
    for tst = 1 to 5
        display "          Test mark ", tst, ":"
        enter testMark
        sumMarks = sumMarks + testMark
    next tst
    predMark = sumMarks / 5
    display "          The exam mark"
    enter examMark
    finMark = (predMark + examMark)/2

```

```

        display "Predicate mark = ", predMark, " and Final mark = ", finMark

        sumFinMarks = sumFinMarks + finMark
        studCount = studCount + 1

        if finMark > highestMark then
            highestMark = finMark
            studNumHighest = studNum
        endif

        display "Enter student number (-1 to end)"
        enter studNum
    loop
    studAve = sumFinMarks / studCount
    display "The average mark obtained by the students is ", studAve, "%"

    display "The student with the highest final mark is ", studNumHighest, " with a final mark of ",
        highestMark, "%"
end

```

Notes:

- The first two statements initialize the accumulator and counter. A counter is needed because for us to be able to determine the average final mark the sum of all the marks must be divided by the number of students in the class, which is unknown before the loop starts executing.
- The next two statements initialize the two variables needed to determine the student number of the student with the highest mark and the highest mark.
- Because a pre-test loop is used, a value needs to be entered for the variable used as the sentinel (in this case studNum), so that the loop condition can be tested for the first time. Should the condition be false from the start (if a -1 was entered before the loop), the body of the loop will not be executed at all.
- When the body of the loop is entered, please note that we don't prompt and read the student number again, as we have read the first student number just before entering the loop. We therefore only enter the other input values for that student.
- Because we have to enter 5 test marks, it is better to use a loop than defining five different variables, and then prompt and read each value into these 5 variables. Because we know that there are 5 test marks, a for-loop is used. In this case the for-loop is nested inside the do-loop. Because the sum of these five marks needs to be determined for each student, the accumulator sumMarks is initialized inside the do-loop, just before the for-loop. The predicate mark is determined after the for-loop, when the total of the 5 test marks is known.
- Because the predicate- and final mark need to be displayed for every student, this display is inside the do-loop.
- Because there is only one best student, and only one class average, it is displayed after the loop is exited. The determining of these values is inside the loop.
- The last two statements before the loop-clause will prompt and enter the next student number, and control is returned to the testing of the loop condition again. These statements enable the user to eventually enter a -1 for studNum, which will make the loop condition to become false and the loop to terminate.

5.3.5 Examples of do-until loops

Example 1

Mary wants to lose at least 10kg in weight. Enter her starting weight. Then, for each day, enter her weight for that day. Calculate and display the weight loss for that day. Also calculate and display how many days it took her to lose the 10kgs, as well as the number of the day she lost the most weight, and her weight loss for that day.

Planning:

	Description	Type	Variable Name
Input:	Starting weight	real	startWeight
	Day weight	real	dayWeight
Intermediate:	Total weight loss	real	totWeightLoss
Output:	Weight loss for the day	real	weightLossDay
	Number of days	real	numDays
	Day number of day with the highest weight loss	real	dayNumHighest
	Highest weight loss	real	highestWeightLoss

I	P	O
startingWeight For each day: dayWeight	Prompt for & enter starting weight Initialize variables For each day: Prompt for & enter the day's weight Calculate and display day's weight loss Count the number of days Accumulate total weight loss Determine the highest weight loss Display number of days & the highest values	For each day: weightLossDay numDays dayNumHighest highestWeightLoss

Algorithm:

```

DoWeightLossCalculations
~ This program calculates and displays Mary's weight-loss effort

    display "Enter Mary's starting weight in kgs"
    enter startingWeight
  
```

```

~ Initialize counters, accumulators and highest values
numDays = 1
dayNumHighest = 0
highestWeightLoss = 0
totWeightLoss = 0

do
    display "Please enter Mary's weight after day ", numDays
    enter dayWeight
    weightLossDay = startingWeight - dayWeight
    display "Weight loss for day ", numDays, " is ", weightLossDay, "kg"

    if weightLossDay > highestWeightLoss then
        highestWeightLoss = weightLossDay
        dayNumHighest = numDays
    endif

    totWeightLoss = totWeightLoss + weightLossDay
    startingWeight = dayWeight    ~ Change starting weight to the new weight to be
                                   ~ able to calculate the next day's weight loss
    if totWeightLoss < 10 then    ~ Prevent adding another day if target is reached
        numDays = numDays + 1
    endif
until totWeightLoss >= 10        ~ Mary wants to lose at least 10kg

display "It took Mary ", numDays, " to lose ", totWeightLoss
display "Mary lost the most weight on day ", dayNumHighest, " at ", highestWeightLoss, "kg"
end

```

Example 2

In this example, Example 2 of the do-while loop on page 23 is implemented using a do-loop-until loop, in order to illustrate the concept of guarding against processing the sentinel value of the loop.

Algorithm:

