# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Aims and objectives</td>
<td>1</td>
</tr>
<tr>
<td>Learning outcomes</td>
<td>2</td>
</tr>
<tr>
<td>How to use this subject guide</td>
<td>2</td>
</tr>
<tr>
<td>Reading</td>
<td>3</td>
</tr>
<tr>
<td>Online study resources</td>
<td>3</td>
</tr>
<tr>
<td>Time management</td>
<td>5</td>
</tr>
<tr>
<td>The examination</td>
<td>5</td>
</tr>
<tr>
<td>Syllabus</td>
<td>6</td>
</tr>
<tr>
<td><strong>Chapter 1: General problem-solving</strong></td>
<td>7</td>
</tr>
<tr>
<td>Essential reading</td>
<td>7</td>
</tr>
<tr>
<td>Aims of the chapter</td>
<td>7</td>
</tr>
<tr>
<td>Learning outcomes</td>
<td>7</td>
</tr>
<tr>
<td>Introduction</td>
<td>7</td>
</tr>
<tr>
<td>Problem-solving</td>
<td>8</td>
</tr>
<tr>
<td>Problem-solving with computers</td>
<td>9</td>
</tr>
<tr>
<td>Reminder of learning outcomes</td>
<td>9</td>
</tr>
<tr>
<td>Sample examination questions</td>
<td>9</td>
</tr>
<tr>
<td><strong>Chapter 2: Problem-solving with a computer</strong></td>
<td>11</td>
</tr>
<tr>
<td>Essential reading</td>
<td>11</td>
</tr>
<tr>
<td>Aim of the chapter</td>
<td>11</td>
</tr>
<tr>
<td>Learning outcomes</td>
<td>11</td>
</tr>
<tr>
<td>Introduction</td>
<td>11</td>
</tr>
<tr>
<td>Constants and variables</td>
<td>11</td>
</tr>
<tr>
<td>Data types</td>
<td>12</td>
</tr>
<tr>
<td>Functions</td>
<td>14</td>
</tr>
<tr>
<td>Operators</td>
<td>14</td>
</tr>
<tr>
<td>Expressions and equations</td>
<td>15</td>
</tr>
<tr>
<td>Reminder of learning outcomes</td>
<td>16</td>
</tr>
<tr>
<td>Sample examination questions</td>
<td>16</td>
</tr>
<tr>
<td><strong>Chapter 3: Programming concepts</strong></td>
<td>17</td>
</tr>
<tr>
<td>Essential reading</td>
<td>17</td>
</tr>
<tr>
<td>Aim of the chapter</td>
<td>17</td>
</tr>
<tr>
<td>Learning outcomes</td>
<td>17</td>
</tr>
<tr>
<td>Introduction</td>
<td>17</td>
</tr>
<tr>
<td>Some basics</td>
<td>17</td>
</tr>
<tr>
<td>Testing your solution in the examination</td>
<td>22</td>
</tr>
<tr>
<td>Reminder of learning outcomes</td>
<td>22</td>
</tr>
<tr>
<td>Sample examination question</td>
<td>22</td>
</tr>
<tr>
<td><strong>Chapter 4: Program structure</strong></td>
<td>23</td>
</tr>
<tr>
<td>Essential reading</td>
<td>23</td>
</tr>
<tr>
<td>Aim of the chapter</td>
<td>23</td>
</tr>
<tr>
<td>Learning outcomes</td>
<td>23</td>
</tr>
</tbody>
</table>
Chapter 9: Arrays

Aim of the chapter ....................................................................................................... 43
Essential reading ......................................................................................................... 43
Sample examination questions ..................................................................................... 43
Reminder of learning outcomes..................................................................................... 43
Two-dimensional arrays ............................................................................................... 45
One-dimensional arrays ............................................................................................... 43
Introduction ................................................................................................................ 43
Learning outcome ........................................................................................................ 43
Learning outcomes ...................................................................................................... 129
Aim of the chapter ....................................................................................................... 43
Essential reading ......................................................................................................... 43
Sample examination questions ..................................................................................... 43
Reminder of learning outcomes..................................................................................... 43
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Cohesion and coupling ................................................................................................. 23
Local and global variables........................................................................................... 25
Parameters .................................................................................................................. 25
Reminder of learning outcomes..................................................................................... 27
Sample examination questions ..................................................................................... 27
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
The sequential logic structure ....................................................................................... 30
Reminder of learning outcomes..................................................................................... 27
Parameters .................................................................................................................. 25
Welcome to this half course, **129 Introduction to programming**. For those of you who are studying this course as part of the BSc Information Systems and Management, as you study the courses on this degree you will soon appreciate that contemporary information systems have substantial technical components. Indeed, most of the various technologies that are used in the information systems of modern organisations are studied by dedicated disciplines, such as software engineering, telecommunications, networks, databases, human–computer interfaces and others. Remember, however, that in your degree, you are not specialising in the engineering of technologies, but in the way people can develop, use and manage technology applications in the context of an organisation. You will not therefore study particular technologies in full depth. Yet it is important to understand and to have a reasonable working knowledge of some of the fundamentals of information technologies. For example, you need to have some knowledge of programming before you can learn to structure and manage information systems development projects that involve the production of software.

This is the motivation for this course, **129 Introduction to programming**. It is a course designed to let you gain the necessary knowledge from which you can learn most programming languages and understand the basics from which complex information systems can be built. In this course, we take a different view to those courses that introduce you to programming but fail to take account of the basics. They begin by teaching you about the fundamentals of a **programming language** and not **programming**. In our experience, we find students who learn languages such as, for example, Visual Basic understand user interface development, and those who learn Java and C++ understand object orientation. Both learn skills that are sometimes hard to transfer. Also, the complexities of these languages obscure the basic message – how do I program?

In this course, we introduce you to programming by equipping you with general skills that you can then use to learn a specific programming language. Note that you might also hear of different programming ‘paradigms’ such as procedural, modular, object-oriented and functional. In this course we investigate both procedural and modular. Both of these are foundations for object-oriented approaches used in Java and C++. Functional approaches (such as those used in LISP) are outside of the scope of this course.

**Aims and objectives**

The overall aim of this course is to equip you with skills that will enable you to write algorithms in structured English or pseudocode. You will find individual outcomes outlined at the beginning of each chapter. You should take note of these outcomes and use them to set up a study programme for this subject. You might like to do this by making a list of these to give you a ‘global’ view of your course and then ticking them off once you feel that you have achieved them.
The main aims and objectives of this course are to:

- introduce students to programming languages and their world views
- develop understanding of software development lifecycles
- develop understanding of how a programming problem is recognised and how a solution to the problem can be designed
- develop understanding of primitive and complex data structures
- develop understanding of programming structures
- develop understanding of how programs can be tested
- develop understanding of how programs can be documented.

