

Heinemann

Software Design and Development



Preliminary Course

Allan Fowler

Heinemann

Heinemann
A division of Reed International Books Australia Pty Ltd
22 Salmon Street, Port Melbourne, Victoria 3207
World Wide Web hi.com.au
Email info@hi.com.au

Offices in Sydney, Brisbane, Adelaide and Perth.
Associated companies, branches and representatives throughout the world.

© Allan Fowler 2000
First published 2000
2003 2002 2001 2000
10 9 8 7 6 5 4 3 2 1

Copying for educational purposes

The Australian *Copyright Act* 1968 (the Act) allows a maximum of one chapter or 10% of this book, whichever is the greater, to be copied by any educational institution for its educational purposes provided that that educational institution (or the body that administers it) has given a remuneration notice to Copyright Agency Limited (CAL) under the Act.

For details of the CAL licence for educational institutions contact CAL, Level 19, 157 Liverpool Street, Sydney, NSW, 2000, tel (02) 9394 7600, fax (02) 9394 7601, email info@copyright.com.au.

Copying for other purposes

Except as permitted under the Act, for example any fair dealing for the purposes of study, research, criticism or review, no part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without prior written permission. All enquiries should be made to the publisher at the address above.

Publisher: Rosie Adams
Editor: Felicity Shea
Designer: Gerry Theoharis
Design development: Giulia De Vincentis
Cover designer: Relish Design
Illustrations: Guy Holt
Photograph researcher: Janet Pheasant

Typeset in 10.5/12.5 Berling by Idczak Enterprises
Film supplied by Type Scan, Adelaide
Printed in Australia by the australian book connection

National Library of Australia
cataloguing-in-publication data:

Fowler, Allan, 1949–
Heinemann software design and development:
preliminary course.
Includes index.
ISBN 0 86462 438 7.
1. Computer science. 2. Computer software—Development.
I. Title

005.3

Disclaimer

All the Internet addresses (URLs) given in this book were valid at the time of printing. However, due to the dynamic nature of the Internet, some addresses may have changed, or sites may have ceased to exist since publication. While the authors and publisher regret any inconvenience this may cause readers, no responsibility for any such changes can be accepted by either the authors or the publisher.

Contents

Introduction	v
<i>Heinemann Software Design and Development</i> and the Preliminary Course Outcomes	vi
1 Social and ethical issues	1
Ergonomics	4
Intellectual property	15
Inclusivity	22
Review exercises	25
Chapter summary	26
2 Hardware and software	27
Hardware	30
Software	48
The relationship between hardware and software	57
Review exercises	63
Chapter summary	64
3 Software development approaches	67
Introduction	70
The structured approach to software solutions	70
The prototyping approach to software solutions	78
Rapid application development (RAD)	81
End-user development	83
Review exercises	85
Chapter summary	86
4 Defining the software solutions	87
Defining the problem	90
Abstraction/refinement	95
Data representation	98
Data types	103
Structured algorithms	112
Checking algorithms	140
Review exercises	147
Chapter summary	149

5	Building software solutions	151
	Coding in an approved programming language	154
	Error-correction techniques	174
	Libraries of code	188
	User interface development	193
	Documentation	207
	Review exercises	217
	Chapter summary	218
6	Checking the software solution	219
	Test data	222
	Evaluation of design	236
	Evaluation of implemented solution	239
	Review exercises	242
	Chapter summary	244
7	Modifying software solutions	245
	Reasons for maintenance coding	248
	Social and ethical implications	255
	Features in source code that improve its maintainability	256
	Interpretation	260
	Documentation	267
	Review exercises	269
	Chapter summary	270
8	Developing software solutions	271
	Implementing projects	274
	Project management techniques	285
	Project documentation	290
	Social and ethical issues related to project work	292
	Review exercises	294
	Chapter summary	295
	Appendix 1: Sample examination paper	297
	Appendix 2: Pascal syntax structure diagrams	304
	Glossary	309
	Index	312

Introduction

This book has been written to support students and teachers in implementing the NSW Software Design and Development Preliminary Course. The course has been introduced by the Board of Studies to give students who are interested in the field of software design and development an opportunity to study the subject and create appropriate software solutions.

The development of software involves careful planning, clear documentation and an appreciation of the effects that the product may have on members of society. To this end, the book covers aspects of development, including analysing the problem, planning a solution, creating and testing the solution, documenting the solution as well as the associated social and ethical considerations.

Since different problems will need different approaches, a number of methods of software development are examined. These vary from the formal, structured approach through to the informal approaches such as end-user development. The nature of the problem to be solved will also dictate which computer language is most suitable to develop the solution. To cater for this need, examples have been drawn from a wide range of languages.

A short biography of a significant contributor to computing introduces each chapter. This feature has been included to help the student understand that computing is a very human activity. For interested students and teachers, further information about these people and their contributions can be found from the Internet.

Each chapter begins with a statement of the chapter outcomes, student knowledge and student experiences. This allows both the teacher and the student to ensure that the syllabus content has been met. The chapters end with a summary, a set of review exercises and a team project.

Teamwork is an important part of software development as many projects are too large and complex for a single individual to tackle. The team projects have been designed to enable students to experience working members of a team. Each of the team projects covers one or more of the aspects of software design and development that have been covered in the chapter. Thus, the projects will also give students the opportunity to review and discuss the material presented in the chapter.

Additional support for teachers and students is available on the Internet site: <http://www.hi.com.au/softwaredesign>.

About the author

Allan Fowler has taught Computing Studies to senior students at Gosford High School since the beginning of 1978. The first courses were school-based programming courses as Other Approved Studies.