```

DoMarkCalculations

sumFinMarks = 0
studCount = 0
studNumHighest = 0
highestMark = 0

do
    display "Please enter the student number (-1 to end)"
    enter studNum

    if studNum <> -1 then ~ Do not process the sentinel value
        sumMarks = 0
        display "For ", studNum, " please enter:"
        for tst = 1 to 5
            display "    Test mark ", tst, ":"
            enter testMark
            sumMarks = sumMarks + testMark
        endfor
    endif
until studNum = -1

```



```

        next tst
        predMark = sumMarks / 5
        display "          The exam mark"
        enter examMark
        finMark = (predMark + examMark)/2

        display "Predicate mark = ", predMark, " and Final mark = ", finMark

        sumFinMarks = sumFinMarks + finMark
        studCount = studCount + 1

        if finMark > highestMark then
            highestMark = finMark
            studNumHighest = studNum
        endif
    endif

loop until studNum = -1

studAve = sumFinMarks / studCount
display "The average mark obtained by the students is ", studAve, "%"

display "The student with the highest final mark is ", studNumHighest, " with a final mark of ",
highestMark, "%"
end

```

Exercises:

1. Provide the exact output that will be produced by the following pseudocode segments:

1.1

```

d = 4
e = 6
f = 7
do while d > f
    d = d + 1
    e = e - 1
loop
display d, " ", e, " ", f

```

1.2

```

j = 2
k = 5
m = 6
n = 9
do while j < k
    do while m < n
        display "Hello"
        m = m + 1
    loop
    j = j + 1
loop

```

2. Do the planning and write an algorithm to solve the following problems:
- 2.1 The Hollywood Movie Rating Guide wants you to design a program that will be installed in a kiosk at their theatres. Each movie-goer enters a value from 0 to 5 indicating the number of stars that he/she awards the Guide's feature movie of the week. If a user enters a star value that does not fall within the range, re-prompt the user continuously until the correct value is entered. The program executes continuously until the theater manager enters a negative number to quit. At the end of the program, display the average star rating for the movie. Use a do-loop-until to solve this problem.
- 2.2 **KeepFit Gym**, offers a free service to all clients and potential clients to calculate their body mass index. The formulae to calculate the body mass index is as follows:

$$\text{BMI} = \frac{\text{weight (kg)}}{\text{height}^2(\text{m}^2)}$$

Clients are then categorised as follows, depending on the value of BMI:

Category	BMI range - kg/m ²
Starvation	less than 14.9
Underweight	from 15 to 18.4
Normal	from 18.5 to 22.9
Overweight	from 23 to 27.5
Obese	from 27.6 to 40
Morbidly Obese	greater than 40

Determine the body mass index for a number of people who made use of this free offer during a certain day.

For every person, enter his/her weight in kg and height in meters. Then calculate and display the body mass index and corresponding category. A weight of zero must terminate the process.

At the end of the day, the total number of people in each category must be displayed.

Use a do-while loop to solve this problem.

- 2.3 People who want to hire a machine to wash their carpets usually go to the **WishyWashy Company** where they can hire one of three different types of machines – A, B or C for heavy, ordinary or light duty washes respectively. The prices are as follows:

Type of Machine	Initial Cost	Additional Cost per Hour or Part Thereof
A	R50.00	R30.00
B	R62.50	R36.25
C	R74.87	R40.50

The input is the name of the client, the code to indicate the type of machine, and the time the machine was used (in minutes). The amount must be calculated by adding the initial cost to the calculated cost for the time used. (Use a select case). VAT of 14% must be added to give the final amount due by the client. Do this for all clients for the day. Use the client name as your sentinel, with a value of END to indicate the end of the clients. At the end of the program, the total amount, the total VAT amount and the total final amount due to **WishyWashy** must be displayed. Use any loop you feel is appropriate.

5.3.6 THE WHILE STATEMENT IN C++

C++ provides the while statement for coding a **pre-test loop**.

Syntax