**Learning outcomes**

At the end of this half course, and having completed the Essential reading and activities, you should be able to:

- compare the merits of various approaches to programming
- recognise, design and produce a solution to a programming problem
- identify and use fundamental program structures
- identify and use primitive data types to store information
- manipulate data using algorithmic expressions
- produce efficient solutions by using control structures and conditional statements
- compile, run and effectively debug a simple program
- design data structures to store more complex information
- write a clearly annotated program with meaningful identifier names
- develop and use software testing methods that help to improve the quality of programs.

**How to use this subject guide**

It is often said that 'practice makes perfect'. Due to the nature of programming the only method for learning it is to continually practise the techniques provided. We have supplied some examples and activities to help you to practise and we encourage you to try some of the activities and to refer to the essential reading. You may be frustrated at first when you try to create your solutions to these exercises. Keep trying - you will get there! Just remember that a lack of discipline is often the downfall of the programmer. Make sure you are organised and think about what you are doing when you are programming. A fully tested, working algorithm\(^5\) is a wonderful thing!

On a practical note, this subject guide focuses on what we consider to be the most important principles of programming that you need to know when starting on the road to program development. There are many more. We have also listed some textbooks that we feel you can use to go further in understanding how to program in specific languages. These books provide excellent guidance for students working alone and we recommend that you attempt to implement the solutions to your activities in a particular programming language.

\(^5\) An algorithm is a set of general instructions in a computer program that use the elements of programming (and their rules) to get the computer to perform a certain task.
Reading

Normally, you will find advice on what books you need to buy or otherwise have access to at the beginning of each chapter of a guide. You should not continue to use an old edition of a textbook if a new edition has been published. Always buy the latest edition of recommended textbooks; if the latest edition is more recent than the edition to which the subject guide refers, use the index and tables of contents to identify the relevant parts of the new edition. In this guide, however, we only use a single book. You should read this in parallel with the guide. Also, check the virtual learning environment (VLE) regularly for updated guidance on readings.

Essential reading

The Essential reading for this subject is:


Detailed reading references in this subject guide refer to the editions of the set textbooks listed above. New editions of one or more of these textbooks may have been published by the time you study this course. You can use a more recent edition of any of the books; use the detailed chapter and section headings and the index to identify relevant readings. Also check the VLE regularly for updated guidance on readings.

Further reading

The Further reading covers introductions to one language: Visual Basic. We recommend that you choose this language rather than Java or C++ as it is better suited to the development of applications with user interfaces. Java and C++ have support for these but to some extent are more complex. We therefore recommend that you attempt to learn Java or C++ once you have mastered Visual Basic. We recommend the following texts:


Please note that as long as you read the Essential reading you are then free to read around the subject area in any text, paper or online resource. You will need to support your learning by reading as widely as possible and by thinking about how these principles apply in the real world. To help you read extensively, you have free access to the VLE and University of London Online Library (see below).

Online study resources

In addition to the subject guide and the Essential reading, it is crucial that you take advantage of the study resources that are available online for this course, including the VLE and the Online Library.

You can access the VLE, the Online Library and your University of London email account via the Student Portal at: http://my.londoninternational.ac.uk

You should have received your login details for the Student Portal with your official offer, which was emailed to the address that you gave
on your application form. You have probably already logged in to the
Student Portal in order to register! As soon as you registered, you will
automatically have been granted access to the VLE, Online Library and
your fully functional University of London email account.

If you forget your login details at any point, please email uolia.support@
london.ac.uk quoting your student number.

The VLE

The VLE, which complements this subject guide, has been designed to
enhance your learning experience, providing additional support and a
sense of community. It forms an important part of your study experience
with the University of London and you should access it regularly.

The VLE provides a range of resources for EMFSS courses:

• Self-testing activities: Doing these allows you to test your own
understanding of subject material.

• Electronic study materials: The printed materials that you receive from
the University of London are available to download, including updated
reading lists and references.

• Past examination papers and Examiners' commentaries: These provide
advice on how each examination question might best be answered.

• A student discussion forum: This is an open space for you to discuss
interests and experiences, seek support from your peers, work
collaboratively to solve problems and discuss subject material.

• Videos: There are recorded academic introductions to the subject,
interviews and debates and, for some courses, audio-visual tutorials
and conclusions.

• Recorded lectures: For some courses, where appropriate, the sessions
from previous years' Study Weekends have been recorded and made
available.

• Study skills: Expert advice on preparing for examinations and
developing your digital literacy skills.

• Feedback forms.

Some of these resources are available for certain courses only, but we
are expanding our provision all the time and you should check the VLE
regularly for updates.

Making use of the Online Library

The Online Library contains a huge array of journal articles and other
resources to help you read widely and extensively.

To access the majority of resources via the Online Library you will either
need to use your University of London Student Portal login details, or you
will be required to register and use an Athens login:
http://tinyurl.com/ollathens

The easiest way to locate relevant content and journal articles in the
Online Library is to use the Summon search engine.

If you are having trouble finding an article listed in a reading list, try
removing any punctuation from the title, such as single quotation marks,
question marks and colons.

For further advice, please see the online help pages:
www.external.shl.lon.ac.uk/summon/about.php
**Time management**

We recommend that if you are studying over the course of one academic year, this half course takes at least three hours of study every week. Adjust this figure accordingly if you are studying this course over a longer period.

About one-third of your private study time should be spent reading textbooks and the other two-thirds doing problems.

As a help to your time management, we have converted the chapters and topics of this subject into approximate weeks of a typical university course, studying from, say, October until April/May. There is nothing formal in the figures given below: it is purely for indicative purposes. What you should gain from the following breakdown is an indication of the relative amounts of time to be spent on each topic.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

**The examination**

*Important:* the information and advice given in this section are based on the examination structure used at the time this guide was written. Please note that subject guides may be used for several years. Because of this we strongly advise you always to check both the current Regulations for relevant information about the examination, and the VLE where you should be advised of any forthcoming changes. You should also carefully check the rubric/instructions on the paper you actually sit and follow those instructions.

For some of you this may be the first time you have attempted an examination at degree level. You may be apprehensive and probably a little frightened. This section provides some advice on how you should approach the examination and what you should do when you are actually sitting the examination.

The course is examined by a two-hour unseen written examination. You can find a Sample examination paper at the end of this subject guide in Appendix 1.