Allan then taught the 2 Unit course, first examined in 1990, and was a participant in the support-material writing workshop held in Port Macquarie that year. He has contributed computing related articles to professional journals in both computing and mathematics and has given many workshops and presentations to teachers, parents and students in computing.

Allan wrote the very successful *Heinemann Senior Computing Studies 3 Unit (Additional) HSC* and contributed to the *Heinemann Senior Computing Studies 2/3 Unit Common HSC Course*. He wrote the solution manuals for both texts.

Acknowledgments

I am indebted to my wife Kaye and my children Katharine and Stephanie for their patience and understanding throughout the writing of this text.

Comments and suggestions made by the reviewers (Anthony Connolly, Dieter Opfer, Greg Tardiani, Rick Walker, Cathie Webber and Chris Wiecek) have significantly assisted in the task of producing this text. I would particularly like to thank Dieter Opfer for his contribution to the text and Glenda Horner for her criticisms and suggestions for improvements to the early drafts.

Finally I would like to thank Rosie Adams and the team at Heinemann for the very professional help and encouragement that they have willingly given.

Allan Fowler

February 2000

Heinemann Software Design and Development and the Preliminary Course Outcomes

The following grid shows how the chapters in *Heinemann Software Design And Development: Preliminary Course* link with the Preliminary Course Outcomes.

Preliminary Course Outcomes	Chapter(s)
P 11 Describes the functions of hardware and software	2
P .2 Describes and uses appropriate data types	4, 5, 7, 8
P .3 Describes the interactions between the elements of a computer system	2, 4, 5, 8
P .1 Describes developments in the levels of programming languages	2
P .2 Explains the effects of historical developments on current practices	1, 2, 3, 4, 7
P .1 Identifies the issues relating to the use of software solutions	1, 2, 3, 4, 5, 6, 7, 8
P .1 Analyses a given problem in order to generate a computer-based solution	3, 7, 8
P .2 Investigates a structured approach in the design and implementation of a software solution	3, 4, 5, 6, 7, 8
P .3 Uses a variety of development approaches to generate software solutions and distinguishes between these approaches	3, 4, 5, 7, 8
P .1 Uses and justifies the need for appropriate project management techniques	5, 6, 7, 8
P .2 Uses and develops documentation to communicate software solutions to others	4, 5, 6, 7, 8
P .1 Describes the role of personnel involved in software development	1, 2, 3, 5, 6, 7, 8
P .2 Communicates with appropriate personnel throughout the software development process	5, 6, 7, 8
P .3 Designs and constructs software solutions with appropriate interfaces	5, 6, 7, 8

chapter 1

→ *Social and ethical issues*

Outcomes

- explains the effects of historical developments on current practices (P 2.2)
- identifies the issues relating to the use of software solutions (P 3.1)
- describes the role of personnel involved in software development (P 6.1)

Students learn about:

Ergonomics

- effects of prolonged use of software, including RSI and injuries created by overuse
- procedures to prevent and minimise injuries
- ergonomically designed and placed equipment
- ergonomic issues regarding software design:
 - acceptable response time in software
 - ‘user friendly’ software, including ease of use, appropriate messages to the user and consistency of the user interface

Intellectual property

- software licence agreements, including:
 - licence terminology
 - legal aspects
 - use of software covered by a licence agreement
- origin of software design ideas
 - evolution of existing concepts, including GUI interface and search engines
 - new and exciting approaches, including Visicalc, web browsers and presentation software

- events that have led to the need for software licence agreements, including:
 - ease of reproduction and copy
 - collaborative development history—the current open environment of the Internet
- sources of code and conditions that apply, including:
 - the Internet
 - books and magazines
 - shareware

Inclusivity

- the need for software design and development to be inclusive
 - cultural perspectives
 - economic perspectives
 - social perspectives
 - gender perspectives
 - disability perspectives
- the general strengths brought to the field of software design and development, including:
 - communication skills
 - ability to work in teams
 - creativity
 - design skills
 - problem-solving skills
 - attention to detail

Students learn to:

- identify sound ergonomic practices when using computers
- assess the ergonomic needs of the user when developing software
- debate the issues relating to intellectual property
- use software in an ethically and legally correct manner
- evaluate existing software interfaces in terms of their inclusivity

Personal Profile—Countess of Lovelace (1815–1852)

Augusta Ada Byron was born on 10 December 1815 in London. She was the daughter of Lord Byron, the poet, and Anna Isabella Millbanke. When Ada was only five weeks old her mother and father separated. Ada was raised solely by her mother, never again being seen by her father.

Very little is known about her youth, but her mathematical aptitude was certainly recognised and encouraged by her mother. Ada was a typical young woman who enjoyed many of the entertainments available to a person of her class. Ada was introduced to Mary Sommerville, a mathematician of high standing, when she was seventeen. She deeply admired Mrs Sommerville who encouraged Ada further in her mathematical pursuits.

As well as acting as a mathematical mentor to Ada, Mary Sommerville also introduced her to William Lord King whom she married when she was nineteen. William, who soon became the Earl of Lovelace, encouraged and supported Ada in her work. Although Ada was intellectually superior to her husband, it caused no friction in their marriage as William took pride in her achievements and accepted her many talents.

Settling into the country, Ada had two sons and a daughter. As was a common practice among the upper class, the children were brought up by others, leaving Ada time to pursue her interests in mathematics. However, illness prevented her from developing her ideas fully.