```
while (condition)
    statement; /* either one statement or a statement block to be processed as long
               as the condition is true */
```

- The keyword **while** tells the computer to create an iteration control structure.
- The condition must be a Boolean expression, which is an expression that evaluates to either true or false. The expression can contain variables, constants, arithmetic operators, relational operators, and logical operators.
- The parentheses that surround the condition must always be included, similar to the **if** statement.
- Besides providing the condition, the programmer must also provide the loop's body statements, which are the statements to be processed when the condition evaluates to true. If more than one statement needs to be processed, the statements must be entered as a statement block by enclosing them in a set of braces ({}).
- The body must include a statement that will alter the value of the variable used to construct the condition to avoid an **infinite** or **endless** loop.

A loop whose instructions are processed indefinitely is referred to as either an endless loop or an infinite loop.

In C++ the structure of the ticket example demonstrated earlier could be:

```
while (time <= 1600 && noTicketsLeft > 0)
{
    statement1;
    statement2;
    .
    .
    statement to alter time or/and noTicketsLeft;
}
```

Now let's try Example 1 on page 21 and 22, to write a C++ program that enters an even number, and a maximum sum value. If an odd number was entered, ask the user to re-enter, until an even number has been entered. Then calculate and display the number of even numbers that were accumulated, starting from the first even number entered, and terminate the loop when the sum is greater than the second entered number.

```
int main()
{
    int sum = 0;
    int count = 0;

    int evenNum, maxSum;

    cout << "Enter an even number:\n ";
    cin >> evenNum;
    while (evenNum % 2 == 1)
    {
        cout << "You have entered an odd number, please re-enter: ";
        cin >> evenNum;
    }

    cout << "Enter the maximum sum:\n ";
    cin >> maxSum;

    while (sum <= maxSum)
    {
        count++;
        sum += evenNum;
        evenNum += 2;
    }

    cout << count << " number of even numbers were added together where the sum is "
        << maxSum << " or less\n";
    return 0;
}
```

Once more, let's try Example 2 on page 23 to 25 using a C++ while loop:

```
int main()
{
    int sumFinMarks = 0;
    int studCount = 0;
    int studNumHighest = 0;
    int highestMark = 0;
    int sumMarks, finMark, examMark, predMark;
    cout << "Please enter the student number (-1 to end): ";
    cin >> studNum;
    while (studNum != -1)
    {
        sumMarks = 0;
        cout << "For " << studNum << "please enter: ";
        for(int tst = 1; tst <= 5; tst++)
        {
            cout << "      Test mark " << tst << " : ";
            cin >> testMark;
            sumMarks = sumMarks + testMark;
        }
        predMark = sumMarks / 5;
        cout << "      The exam mark: ";
        cin >> examMark;
        finMark = (predMark + examMark)/2;
        cout << "Predicate mark = " << predMark << " and Final mark = " << finMark << endl;
        sumFinMarks = sumFinMarks + finMark;
        studCount = studCount + 1;
        if (finMark > highestMark)
        {
            highestMark = finMark;
            studNumHighest = studNum;
        }
        cout << "Enter student number (-1 to end)\n";
        cin >> studNum;
    }
    studAve = sumFinMarks / studCount;
    cout << "The average mark obtained by the students is " << studAve << "%\n";

    cout << "The student with the highest final mark is " << studNumHighest
        << " with a final mark of " << highestMark << "%\n";
    return 0;
}
```

5.3.7 THE DO WHILE STATEMENT IN C++

C++ provides the do while statement for coding a **post-test loop**.

Syntax:

```
do //begin loop
{
    statement1;
    statement2;
    :
    statementn;
} while (condition);
```

- The do and while are C++ keywords,
- As in the while statement, the condition surrounded by parentheses must evaluate to either true or false.
- The condition may contain variables, constants, arithmetic operators, relational operators, and logical operators
- Besides providing the condition, the programmer also must provide the statements to be processed when the condition evaluates to true. If more than one statement needs to be processed, the statements must be entered as a statement block by enclosing them in a set of curly braces {}.

Let's try Example 1 on page 26 to 27 using a C++ do while loop:

```
int main()
{
    //This program calculates and displays Mary's weight-loss effort
    double startingWeight;
    double totWeightLoss =0;

    cout<< "Enter Mary's starting weight in kgs:\n ";
    cin >> startingWeight;