You will be required to answer three questions in total. Section A will be one compulsory question, which is worth 60 marks, and then you will need to choose two questions from a choice of three questions in Section B, which are worth 20 marks each, giving a total of 100 marks.

You should ensure that you do answer three questions and allocate your time carefully in the examination. It is particularly important in this course to devote some time prior to answering each question to thinking about and planning your answer. Please write clearly and legibly.
**Work hard and work effectively.** There is a long build-up to this examination. This course may not appear to have as much content as some others that you will encounter during your degree. This is because it is an applied subject which requires a lot of thinking and you must also practise your problem-solving and writing your pseudocode programs. It is not essential, but it would also help to learn a programming language (such as Visual Basic) that will help you to learn how each of the logic structures works in this course. Keep practising and above all manage your time!

When you get to the examination try to relax. It is vitally important for you to read the examination paper carefully. The first question (Question 1, section A) will always be compulsory. You will then have to answer two of the three questions in Section B. If you have thought about and done the activities in Chapter 10, then you will have excellent preparation for the first question.

Always remember to read the questions carefully. **Never forget to discuss your answer.** The question might say, ‘When you have written your algorithm, discuss exactly how your algorithm satisfies these requirements.’ If you do not make this discussion then you could lose half your marks.

In Part b of Question 1, and in the other questions of Section B, if examples are required, give an example. If a definition is required, give a definition. If a discussion is required, then give a discussion. **Never skip part of what the question asks for; you will lose marks if you do.**

Remember, it is important to check the VLE for:

- up-to-date information on examination and assessment arrangements for this course
- where available, past examination papers and Examiners' commentaries for the course which give advice on how each question might best be answered.

We wish you good luck with this course and all the courses of your degree.

---

**Syllabus**

Selecting a programming language and world view. This topic briefly addresses the most common programming language world views (functional, modular, procedural and object-oriented) and investigates how to choose the most appropriate for a given problem.

Designing a solution. This topic discusses how a programming problem can be analysed to create an abstract, machine-independent solution design. The methods for the refinement of the abstract solution into a language-dependent solution are also investigated.

Implementing your solution. This topic investigates the program structures, control structures and selection structures that are available to the programmer to implement a given solution. The concepts of primitive and complex data structures are presented and discussed. As part of the implementation process, the programmer should produce programs that are understandable to other programmers. This investigates how a program can be documented to do this.

Testing your solution. Errors can occur during the design and implementation of a solution. This topic investigates such errors and what techniques can be used to rectify them. The use of a test plan to identify logical programming errors will also be discussed.
Chapter 1: General problem-solving

Essential reading

Aims of the chapter
The aims of this chapter are to:
• introduce an approach to problem-solving
• use this to solve a problem.

Learning outcomes
On completing this chapter, and the Essential reading and activities, you should be able to:
• list and describe the six steps to solve a problem that has an algorithmic solution
• use these six problem-solving steps to solve a problem.

Introduction
Let’s begin! But where? When we write a program we typically are doing it in order to solve a problem or to satisfy a need. For example, we might repeatedly need to add up bills at a restaurant or calculate the average of a set of examination marks. However, accurately identifying the problem is critically important to correct programming – you need to ensure you are solving the right problem. To set you on the right path, this chapter gives you an overview of what you need to do to find the right problem and figure out how to solve it. It covers:
• problem-solving
• problem types
• problem-solving with computers.

Note that we strongly advise you to follow the above in the essential reading (Sprankle). There are a huge number of programming books out there. We have carefully selected the one that focuses on problem-solving and developing a solution to the problem before you get lost in program language syntax. You can learn a programming language first but in our experience students who do this make the mistake of focusing on programming language problems and not programming problems. The set of skills you will learn here and in subsequent chapters can be applied to almost any programming language and will give you a solid foundation for future courses.
Problem-solving

How do we solve problems? Everyone has a way to do it but, at least for this course, we suggest that you follow the six steps that Sprankle suggests.

1. **Identify the problem.** What is the problem that you are trying to solve? For example, how do I get to work? Many problems in information systems are caused by people wrongly identifying the problem and creating a good solution for the wrong problem! In the examination for this course you will be given the details of the problem we want you to solve. In real life you will not. So think carefully and be prepared to start again if you have wrongly identified the problem. For example, the problem might not be how do I get to work but what is the best way to get to work?

2. **Understand the problem.** When you think you know what the problem is, try to understand it. There are many poor programs that are based on a poor understanding of the problem. If you are studying this course as part of the full BSc degree, when you reach course 138 Information and communication technology: principles and perspectives you will study human–computer interaction that will give you more skills in this area. For most problems that you encounter in this course your basic knowledge of mathematics should be enough to help you. In real-world problems, however, never be afraid to admit ignorance. If you are a programmer working for a client to solve one of their problems and you find yourself in doubt, you would (and should) ask the client. So taking this back to our example of ‘How do I get to work?’, a simple answer is that I can drive, get the bus or the train, walk, cycle, etc.

3. **Identify alternative ways to solve the problem.** How many ways can you think of solving the problem? In this case – are there any other ways of getting to work? In programming this often becomes a list of the different programming techniques you can think of to implement the program you design.

4. **Select the best way to solve the problem from your list.** How do you choose? You need selection criteria to do this. There are many of these. In the above example you could choose by saying ‘as fast as possible’. Experience is the key to choosing the best solution!

5. **List instructions that enable you to solve the problem with your selected solution.** For programs we will come back to this in the next chapter. In our small example the instructions could be, ‘Leave the house, turn left, walk 100m, etc.’

6. **Evaluate the solution.** Was your solution correct? How can you demonstrate it was correct? This is tricky as you need to decide which factors to judge your solution against. In programming we often call this testing. In the real world, success is also often judged by the satisfaction of the client. Did you find that your choice to get to work was the best? Did you think it could be improved? Problem-solving is often an iterative process of trying different solutions and modifying them to improve.

These steps are a written form of what we sometimes do to solve a problem. Use them to help you think about how to solve a problem and make sure that you use all the steps. Eventually, you will use them without thinking about it.

---

1 Iterative means taking steps that keep repeatedly refining your approach as you make it better and better and better.
Problem-solving with computers

Some terminology: we use solution to mean the list of instructions that you create to solve the problem that you have identified. Follow these instructions to produce an outcome or answer. We call these results. If the set of instructions has been implemented on a computer, we call these instructions a program. In the next chapter we shall look at the basic building blocks for a program that you can use to create a solution to produce results.

Activities

1. How do you buy a book? Where do you buy it from? How do you pay for it? Use the problem-solving approach suggested in this chapter to identify alternative solutions to these problems and to decide which is the best place to buy a book and how to go about it.