In 1836 she started a correspondence with Charles Babbage (see Chapter 2) which was to last until her death. In 1842 she translated a paper on Babbage's analytical engine written by an Italian mathematician, L. F. Menebrea, adding her own notes which extended the ideas considerably. In Ada's final letter about the paper, she clearly shows that she understood the limitations of the machine when she wrote, 'The analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform. It can follow analysis, but it has no power of anticipating any analytical revelations or truths. Its province is to assist us in making available what we are already acquainted with.'

Ada's life was tragically cut short by cancer in 1852 when she was just 36. She is buried beside her father in Hucknall Torkyard, Nottinghamshire. It was only a century later, with the advent of computers, that the real worth of her work was appreciated.



Ergonomics

The term **ergonomics** refers to study of the relationship between people and their work environment. It is the process of designing or arranging workplaces, products and systems so that they fit the people who use them. The way a computer is used, and the work environment or surrounding influences, can have an effect on the body. The work environment includes the operator's desk, the type of chair, where the computer equipment is placed, the technique used at the keyboard, the work routine, the software being used, and the room lighting, noise and temperature. The relationship with all these factors affects health and efficiency. If computers are being used in the correct way, the working environment is safe and the user will be working at maximum efficiency. Conversely, incorrect use of computers can cause health problems such as eyestrain, headaches, back-aches, fatigue, muscle pain and **repetitive strain injury (RSI)**.

Effects of prolonged use of software

Repetitive strain injury

Repetitive strain injuries are caused by a number of different factors, including:

- the number of movements made in performing a task
- the load or forces required to perform the task
- the amount of static muscle work required to perform the task (i.e. the muscle work needed to keep the body in the position to perform the task)
- stress on the body and/or various joints
- the physical capacity of the individual
- the time needed to perform a task.

Two common forms of RSI are carpal tunnel syndrome and tenosynovitis:

- Carpal tunnel syndrome (CTS) is produced by repeating the same small movements many times. Typical symptoms are numbness or burning in the fingers or wrist. If not addressed early on, the injury can cause permanent damage.
- Tenosynovitis is an inflammation of the tendon sheath. It occurs when the repetitive activity exceeds the tendon sheath's ability to lubricate the tendon. The friction resulting from the excessive repetitive activity causes pain and swelling in the tendons.

To reduce these health problems, there have been numerous reports and standards that deal with ergonomics and personal computers. These standards can contain conflicting results as they are based on different anthropometric data (body size and shape). The Australian Standard AS 3590.2 and the Worksafe Australia checklist will be referred to in the following sections. Everybody who uses a computer should endeavour to meet these standards.

Procedures to prevent and minimise injuries

This section examines environmental factors such as lighting, indoor climate and noise. These factors affect the way a computer is used.

Lighting

Incorrect lighting can cause eyestrain, resulting in a burning of the eye, double vision, headaches and reduced visual powers. Lighting needs to be uniform, and

bright enough for all text to be read easily on the screen, keyboard and paper. Harsh lighting should be avoided and the level of brightness in the room should be similar to the brightness of the screen. Adjust the contrast and brightness on the computer screen to a comfortable level. Clean the computer screen and other surfaces regularly. Light fixtures should be positioned directly above the computer to reduce the reflected glare (see Figure 1.1). All parts of the work environment should have non-reflective surfaces to minimise glare. Reduce or eliminate glare by using window shades, diffusers on overhead lighting and anti-glare filters for computers.



Figure 1.1 Reflected glare affects reading.

Indoor climate

If the climate of a room is uncomfortable, it can cause weariness, sleepiness, loss of performance and increased errors. The maintenance of a comfortable climate indoors is essential for well-being and maximum efficiency. Air temperature, air humidity and air movements affect the indoor climate. The temperature range at which a person feels comfortable varies and depends on the amount of clothing being worn and the amount of physical effort. For a clothed and resting person, the temperature should range between 20°C and 23°C. If the relative humidity of the air is between 30 per cent and 70 per cent it will not create any discomfort. Air movements such as draughts are unpleasant if they exceed 0.2 metres per second and are at the level of the head and knees.

Computer equipment can generate heat and raise the temperature of the work environment, so air-conditioning is often required. Furthermore, good air circulation is essential; otherwise the stuffiness can make people drowsy and reduce efficiency.

Noise

Excessive noise in the work environment can be a significant distraction. Noise levels should not exceed 55 decibels. Levels above this make communication with others difficult and can affect concentration. Protection from noise can be obtained by:

- planning the subdivisions of the building (The noise level will decrease with increasing distance from the source; hence offices should be sited as far away from noisy areas as possible.)
- sound insulating a room by covering the walls and ceilings with sound-absorbing materials
- enclosing the source of the noise with sound-absorbing materials
- using headphones, ear plugs and soft music.

Most pieces of computer equipment do not produce excessive noise and are seldom the source of complaints.

Work routine

The work routine is the way the job is done, and this directly affects performance. The design of the job, the workload and the pressures of the job are important factors in work routine.

Job design

The job should be designed to suit the outcomes to be achieved, the methods to be followed and the required skills. If jobs are carefully designed they can be both satisfying and productive. The following issues need to be considered when designing a job:

- What is the extent of the job?
- Does the job involve a variety of tasks?
- What opportunities exist for social contacts?
- What satisfaction can be gained from the job?
- How can boredom be reduced?

Jobs that are highly repetitive, such as data entry using a computer, require rest pauses and a variety of tasks that involve movement away from the computer.

Workload

The body needs a balance between work and rest. It is important that workload matches and does not exceed an operator's needs and capabilities. Rest pauses are taken when switching work, dealing with changes in the nature of the work (waiting for a printer), at prescribed times (lunch), and voluntarily. These rest pauses reduce the effects of fatigue and improve performance and efficiency. When the workload does involve extended periods of computer use, a rest should be taken every hour, away from the computer. Special exercises can be done during these rest pauses to prevent RSI.