    //Initialize counters, accumulators and highest values
    int numDays = 1;
    int dayNumHighest = 0;
    double highestWeightLoss = 0;
    double dayWeight, weightLossDay;

    do
    {
        cout << "Please enter Mary's weight after day :" << numDays<<": ";
        cin >> dayWeight;
        weightLossDay = startingWeight - dayWeight;
        cout << "Weight loss for day " << numDays << " is " << weightLossDay << "kg\n";

        if (weightLossDay > highestWeightLoss)
        {
            highestWeightLoss = weightLossDay;
            dayNumHighest = numDays;
        }

        totWeightLoss = totWeightLoss + weightLossDay;
        startingWeight = dayWeight;    // Change starting weight to the new weight to be
                                      // able to calculate the next day's weight loss
        if (totWeightLoss < 10 ) // Prevent adding another day if target is reached
            numDays = numDays + 1;

    } while (totWeightLoss <= 10);      //Mary wants to lose at least 10kg

    cout << "It took Mary " << numDays <<" to lose " <<totWeightLoss<<endl;
    cout <<"Mary lost the most weight on day " <<dayNumHighest << " at "
        << highestWeightLoss<< "kg\n";
    return 0;
}
```

Let's try Example 2 on page 27 to 28 using a C++ do while loop:

```
int mian()
{
    int sumFinMarks = 0;
    int studCount = 0;
    int studNumHighest = -1;
    int highestMark = 0;
    int sumMarks, finMark, examMark, predMark, testMark;
    do
    {
        cout << "Please enter the student number (-1 to end): ";
        cin >> studNum;

        if (studNum != -1)    // Do not process the sentinel value
        {
            sumMarks = 0;
            cout << "For " << studNum << " please enter: ";
            for (int tst = 1; tst <= 5; tst++)
            {
                cout << "      Test mark "<< tst << ":\n";
                cin >> testMark;
                sumMarks = sumMarks + testMark;
            }
            predMark = sumMarks / 5;
            cout << "      The exam mark: ";
            cin >> examMark;
            finMark = (predMark + examMark)/2;

            cout << "Predicate mark = " << predMark << " and Final mark = "
                 << finMark<<endl;

            sumFinMarks = sumFinMarks + finMark;
            studCount = studCount + 1;

            if (finMark > highestMark)
            {
                highestMark = finMark;
                studNumHighest = studNum;
            }

        }

    } while( studNum != -1);

    studAve = sumFinMarks / studCount;
    cout << "The average mark obtained by the students is "<< studAve << "%\n";

    cout << "The student with the highest final mark is "<< studNumHighest
         << " with a final mark of " << highestMark << "%\n";
    return 0;
}
```

break and **continue** Statements

The **break statement** provides an early exit of an iteration control structure (for, while or do while). After the break statement executes, the program continues to execute with the first statement after the structure. The program below will terminate the loop when a negative value or zero is entered:

```
int multiple = 1;
while (true)
{
    cout << "Enter any positive integer: ";
    cin >> num;
    if (num <=0 )
    {
        break;
    }
    multiple *= num; // will only execute when num > 0
}
```

The **continue statement** is used in an iteration control structure to skip the remaining statements in the loop and proceeds with the next iteration of the loop. When the continue statement is encountered inside a while or a do while, the computer will evaluate the condition to decide whether to execute the next iteration. The program below will continue to prompt the user when a negative value or zero is entered:

```
int multiple = 1;
int count = 0;
while (count < 5)
{
    cout << "Enter any positive integer: ";
    cin >> num;
    if (num <=0 )
    {
        continue;
    }
    // the following statements will only execute when num > 0
    multiple *= num;
    count ++;
}
```


5.3.8 SUMMARY

In addition to these notes, study the following pages on **Chapter 5** inside the prescribed textbook.

- **Page 85 – 90 (Assignment Variations)**
- **Page 179 – 224 (repetition)**

Do the following exercises:

- **EXERCISES 5.1** Page 186- 188
 - Do all the exercises (1 – 7)
- **EXERCISES 5.2** Page 198- 201
 - Do all the exercises (1 – 12)
- **EXERCISES 5.3** Page 212- 216
 - Do all the exercises (1 – 18)
- **EXERCISES 5.4** Page 220- 220
 - Do all the exercises (1 – 4)

© COPYRIGHT: TSHWANE UNIVERSITY OF TECHNOLOGY (2020)
Private Bag X680
PRETORIA
0001