2. A football team has played several matches this season. In the next game a fan would like to know whether or not his team is expected to win. How do you find a solution to his problem? Use the problem-solving approach we described in this chapter to discover how.

Reminder of learning outcomes

Having completed this chapter, and the Essential reading and activities, you should be able to:

- list and describe the six steps to solve a problem that has an algorithmic solution
- use the six problem-solving steps to solve a problem.

Sample examination questions

1. How do you create an algorithmic solution to a problem? To answer this question, first describe the six steps used in problem-solving and then use the steps to solve an illustrative problem of your choice.

2. You have a new job! There are different transportation options that you can use to get to your new employer (car, bus, walking, train, etc.), but which should you use? Use the six problem-solving steps to show how you could develop an algorithmic solution to this problem.
Chapter 2: Problem-solving with a computer

Essential reading

Aim of the chapter
The aim of this chapter is to introduce you to the basic building blocks used in programming to create a solution to a problem.

Learning outcomes
On completing this chapter, and the Essential reading and activities, you should be able to:
• differentiate between variables and constants
• differentiate between character, numeric and logical data types
• identify operators, operands and resultants
• identify and use functions
• identify and use operators according to their placement in the hierarchy chart
• set up and evaluate expressions and equations using variables, constants, operators and the hierarchy of operations.

Introduction
The previous chapter introduced problem-solving. As Sprankle notes, there are generally three types of problems that can be solved on computers. These are:
• computational: problems involving some kind of mathematical processing, e.g. how long does it take to get to work?
• logical: problems involving decision-making, e.g. what is the best route to drive to work?
• repetitive: problems involving a repeated set of mathematical or logical processing, e.g. what is the best form of transport I can use to get to work?

All these need some basic concepts in computing. We will introduce you to these in this chapter. You should take care to become familiar with these before proceeding to other chapters.

Constants and variables
Constants and variables are two entities which are used to store information (technically called data) in your program. Data has values. For example, the cost of electricity on a bill is data and has a value. If the value of data changes (for example, the cost of electricity changes from bill to bill) then we say that the data value is variable. If it does not change
(for example, a fixed standing charge on a bill), we say the data value is **constant**. In programming, we can actually use these terms to describe data.

A **constant** is a value that never changes as the instructions in a program are followed. Constants can be any type of data (see later).

A **variable** is a value that does change as the instructions in a program are followed. Variables can be any type of data as well (see later).

Both constants and variables are given names. For example, Cost could be a variable that is used to refer to the cost of a bill. Rate could be a constant that refers to the rate used to calculate the cost of a bill. Technically speaking, in a computer program these refer to a location in computer memory that stores the value of the constant or data. We use the name of the constant or variable to refer to these locations rather than the memory address. This is done for various technical reasons that you will discover when you use actual programming languages on your own. For this course, the main reason why we use these names is to make the program clearer. You should always use names that are most meaningful to the end user (the person who is going to use the system you design) or to the problem. For example, if you receive monthly electricity bills and you would like to know the average monthly cost of electricity then you could write a program with variables Cost1, Cost2, etc., or CostJanuary, CostFebruary, etc. The first of these is not as meaningful as the second. This is very important as it helps to make the program more understandable. Sprankle uses seven rules to name constants and variables. These are:

1. Name a constant or variable according to what it represents.
2. Do not use spaces in a constant or variable name.
3. Start a constant or variable name with a letter, not a number.
4. Do not use a dash or hyphen in a name (the computer will think it a subtraction sign). An underscore is fine.
5. Names must be exact. Do not vary the spelling or format of a name when you use it.
6. Be consistent when you use UPPERCASE and lowercase characters. Some computers are case sensitive and treat these as being different.
7. Use a consistent naming convention. We use first letter capitalised, no spaces and then the first letter of each word capitalised. For example, ThisIsAVariable is correct but thisIsAVariable or Thisisavariable are not.

**Data types**

Constants and variables contain **data**. To make programs even more understandable, we can give data different types. This restricts the values that a constant or a variable can take. We say that a variable has a **data type**. In this course we use four data types. These are numeric, character, string and logical. There are two numeric data types. These are integer and real. Each data type has a set of valid values termed a **data set**. We now introduce the basic or primitive data types that are available to us.

The **numeric data types** are used to represent numbers such as 345, -10, 0, etc. There are two kinds of numeric data types. The **integer** data type is used to represent whole numbers such as -9, -3, 0, 1, 2, 3, etc. The **integer data set** is therefore whole numbers. The **real** data type is used to represent whole numbers with decimal parts (the **real data set**). For
example, these could be 0.001, 12.3, -2356.05, etc. The choice is down to the programmer.

NoStudentsInClass would tend to be an integer while AverageSpeed would most likely be a real. One can use reals for both, but would this confuse someone trying to understand your program? Finally, numeric data is used in numeric calculations.

The **character data type** is used to represent alphanumerical (letters and numbers) data. They are represented between single quote marks. Examples of these include ‘1’, ‘a’, ‘Z’ and ‘%’. The complete list of available characters has been standardised. The main example of this is the American Standard Code for Information Interchange (ASCII). This contains 256 characters and the full listing can be found in Sprankle (Appendix C). If you have chosen the character data type for a constant or a variable, they can only be used to represent a single alphanumerical character. For example, choice might be a variable that we have chosen to be of the character data type as it only takes ‘y’ or ‘n’ to indicate what someone’s choice is. The **character data set** is therefore all letters, numbers and special symbols.

The **string data type** is used to represent many characters. Examples are ‘Mary had a little lamb’, ‘Press any key to continue’ and ‘Manchester United 3, Chelsea 0’. Strings can consist of any number of character data. Again, to denote their value, they are represented between single quotes. If you were writing a program that asked students to enter their details you might use string variables called Name and Address. Depending on the computer and programming language, characters and strings can be case sensitive. This means that ‘a’ and ‘A’ are different. In this course you should assume that the programs you write are case sensitive. The **string data set** is the combination of more than one character.

The **logical data type** is used to represent decisions (see Chapter 6). There are only two values, **True** and **False**. The variable Warm might be declared as a logical data type to describe if temperature values are warm or not. The **logical data set** is True and False.

As with Sprankle, in this course it is useful to follow six simple rules for how to use data types. These are:

1. There are four data types: numeric (integer and real), character, string and logical (Sprankle uses three, but string has been ‘promoted’ in this course due to its useful nature). Note that there are many others that can be used and you will find these out as you become more experienced. The goal of this course is to introduce you to this subject and build a foundation upon which you can write complex programs.