Work pressure

Work pressure causes stress and has an effect on health and efficiency. It can create feelings of anxiety, tension, depression, anger, fatigue, lack of vigour and confusion. The factors affecting the degree of stress include workload, job satisfaction, job design, social support and job security. Reduce stress by:

- planning ahead and setting realistic expectations for what can be accomplished during the workday



Figure 1.2 An organised work environment.

- organising the workload to help even out busy and slow times
- varying tasks to make the day more interesting, for example, delivering a message in person instead of phoning
- arranging the desk and work environment so that you have space to work and are not distracted by useless information (see Figure 1.2).

Most people who use a computer at work do not find it stressful. There are some problems involved in learning a new task with the computer, but once the task has been mastered, people enjoy the interaction. However, computer breakdowns or software freezes can create a stressful situation while users are waiting for the computer to be repaired, as the work is building up and the next day's workload is increasing.

Touch-typing

Touch-typing is the ability to locate keys by finger touch rather than by looking at the keys. Touch-typing gives the advantage of being able to keep your eyes on the document. Two-finger typists have to look from the document they are typing to the keyboard and back again, often losing their place and hence losing time. There is also a greater risk of making spelling or other mistakes. Two-finger typists have to memorise the words to be typed and then focus on the keyboard. If they have not memorised the words correctly, mistakes will occur.

Exercise 1.1

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

The relationship between people and the _____ elements studied in _____ . The correct use of computers can help the user to be _____ and _____. Incorrect use may cause health problems such as _____, _____, _____, _____ and _____. Elements of the work environment are _____, _____ of equipment, keyboard _____, work _____, _____ being used and the _____ routine.

- 2 What factors of an office environment can affect the health of a worker? Describe the way in which each of these factors affects the worker's health. You can present your answer as a word-processed document called OFFICEWK.
- 3 Perform a search on the Internet to discover the latest standards in ergonomic design. Report briefly on these standards, acknowledging the sources of your information. This document should be word processed and saved with the filename STANDARDS.
- 4 Report on the ergonomics of your school computer rooms. What improvements would you make if you were allowed an unlimited budget?
- 5 Design an ergonomically sound home office, to be set up in a 3.6 m by 3.6 m room which has one window 3 m wide opposite a door that is 0.85 m wide. Your design should be presented as a scale drawing accompanied by a report which gives details of the placement of items together with the reasons for this placement. You may use appropriate computer technology to help you with this task.
- 6 Explain how the tasks that workers have to perform may affect their health.

Ergonomically designed and placed furniture

In the past there was no thought given to the height of the operator's desk or chair. Whether a person was 140 cm or 190 cm tall, the same piece of furniture would be used. We now know that furniture needs to be adjusted to suit each person's body, otherwise problems may develop in the back, neck, shoulders, arms and legs. The desk and chair need to be positioned so that these body parts are used effectively without strain and undue fatigue.

Desk

The desk design is related to work efficiency. There needs to be sufficient space for the computer equipment and other documentation. The following recommendations will reduce health problems.

- The height of the desk should be about 200 mm above seat level. In this way the forearms will be parallel to the floor when typing and muscle strain will be reduced. The recommended height is between 660 mm and 680 mm for a fixed desk, and between 610 mm and 720 mm for an adjustable desk.
- The depth of the desk should be about 900 mm to allow the keyboard to be repositioned and to provide room for other computer equipment. There should be at least 50 mm for palms and wrists between the front edge of the desk and the keyboard. This will reduce the strain on forearms when typing.
- Legroom should be sufficient at both knee and foot level to permit changes in position.
- An adjustable section for the keyboard is not considered essential for an ergonomic desk. If the desk is the correct height for the operator there is no need for the height of the keyboard to be adjusted.

Chair

The chair is an important ergonomic factor as it affects your posture. Poor posture and an uncomfortable chair can result in pains in the neck, shoulders, back, wrists, joints, arms and legs. It is important that chairs be designed to provide maximum comfort and minimum restriction. The following recommendations will reduce health problems:

- The correct seat height depends on the height of the operator. Most ergonomic chairs have an adjustable seat height with the recommended range of 370–520 mm from the floor. This allows a clearance of 200 mm between the seat and the desk. If compromising with a fixed height chair it should be in the range of 410–30 mm from the floor.
- Seat depth is measured from the front edge of the seat to the backrest. The recommended range is 380–420 mm. If the seat depth is too large, small people cannot sit back far enough to get the benefit of the backrest.
- The backrest should be adjustable to fit snugly into the small of the back. This supports the back and minimises the tendency to slump in posture. The recommended range for the backrest above the seat height is 170–250 mm.
- The seat should be flat, well padded, and slanted slightly backwards. This will force the operator to lean back against the backrest and maintain good posture (see Figure 1.3). Chairs with an adjustable slope should be set with a maximum backward slope of 5 degrees.
- Chairs should have a swivel base and castors to allow the operator to move without any unnecessary twisting that could damage the back.