2. The programmer chooses the data type during the programming process.

3. Data types cannot be mixed. Once a constant or a variable has been given a data type it cannot change.

4. Each data type has a data set.

5. Only numeric data types can be used in calculations resulting in a numeric result. All other numbers should be the string data type (to prevent confusion – think of this as preventing the misuse of data in your program).

6. When you introduce a variable in a program (i.e. tell the program that there is a variable of a given name for the first time) you give it the data type. For example:
Functions

Functions are small sets of instructions that perform tasks and (possibly) return results. The primary reasons to use functions is to divide the program into a set of small and manageable sub-programs. This makes the program more readable as you work on it, and as it is in ‘chunks’ it is easier to reuse commonly occurring tasks. In this course, when we refer to functions they appear as:

• Functionname(Data)

However, to use a function, a programmer will write:

• Result = Functionname(Data)

For example, to calculate a maximum of three numbers, we might have a function:

• Maximum(Num1, Num2, Num3)

To use it, we would write:

• MaximumNumber = Maximum(Num1, Num2, Num3)

where MaximumNumber, Num1, Num2, Num3 could be either integer or real data types.

Functions are typically implemented for you (i.e. they are already available in the program; the guide to the programming language will tell you which). These differ from language to language and are typically listed in a programming reference guide. The following are typically available functions.

• mathematical functions
• string functions
• conversion functions
• statistical functions
• utility functions.

Activity

See Table 2.5 in Chapter 2 of Sprankle for examples of these functions.

Operators

Operators represent simple instructions to process data. For example, the addition operator ‘+’ instructs the program to add two numeric data types together. There are several types of operators: these are mathematical, relational and logical.

Operators use operands and give a resultant. Operands are the data that the operator uses and different operators use different numbers of operands. For example, the addition operator used two operands to give one resultant. For example, 2 + 5, where 2 and 5 are the operands. The resultant of this operation would be 7. The data type of the operands and the resultant depend on the operator.
Activity

See Table 2.6 in Sprankle for examples of these.

**Mathematical operators** are the basic operations required in simple arithmetic. These include addition, subtraction, multiplication, division, etc. The data types of operands and resultants of mathematical operators are always numeric.

**Relational operators** involve a decision being made. These include greater than, less than, equal to, etc. The data type of the resultant of a relational operator is always logical. The operands can be either numeric, character or string (a computer places a numerical value on characters and strings so relational comparisons are valid). These operators are often used to make decisions and to control loops.

**Logical operators** are used to connect relational expressions. The operands and resultants are always logical data types. In this course we recognise three logical operators: **NOT**, **AND** and **OR**. These three operators have ‘truth tables’ that are used to define their behaviours. These are very important. The role of both relational and logical operators will become clearer as we consider more complex program features.

Activity

In Spankle, Chapter 2, these are given as 'Definitions of the logical operators' in Table 2.7.

Another element of operators is **operator hierarchy** (or precedence). These represent the order in which a series of operators take place. However, you can also use brackets or parentheses to reorder the hierarchy. These follow similar rules found in mathematics. So, for example,

- \( \text{Answer} = (3 \times 8) - (2 \times 4) \)

would result in 3*8 being evaluated first to get 24, then 2*4 to get 8, and then 24 – 8 to get Answer to have a value of 16. In other words, we start with the innermost parentheses and then work outward from left to right. In this example we would calculate 3*8 first to get 24 and then 2*4 to get 8. We would then calculate 24 – 8 to get 16. It is important to follow this step by step or you will finish with a different (and wrong!) result.

Activity

Sprankle describes these in Table 2.8.

**Expressions and equations**

How do we actually bring together all the concepts found in this chapter? The answer is that you can use all of these to build **expressions** and **equations**. These are the heart of a program and enable you to do all sorts of complex calculations and decision-making that make programs useful. An **expression** allows you to process data with operands and operators. An **equation** stores the resultant of an expression in a variable. For example,

- \( \text{Area} \times \text{Height} \)

is an expression that calculates volume;

- \( \text{Volume} = \text{Area} \times \text{Height} \)

is an equation that uses the expression and stores the resultant in a variable Volume.
To understand fully how to develop expressions and equations you should follow the advice given in Sprankle (see pages 25 to 32) on setting up and evaluating mathematical expressions, relational expressions and logical expressions.

**Activities**

1. Give examples of all the different data types identified in this chapter. Can a character data type and a string data type be used to represent 'A'? Can an integer data type and a real data type be used to represent the value 2?
2. Give the decision tables (as described in Sprankle, page 23) for the logical operators identified in this chapter.
3. Give examples of mathematical expressions, relational expressions and logical expressions. Can you explain and give examples of situations in which you would use each one?

**Reminder of learning outcomes**

Having completed this chapter, and the Essential reading and activities, you should be able to:

- differentiate between variables and constants
- differentiate between character, numeric and logical data types
- identify operators, operands and resultants
- identify and use functions
- identify and use operators according to their placement in the hierarchy chart
- set up and evaluate expressions and equations using variables, constants, operators and the hierarchy of operations.

**Sample examination questions**

1. Define the terms **constant** and **variable**. In this course you have encountered four different data types. Define each of these and give two examples for each one.
2. Define the term **function**. Give the generic form of function. In this course you have encountered five different types of function. Define each of these and give an example of each one.
3. There are three different logical operators. List these and define the decision table for each one. Give an example of how each operator works. Give an example of what happens when two of these operators work together.
Chapter 3: Programming concepts

Essential reading

Aim of the chapter
The aim of this chapter is to introduce you to the main concepts in programming.

Learning outcomes
On completing this chapter, and the Essential reading and activities, you should be able to:
• list the data required for a problem
• define the modules to be used in the solution of a problem
• define the input, processing and output for each module of a solution
• use algorithms to develop the instructions for each module in the solution of a problem
• describe the importance of documentation.

Introduction
The previous chapters have introduced the basic approaches to problem-solving with a computer. The question is, how do you actually program a computer? To do this we first need to know how a computer stores data and then how to work with a computer (essentially, how do we communicate with a computer?). Note that in this we generalise the problem-solving and programming tools that are discussed in Sprankle. We are very happy if you wish to use these. However, you will not be required to go into all the depth described by Sprankle in examination conditions. This chapter emphasises the fundamentals that must be captured in whatever approach you use to program. The learning outcomes identify these. Note also that learning to use a real programming language such as Visual Basic (see the books recommended at the beginning of this guide) is invaluable. You will be able to see your ideas ‘come to life’. Remember, in the examination we will be testing your knowledge on programming and not a particular programming language.

Some basics
In this course we are learning how to program at an algorithmic or abstract level. Our reason for this is that most approaches to programming immediately use a programming language. In our experience, students focus on how to use the language, not how to program. The result of this is that students often miss the fundamentals of programming and cannot easily switch between languages (say, Visual Basic and Java). At the beginning of this guide, we list two texts that you can use to begin to learn Visual Basic, an excellent start to programming in an actual language. This course and its examination, however, focus on the use
of algorithms for programming. These are sets of general instructions that use the elements of programming (and their rules) that we have introduced so far. What else do we need to know? In later chapters we will introduce various programming structures and techniques that you can use to solve a problem. In this chapter we introduce some basics. These are how the computer stores data and how we interact or communicate with a computer.

How the computer stores data
Computers store data in memory locations. This is hardware memory and is called RAM (random access memory). Computers refer to these memory locations as variables and constants. Each location can hold only one variable or constant and each time a new value is given to a variable or constant, the old value is destroyed. When a program starts there are no values in memory locations (and they must therefore be set to a default value), and when a program finishes those values disappear. In the real world, if you switch off a computer you will lose everything in RAM. We call this volatile memory.

To keep things in permanent storage, we use files. These are stored externally to RAM in external storage media such as a hard disk. When a computer is switched off, or a program finishes, information stored in these files remains. There are two types of files: these are program files and data files. When instructed to start a program, the computer finds the appropriate program file from storage, loads it into RAM and then begins to work on the program. Data files contain the data required to execute the program files. To help the transfer of information to and from storage we place the information in a temporary holding area called a buffer. We do this so that we can perform various operations on the data and to speed up input and output processes to/from a computer (otherwise we could only transfer a byte of data at a time).

Communicating with a computer
Computers need specific instructions. These instructions are governed by specific rules. Together we can call these a programming language. In this course we will learn a structured language called pseudocode that we will use to write algorithms. With a programming language, if you use an incorrect instruction, or break the programming rules, the computer will produce an error and/or the result will be wrong. We call these rules syntax and if we have given an incorrect instruction we say that we have a syntax error. If a program does not behave as expected, we say that we have a bug and the elimination of the bug is called debugging.

Organising the problem
How do we begin to program? Sprankle suggests several ‘tools’ that you can use to do this. Again, we are happy for you to use these. To begin we must first analyse the problem. Then we must (if appropriate) organise the solution to the problem into a series of modules. The input, the expected results and the outputs of the modules must then be specified. Module contents must then be detailed using algorithms. Your program must then be tested to demonstrate that it is correct. This is enough for this introduction to programming. We encourage you to learn a programming language such as Visual Basic to put your ideas into practice.
Analysing the problem

To produce a program based on a solution to a problem, the programmer must first analyse the requirements of the problem. One way to do this is to split the problem into four parts. These are:

1. the available data (the constants and variables)
2. the required results (the output)
3. the processing that is required in the problem (with any equations and expressions)
4. a list of alternative solutions.

For example, the first activity you will come across in Chapter 10 is:

A company pays their employees on a weekly basis. You have been commissioned to write a program for part of a payroll system that will calculate how much an employee will be paid for a week's work. There are two payment rates. The normal payment rate is $5 an hour and is paid for work between 08:00 and 17:00. The overtime rate is $10 an hour and is paid for work outside of 08:00 and 17:00. Payment is calculated on the number of complete hours worked. You may assume that data is entered as two figures. The first is an integer that indicates the time at which the employee started work (08, 09, 10, etc.). The second is also an integer and indicates the time at which the employee finished work (17, 18, 19, etc.).

The available data in this is the payment rates (constants) and the data that the user enters for the start and end times of each day. The required result is how much an employee will be paid for a week's work. The processing required is subtracting the end time from the start time. Then 9 must be subtracted from this to get the number of hours for which overtime must be paid. If it is less than nine then the employee worked less than 9 hours. These values must then be multiplied by 10 and 5 respectively and then be added together for each day to give the result. The alternative solutions are to perform the calculation for each day and then add the values together, or wait to get all the data and then perform the calculations (it's usually best to get all the data first).

Once you have done this you must choose a specific solution. The choice is, unfortunately, typically down to experience. Programming is often about practice. Good programmers are those with much practice.

Developing the program structure

Once you have analysed the problem, the next step is to identify the program structure. This is done by breaking down your solution into subtasks called modules. 'Connecting' the modules of your solution identifies the interaction of processing between modules. Each module should contain the instructions to accomplish one task. These are typically the input of data, the calculation of results and the output of the results. There should always be one module that controls the flow to most of the other modules. This is the control or main module.

There are different ways of choosing modules. In this course we shall use a procedural approach (as opposed to an object-oriented approach – this can be developed from the procedural approach). In procedural programming there is a direct order of processing with each instruction and module being executed in a sequential order. To develop a program using the procedural approach, we use the top-down method. This is straightforward and should be intuitive. First you start with the overall
Introduction to programming

A task (solution) is then broken down into subtasks (modules). These in turn can be broken down into smaller subtasks. This process carries on until a reasonable level of subtasks is reached (and again this is based on experience). There is one exception: this is the control module. This does not get divided as it controls the execution of the program. The next chapter discusses the subject of modules in more detail.

Specifying module input/output

Once you have identified your modules you need to add details to them. The details are the input data to a module, the processing that is required and the output from the module. The reason why you do this is that when you begin the programming process, you need to have this information for the whole program.

The next step is to produce a set of modules. You do this to simplify the programming process. Instead of writing the whole program in one go, you can write each module in turn, testing each one as you go. Before you can do this (and in a similar way to writing the input, processing and output of the whole program), you need to write what each module needs in order to perform its task (the data), what the task is (the processing) and what the module produces (the output). Once you have done this, the next task is to turn the description of processing into an algorithm.

Writing the algorithms

In this course we write the algorithms of our modules using a structured form of English called pseudocode. Since we are taking a generic approach to programming, the pseudocode that we use is intended to be as generic as possible. We do have some simple rules but the main purpose of pseudocode is to avoid the need to worry about language syntax at this stage; instructions can be written clearly in pseudocode. Once you have mastered our approach to programming, it will be easier to use a language to develop programs (it is difficult to learn how to program and worry about language syntax at the same time). We will introduce more rules as we go along. For now, our first rule is:

The instructions cannot assume anything, cannot skip steps, must be executable one step at a time and must be complete.