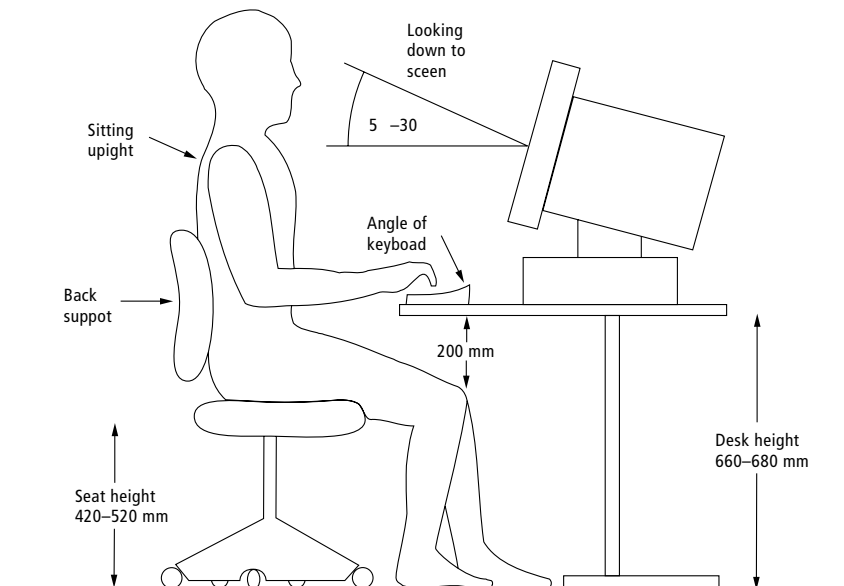


Figure 1.3 The correct arrangement of furniture and computer equipment.

The desk and chair can reduce muscle pain by helping you to maintain good posture. The feet should rest flat on the floor, thighs should be parallel to the floor, and the forearms should make a right angle at the elbow and rest on the desk. The small of the back should be supported, the shoulders relaxed (not slumped), and there should be no pressure under your thighs. It is important to alternate between different postures on a regular basis.

Ergonomically placed equipment

Clearly, ergonomically designed furniture is important in setting up a safe working environment, but so is the placement of the computer equipment. In recent years system boxes bought as a tower or mini-tower have become very popular. They allow the user to position the system box away from the screen and keyboard, and thus provide increased desk space. Ergonomic recommendations exist for the screen, keyboard and mouse.

Screen

Looking at a screen all day can lead to eyestrain but will not cause permanent eye damage. Symptoms of eyestrain include sore, tired, itchy, dry or burning eyes; brief difficulty refocusing the eyes; blurred or double vision; headaches; and increased sensitivity to light. Taking regular breaks from the screen and blinking to keep the eyes lubricated can prevent eyestrain. In addition, the following recommendations for the screen will reduce eye soreness:

- The screen should be about an arm's length away, with the operator looking down at the screen, not up. The screen should be between 15 and 30 degrees below the eye level (see Figure 1.3).
- The image should be clear, sharp, and steady.
- The surface of the screen should also be adjusted so that it is at right angles to the line of sight. For this reason the screen should be tilted slightly upwards.
- Minor adjustments of angle, brightness and contrast should be possible, to cater for individual differences.

- A filter can be placed in front of the screen to reduce the amount of glare from natural or artificial light.

The effects of electromagnetic radiation (EMR) generated from screens are not known for certain, but there are some statistics that link computers to miscarriages, birth defects, skin problems and cancer. These statistics have related to people using computers for at least eight hours a day. Scientists are investigating a theory that low-frequency EMR vibrates molecules like a base speaker resounds through the body. In addition, they have been able to measure an increase in the calcium concentration in certain cells when exposed to EMR. Though it is not certain, there seems to be enough evidence to cause concern. For this reason, international standards have been set which aim to reduce EMR emissions. NEC™ is the first computer manufacturer to have produced an ergonomically designed monitor that meets these requirements.

Keyboard

When using the keyboard, an unnatural strain is being placed on muscles and this can cause RSI. The following recommendations will reduce RSI:

- The keyboard should be placed on a stable desk and positioned so that the forearms are parallel to the floor.
- The angle of the keyboard relative to the desk should be between 5 and 18 degrees (see Figure 1.4). It is desirable for this angle to be adjustable.
- The keyboard should be detachable and separate from the screen.
- The keyboard may be stepped, sloped or concave.
- The keys should require a minimum of pressure and the operator should be aware of a completed keystroke by touch and sound.
- The keyboard should contain a numeric keypad if large amounts of data containing numbers are to be entered.

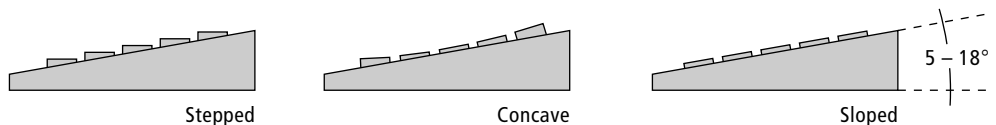


Figure 1.4 Types of keyboard that are acceptable.

Some companies have introduced ergonomic keyboards that help you maintain good posture when typing. Microsoft's Natural Keyboard (see Figure 1.5) has a wedge shape that keeps the shoulders straighter and the arms in a more relaxed



Figure 1.5 Microsoft's Natural Keyboard is an example of an ergonomically designed keyboard.

position. It has a built-in palm rest, and a wrist-levelling device that allows the keyboard height to be adjusted to help maintain a straight wrist. The PerfectTouch 101 keyboard is similar to Microsoft's except that the split angle is adjustable. This lets you set the angle for optimal comfort.

Mouse

The mouse is often forgotten when designing an ergonomic workstation. However, it is important to consider the following features when purchasing a mouse:

- The mouse should fit the hand and be easily moved.
- The button should require a minimum of pressure.
- The sensitivity of the mouse should be easily adjusted to suit the operator.



Figure 1.6 The first mouse, designed by Douglas C. Englehart in 1964.

Exercise 1.2

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Furniture needs to be _____ to suit each person's _____. Problems may develop in a person's _____, _____, _____, _____ and _____ if such furniture is not used. Furniture must be positioned so that these body parts are used _____ without strain or _____.