The algorithm for a control module looks like this:

```
Control Module
1. Instruction
2. Instruction
3. Instruction
...
End
```

where ‘instruction’ can be a simple program instruction or the name of a module. The algorithm for a module looks like:

```
ModuleName(list of parameters)
1. Instruction
2. Instruction
3. Instruction
...
Exit
```
Note that the control module ends with ‘End’ to signify the end of the program. All other modules end with ‘Exit’ to signify that the module has finished processing and is returning to the next instruction in the control module.

Other pseudocode words that we can use are the functions and expressions that were introduced in the previous chapter. Process ModuleName to indicate that we are now ready to start the execution of a module.

Other words will be introduced in the following chapters. Note that Sprankle uses flowcharts as well as algorithms written in pseudocode to specify programs. In this course we are happy for you just to use pseudocode.

**Documenting your solution**

In programs it is very important to place comments in the algorithm. The purpose of comments is to explain what that particular part of the program is doing, or what a particular variable is used for. We call this documenting your program. All too frequently programmers do this task very poorly. We would recommend that you do not! Think about coming back to some code that you wrote a week ago. Could you remember what every part of the code does? Now think about leaving the code for a month!

In this course, you can use the symbol */ to indicate the start of comments and /* to indicate the end of comments. For example,

```
ModuleName(list of parameters)
/* This module takes the list of parameters and performs the following task. /*
1. Instruction
2. Instruction
3. Instruction
...
Exit
```

**Testing your solution**

When your solution is complete, it is important to test it to make sure there are no errors and it meets the requirements written at the beginning of the programming process (there are many programs that ‘drift’ away from their original requirements). To test a solution, the programmer selects test data, a set of values for the input data of the program, and for each one works methodically through the instructions of the program. Care must be taken to select the right data to test it with, as not all input data will test all the functionality of your program. If any bugs (errors) are identified in your program, then you should correct them. For example, if your program requires that a user enter integers, then what happens if the user enters a character? Your program should be able to ignore the wrong input and ask again for the correct input (an integer). Similarly, when you are dealing with loops or conditionals, you should be able test if all these behave as you would expect. Typical errors occur when a wrong number of loops are counted (for example, counting from 0 or 1) or conditionals are badly organised.
Testing your solution in the examination

In the examination for this course you will be asked to write programs. The problem is, how do you demonstrate (or test) that they work in an examination?

For every programming problem you will be asked to write the program and then a short essay on how the program works. This is how you ‘test’ your solution under examination conditions.

So, by starting with the requirements for the program you would discuss the parts of the algorithm that implement the solution to them. The essay explanation is therefore one along the lines of ‘the program begins by initialising data and then asks for the number of cars to be entered (line 10)’, etc. It is useful in the examination therefore to put line numbers on each line of the algorithm that you write out so that you can identify them specifically.

Activities

There aren’t any specific activities as the content of this chapter forms part of the set of concepts that you will need for programming. The activities for this chapter are brought together with the other programming concepts to form a complete activity chapter at the end of this guide (Chapter 10).

Reminder of learning outcomes

Having completed this chapter, and the Essential reading and activities, you should be able to:

• list the data required for a problem
• define the modules to be used in the solution of a problem
• define the input, processing and output for each module of a solution
• use algorithms to develop the instructions for each module in the solution of a problem
• describe the importance of internal and external documentation.

Sample examination question

1. Discuss how you should organise a solution to a problem. You should consider four major steps in this process. Support your answer with a worked example.
Chapter 4: Program structure

Essential reading

Aim of the chapter
The aim of this chapter is to introduce the structure of a program and to consider concepts such as cohesion and coupling.

Learning outcomes
On completing this chapter, and the Essential reading and activities, you should be able to:
• explain the need for structured (modular) programming
• explain how to design modules and functions, in terms of cohesion and coupling
• explain the difference between local and global variables
• explain the use of parameters.

Introduction
So far we have introduced techniques to organise solutions to problems as well as techniques to implement them in a computer. The main theme so far in this course has been to develop algorithms from which programs can be developed. To carry on with this theme, this chapter discusses program structure. Specifically:
• the purpose of modules
• four logic structures (sequential, decision, loops and case)
• readability.

Modules and their functions
What is a module? A module can be thought of as a chapter of a book. It has a start point, an end point and carries out some function. Programs essentially consist of many modules, each with their own particular purpose. Apart from making the design and implementation of a program easier (than just trying to write a whole program in one go), modules also make testing a lot easier as you can test each module as you go along (rather than the whole program). Modules can be used by other modules.

Activity
Now read Sprankle page 65, who suggests five rules that are very useful when designing modules.
1. A module is almost a single program in its own right. It has a start point and an end point where processing begins and ends.
2. A module has a single purpose or function.
3. A module should be short enough to be quickly understood and changed.
The length of a module is ultimately down to its purpose/function and the number of instructions needed to do that job.

A module helps to control the order of processing (by calling the module’s name from the program’s control module - see below).

Different types of module can be identified by their purpose. For example:

- The **control module** controls the overall flow of the program as well as all other modules in a program.

- The **initialisation module** is at the beginning of the program and is executed only once. It performs tasks such as opening files and setting the initial values of variables used in the program.

- The **process module(s)** are one or more modules that form the main part of the program. Each can be executed many times during a program. There are several types:
  - calculation modules for arithmetic and mathematical tasks
  - print modules to print output from the program to an output device such as a screen or printer
  - read/data validation modules to read data from an input device, such as a keyboard, into the computer’s memory and to check it is in the correct format, range, etc.

- The **wrapup module(s)** that end the execution of the program. These are executed only once and perform tasks such as closing files, printing the final results, etc.

**Cohesion and coupling**

Cohesion and coupling are two very important concepts in modules. When writing a program, what exactly goes into a module? Modules should be functionally separate and should perform a single task. Each module should also be connected to another module (usually by the data that they use). The first of these connection approaches is **cohesion** and the second **coupling**. Cohesion lets a programmer develop a module as a unit of a larger program, sometimes called a ‘black box’ as the contents of the module are hidden from other programmers. Coupling demands that this module (or black box) be easily connected to other modules in the program. To do this, the developer of the module makes it clear which variables the module uses. For example, if there is a module DataEntry in a program that is designed to calculate the average number of cars passing a point in a day on an hourly basis, then the module might use the variables CarNumberHour1, CarNumberHour2, etc. The next module, AverageCarCalculator, might be the one to perform the calculation. This module might use variables that the first one used. We can say that these modules are coupled together and that the instructions used within them to perform their given tasks give the modules their cohesion. Additionally, we can say that the variables CarNumberHour1, CarNumberHour2, etc., are passed from the module DataEntry to the module AverageCarCalculator. We consider three approaches to coupling modules: global variables, parameters and return values.

**Activity**

See Sprankle Figure 4.5 for an illustration of coupling and cohesion.
Local and global variables

The difference between local and global variables is something termed scope. Local variables are defined and used within a module and global variables are defined at the highest level of the program and are used by any module. The scope of a local variable is therefore the module and the scope of a global variable is therefore the program. Great care should be used when using global variables as any module can change a global variable in any way it pleases (these are called side-effects). Global and local variables should never be called the same name as this can lead to confusion. Think of the case where a global variable called Amount and a local variable called Amount are used in a module. If, in the module, the statement 'Print Amount' is used, which Amount is referred to, the global one or the local one? The answer is that one can never be sure. The solution is always to name them differently.

Parameters

These are very important in programming as they facilitate coupling between modules. They are also quite difficult to grasp. As has been mentioned, a module passes a variable to another module. These are local variables and are termed parameters. To use them, we place them after the module name and in brackets. For example:

- CalculateAverage(X,Y,Z)

is a module that has three parameters X, Y and Z.

Some terminology that we use with parameters:

- In a program, modules are declared (written – we say ‘declared’ in the sense of introducing the module to the program so that it knows what we mean) either before or after the control module. Either is fine so long as you do not do both (programming languages usually declare a module before the control module). When a module is used by another module we say that it is called by the module that uses it (the calling module).
- The list of parameters with a module declaration is called the formal parameter list.
- The list of parameters given to a module when it is called is the actual parameter list.
- The names of the variables in a module’s formal parameter list do not need to match the module’s actual parameter list. However, the number of variables must match.
- We can say that the actual parameters are passed to a module when it is called.

There are two approaches to passing parameters to a module: call-by-value and call-by-reference.

- Call-by-value occurs when just the value of a variable is passed (i.e. a copy of the variable). Any changes made to the variable only exist within the scope of the module being called (i.e. it is a local variable). When the module finishes execution the value ceases to exist and the program carries on execution with the original value of the variable.
- Call-by-reference passes the actual variable to the module. We denote this by adding * to the name of the variable. Any changes made to the variable within the scope of the module remain when the module finishes its execution. The scope is shared between the
calling module and the called module. In reality, the actual memory address of the variable is passed to the called module, not the value of the variable. However, in this course it is enough to know that any changes to a variable made by a called module continue to exist in the calling module when the called module is finished.

The variables called by value or called by reference in a module's actual parameter list must match the variables called by value or called by reference in the module's formal parameter list.

Call-by-value and call-by-reference help a programmer control side-effects in a program by allowing him/her to say when a module can modify a variable and when (outside the scope of the module) it cannot.

We use the term Process to call a module.

We use the term Exit to end a module and to return back to the calling module.

Modules can call modules that can call modules, etc.

Parameters couple modules together.

Consider the following example of a program that calculates the average number of cars passing a point over three hours. Data is entered for each of the three hours.

<table>
<thead>
<tr>
<th>Process</th>
<th>ReadData(*NoHour1, *NoHour2, *NoHour3)</th>
<th>Process CalculateAverage(NoHour1, NoHour2, NoHour3, *Average)</th>
<th>Process PrintAverage(Average)</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadData(*Hr1, *Hr2, *Hr3)</td>
<td>Print 'Please enter the number of cars in the first hour'</td>
<td>Enter Hr1</td>
<td>Print 'Please enter the number of cars in the second hour'</td>
<td>Enter Hr2</td>
</tr>
<tr>
<td>CalculateAverage(X, Y, Z, *Result)</td>
<td>Total = X + Y + Z</td>
<td>Result = Total / 3</td>
<td>Exit</td>
<td></td>
</tr>
<tr>
<td>PrintAverage(Av)</td>
<td>Print 'The average number of cars per hour is:'</td>
<td>Print Av</td>
<td>Exit</td>
<td></td>
</tr>
</tbody>
</table>

The first module in the program is the control module. This begins by calling ReadData. The formal parameter list of ReadData is *Hr1, *Hr2 and *Hr3. The actual parameter list is *NoHour1, *NoHour2, *NoHour3. Both lists match and all the variables are call-by-reference. When the control module calls ReadData by the statement Process ReadData(* NoHour1, *NoHour2, *NoHour3), the calling module passes the parameters NoHour1, NoHour2 and NoHour3 to the ReadData module.

ReadData then asks for the value of each to be entered and then returns these values to the calling module (the control module) when done (when it encounters Exit).

The control module then executes the statement Process CalculateAverage(NoHour1, NoHour2, NoHour3, *Average).

This matches the formal parameter list of CalculateAverage(X, Y, Z, *Result) in that the first three variables are call-by-value and the last is call-by-reference. Why the difference? The answer is that there is no reason for the CalculateAverage module to change the value of X, Y and Z. By making these call-by-value, the module CalculateAverage cannot change them at all. As with any module, the calling module passes the variables to CalculateAverage, the module adds the three numbers together and stores them in the local variable Total. The value of Result is then calculated and then returned when the module executes Exit.

PrintAverage is executed in a similar manner.
Activities

1. Write a program ‘StopWatch’ that takes two inputs (start time and end time) as hours, minutes and seconds and outputs the elapsed time as hours, minutes and seconds. The program must consist of three modules: ReadData, CalculateElapsedTime and PrintElapsedTime. What are the parameters for each module? Which are call-by-value and which are call-by-reference? Can any of your modules be called twice? (Clue: ReadData).

2. You have been given a program to test that, given two dates, calculates and displays the number of days between the dates. Produce a test plan that could be used to test the program for the ‘Y2K’ bug.
   
   (Hint: don’t forget about leap years. A leap year occurs if the year is a multiple of 4. More accurately, take into account the fact that years whose numbers are multiples of 100, but not 400, are not leap years.)

3. For both of the above programs, write short explanations of how they work.
   
   Remember, to get started you should apply your knowledge from the first three chapters. Analyse the problem first.

Reminder of learning outcomes

Having completed this chapter, and the Essential reading and activities, you should be able to:

• explain the need for structured (modular) programming
• explain how to design modules and functions, in terms of cohesion and coupling
• explain the difference between local and global variables
• explain the use of parameters.

Sample examination questions

1. Define the term module. List five rules that could be used in designing a module. Using appropriate examples, define and explain four different types of module.

2. Briefly discuss the differences between cohesion and coupling. Discuss how three different approaches to coupling modules can be used.

3. Define the terms call-by-reference and call-by-value. Using appropriate examples discuss when it is appropriate to use each of these in a program.