- 2 Use a drawing program to draw a picture of an adjustable desk and an adjustable chair. On the drawing, mark in the recommended distances and heights. Save your file as DESKSEAT.
- 3 Investigate a number of different keyboards that are available for purchase. For each keyboard give details of the claimed ergonomic features. Choose the keyboard you like the best and, using the results of your investigation, state why you would choose this one in preference to the others.
- 4 Examine the placement of equipment in the computer room of your school. Describe those features of the placement that are ergonomically good and those that are ergonomically bad. Suggest some improvements that could be made to make this workspace an ergonomically better place to work.

Ergonomic issues regarding software design

Software is the link between the operator and the computer and, like the hardware, it should be ergonomically designed to make the operator feel relaxed and comfortable. A range of software has been developed to meet the needs of all software users, depending on their level of software understanding and their task requirements. Software should be designed to minimise movement, improve speed, and be easy to use. If the software is easy to understand and use it is called ‘user-friendly’. Most people find the GUI (graphical user interface) environment easier to use than trying to remember commands. Instructions are entered by using a mouse and clicking on icons (pictures) and menus.

Software can be purchased that controls the use of other programs and prevents the operator from using typing techniques that might result in RSI. For example, typing speed can be controlled and rest breaks can be built in. PC Dynamics offers a shareware program featuring a carpal tunnel syndrome typing break reminder that pops up an animated hourglass when it’s time to take a break. Furthermore, if eyes are being strained in reading normal text, and they have been tested, use software that increases the size of lettering on the screen.

Case study

Microsoft Word

Microsoft Word contains features that make it easier for people who are blind or have low vision to read and for those with limited dexterity to write. These features are outlined in the Microsoft Word Help screen, and are reproduced here:

- The magnification of your document can be changed.
- Shortcuts can be used to insert frequently used text and graphics.
- The AutoComplete feature can be used to insert entire items—such as today’s date, the day of the week or month, your name, or AutoText entries—when a few identifying characters are typed.
- A toolbar can be created that contains only the buttons and menus you use most often. The toolbar buttons can also be made larger and the related ones grouped together.
- A toolbar button or a menu command can be created or a shortcut key assigned that can be used to quickly gain access to frequently used commands, styles, AutoText entries and fonts. For example, you can create a shortcut key that applies a frequently used paragraph style or character style.
- Lists of all the shortcut keys available can be viewed and printed.
- The Microsoft IntelliMouse pointing device can be used to scroll and zoom directly from the mouse. For example, you can automatically scroll to the end of the document with just one mouse click and without using keys.

Acceptable response time in software

One of the most annoying features about some software is its slow response to the entry of commands. A user expects a computer program to react ‘immediately’ to a request. For example, the tool for drawing a rectangle on the screen should respond immediately to the movement of the cursor if it is to be useful. Any delay in displaying the rectangle may lead to errors in the document. It is also important that similar items have similar response times, as users often anticipate the result of an action before that action has been completed. This means, for example, that no matter which of the pull-down menus is chosen, that menu should be displayed as quickly as all the others.

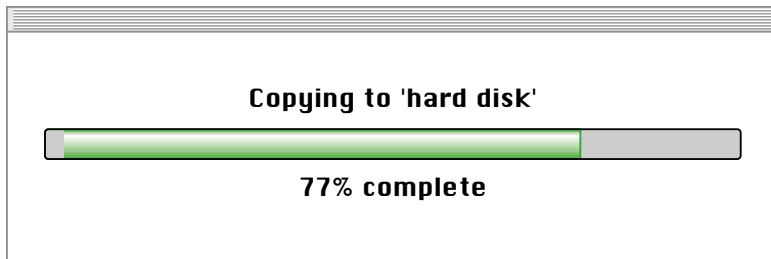


Figure 1.7 A 'thermometer' can be used to show the progress of an action.

Where the response of a program to a command is going to be slow, there is a need for the program to indicate that an action is being undertaken. Graphical user interfaces, such as Windows and the Macintosh operating system, have the ability to change the cursor to a watch or hourglass thus giving an indication that something is occurring. Another common way of indicating that an operation is occurring is to show progress by means of a 'thermometer'.

'User-friendly' software

As computers have become faster and more powerful, it has become possible to create software that presents a more friendly user interface. Early computers were programmed by completing physical circuits by means of wires, plugs and sockets, which meant that they were very hard to use. As technology progressed, first paper tape and then punch cards were mainly used, which still required a specific knowledge to operate the computers. The advent of the cathode ray tube as a display device opened up the use of computers to a wider group of people, as it was possible to communicate with the user by means of text in real time. The text-based user interface became widespread, allowing people with little technological knowledge to use the power of a computer as long as the messages were of the appropriate type. Now that the graphical user interface is widely used, people with virtually no technical knowledge can interact with a computer.

In order to help both the experienced and the inexperienced user, a software designer has to design the user interface with three factors in mind. The first is to make the interface as easy to use as possible. Messages to the user are an important part of the communication process and must be well structured. The interface must also be consistent across the whole of the application, in order to improve the user's confidence, speed and accuracy. These factors will be examined in detail later in the course but will be looked at briefly here.

Ease of use

Features that affect the ease of use of a software application are its **learnability** (how easy it is to learn to use the application), its **flexibility** and the **robustness** of the interface.

Learnability often relies on the application being put in a familiar context, with analogies between the tools available on the computer and those used for a similar task in a non-computer setting. For example, graphics manipulation programs use various styles of 'brush' to create an effect with the colour being applied. This mirrors the effects that can be achieved with paint and various methods of applying it to a canvas.

Flexibility allows the user to customise the application to work in the most familiar way. For example, many word processors can be customised to automate common tasks and to present menus that contain those commands a user will be most likely to use.

Robustness of an interface refers to the way that it stands up to use. The user is using the computer to achieve a certain goal, such as creating a letter. Features that assist in the achievement of the desired result contribute to the robustness. Among these features are the ability of the user to gauge progress from the display, the way in which the program allows the users to recover from an identified error, the rate at which communication takes place between the user and the computer, and whether the application performs all the procedures the user needs to complete the task.

Appropriate messages to the user

Messages to the user are an important part of the user interface as they are the way in which the computer communicates with the user. Language should be plain and non-threatening, and any messages should avoid attempted humour and derogatory comments. The messages must convey information as clearly as possible.

Consistency of the user interface

One of the best ways of ensuring that a user gains confidence in using a new software application is to make sure that screens are presented in a consistent way. This involves placing similar items in the same place on all screens, separating items in the same way and making sure that all dialogues are presented and responded to in the same way.

Exercise 1.3

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Software should be _____ designed to make the operator feel _____ and _____. Software should be designed to minimise _____, improve _____ and be _____. Most people find the _____ easier to use than trying to remember _____. _____ in a GUI are entered by using a _____, and clicking on _____ and _____.

- 2 Examine the features of the word processor you use and list those that could help disabled people in their work. Save your answer as a word-processed document called FEATURES.
- 3 Describe the features of your word processor that make it easy for you to use. In what ways do you think that your word processor could be improved? Give reasons for your answer. Present your answer as a word-processed report or as a slide presentation.
- 4 Explain why a program needs to react immediately to a user's request. Give examples to support your answer.
- 5 Compare two pieces of software that perform the same task. In your comparison, examine the user interface and choose the one you think has the better user interface. Give reasons for your answer.

Intellectual property

Books, newspapers, paintings, photographs, music, films and television programs are all the result of a creative process. Each of these products is covered by copyright which protects the interests of the creator. Copyright in these products acknowledges both the fact that the creator has produced an article and the right for the creator to be paid for the thought and effort involved in its creation. The laws of copyright protect authors' ownership of their intellectual property.

Computer software and files are no different from artistic products, as their creation also involves a great deal of thought and effort. Software is covered by the laws of copyright. As it takes a great deal of money, skill, time and effort to produce and maintain application software, copyright also protects the large monetary investment made by the software company. The main problems found with software items are the ease of copying and the difficulty of detecting breaches of copyright.

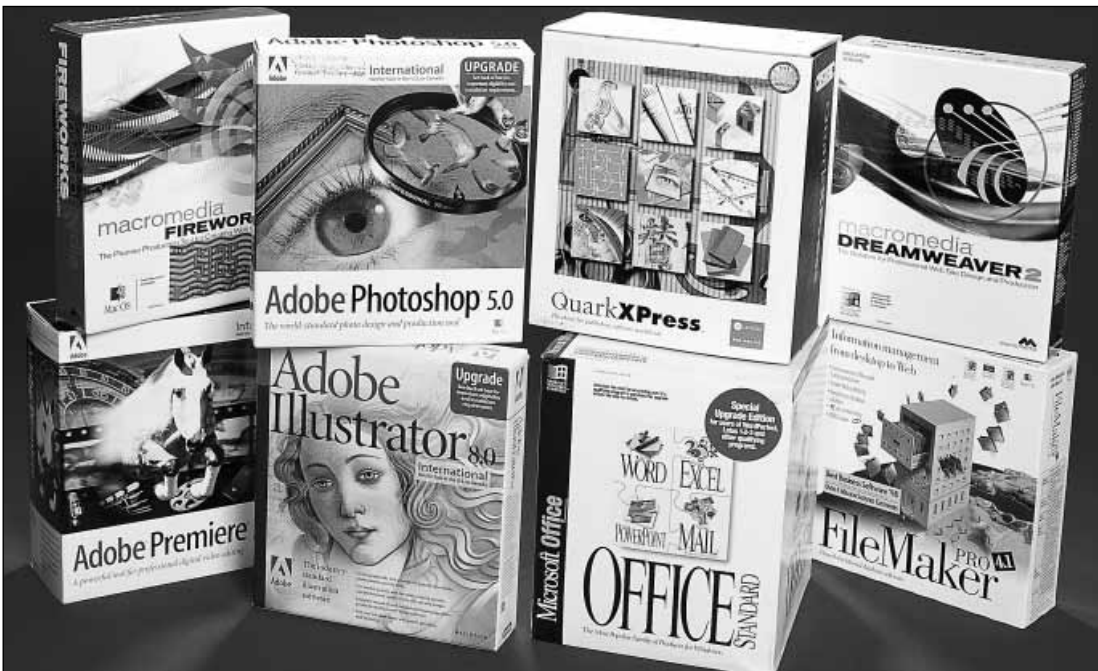


Figure 1.8 Software is governed by the laws of copyright, in the same way as books, magazines and videotapes.

Software licence agreements

When software is purchased, it comes with a licence agreement. This agreement is a contract between the purchaser and the software company, allowing the purchaser to legally use the software. When purchasing a licence, the user is not buying the software itself but the right to use it. The code still remains the property of the vendor. There are different types of agreement covering single and multiple users. Single-user licences allow the installation and use on one computer. Multiple-user licences can vary from allowing use on a fixed number of computers to allowing use on all computers on one site or within one organisation. A distribution licence granted to a school is a special licence that allows the school to legally distribute the software to students and staff as well as install it on the computers on the school campus.

Software is often distributed through outlets other than shops, such as direct downloads from the Internet. It is very hard in these cases to incorporate the paper licences that contain the agreement. These software items come with a licence in electronic form, often as shareware or freeware. A shareware licence allows a user to trial the software before paying the developer for a full licence. Shareware may also be configured so that, when a licence has not been purchased, some features are disabled, or reminder messages appear on the screen. Freeware is usually accompanied by a licence but no fee has to be paid to the developer. Even though a software title is freeware, it is still covered by the laws of copyright and cannot be distributed or used in any way that is contrary to the licence agreement.

Licence terminology

Since the software licence is a legal document, it is often written in legal terms. Some of these terms and their definitions are listed below:

- **Licence:** This is a legal statement that allows a purchaser to use software produced by another person. Buying a licence allows people to *use* the software; they do not *own* the software.
- **Use of the program:** This means that the program is either in RAM or on the hard disk of a computer. Even though it may not actually be running at the time, you are still using a copy of the program if it is loaded on the hard disk of the computer. However, some licences state that a program installed on the hard disk of a network server is not in use until it is in the RAM of those terminals connected to the server.
- **Network:** This term means two or more computers connected together.
- **Reverse engineering:** This term refers to the conversion of the machine code as provided on the CD or disk back into a human-understandable form such as assembly language. Licences disallow this practice as the machine instructions are an expression of the programmer's thoughts, in the same way that the words in a book are the expression of an author's thoughts.
- **Derivative works:** This term means a program which is basically the same as the copyrighted one. For example, a program which teaches Japanese cannot be modified to make a program that teaches Chinese. If someone wanted to produce a program to teach Chinese, then they have to build it up from the beginning.

Legal aspects

Copyright consists of a set of legal rights given to authors of original material. Software covered by the laws of copyright includes programs, applications, data codes and manuals. Also included are any documents included with the original software, such as sample files and templates. Not included are user-generated documents such as word-processing documents or spreadsheets files. However, if these documents include items such as clipart, which has been supplied with the application, the item is still regarded as subject to the conditions of the copyright.

When buying a licence, the conditions of use are specified in that licence and the software must be used and stored according to those conditions. Generally these conditions will cover aspects such as the following:

- installation on a fixed number of computers
- the keeping of backup copies in case the original fails.

Only the copyright owner has the rights to the following:

- reproduction of the software in a material form
- publication of the software
- broadcasting of the software (including distribution by such means as the Internet)
- adaptation of the software in some form.

As mentioned previously, there are three types of software licence: freeware, shareware and commercial. However, it should be remembered that even if a program is freeware or shareware the author's rights are still protected by the laws of copyright.

The *Copyright Act 1968 (Cwlth)* and the *Copyright Amendment (Re-enactment) Act 1993 (Cwlth)* are the only Australian laws covering copyright. They provide for penalties of up to \$60 500 for individuals and up to \$302 500 for organisations who commit offences under the Act.

Use of software covered by a licence agreement

It is the responsibility of the user to fully understand and abide by the software licence that comes with an application. By installing and using a software application, a user is entering into a legally binding contract.

Exercise 1.4

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Copyright protects the _____ of the creator of a software title. Authors create an _____ property when they design software. Copyright also protects the _____ made by the software company. When purchasing a _____ licence, the buyer is only buying a _____ to use the software; the code still remains the _____ of the seller. The most common forms of software licence are _____ user _____ user and _____ licences.

- 2 Choose one software licence and rewrite it, keeping the same meaning but using simpler language.
- 3 Obtain copies of several software licences and compare the terms of the agreements.
- 4 Explain, in your own words, the meanings of the terms that you have found in the software licences you have examined.
- 5 Investigate some legal cases involving copyright. The Internet is a good source of material, including various judgments. Choose one case and write a report discussing what the case was about, who was involved and what the outcome of the case was. Present your report as a word-processed document called LAWCASE.

Origin of software design ideas

The first computers were designed with scientific and military applications in mind. The programs, therefore, were specialised applications in these areas. As computers became more accessible, so the applications became more diverse. In

the early days of commercial use, for example, software was mainly concerned with bookkeeping duties. Now the same enterprises use computers for a wide variety of tasks.

Many of the tasks performed by software are adaptations of non-computer tasks, for example text processing and database management systems. These software applications have been refined and now provide capabilities that were not available to the general population. We will look at the evolution of the graphical user interface and search engines as an example of this evolution.

Evolution of existing concepts

The origins of the GUI can be found before the advent of the electronic digital computer. In 1945, Vannevar Bush published an essay called 'As We Think', in which he talked of a desk consisting of multiple screens, with a keyboard, buttons and levers. The desk, called a Memex, was able to store and display a large amount of information.



Figure 1.9 The Apple Lisa was the first commercial computer to use the GUI interface.

During the early development of the computer, the human interface was of secondary importance. Most input was by punched card and paper tape, the human readable output usually in the form of printed matter. As computers became more widely used, they employed the cathode ray tube for the display of information, but only using a text-based interface. Some of these computers were able to display some form of graphics, but they were not used in the interface. In 1972, Xerox produced the Alto and Star computers that ran by using a bit-mapped screen display. The Xerox display employed icons which were selected then acted on from a pop-up menu. It was not until 1983, when the Apple Lisa was released, that the GUI interface became commercially available.

Unlike the Xerox machines, the Lisa had overlapping windows and the direct manipulation of icons and windows. During development, the menu bar was placed at the bottom of the window, but before the release of the computer it was moved to the top. The Lisa also used the concept of the desktop, with documents being able to be manipulated on it. Although the Lisa was not a commercial success, it left a legacy of the graphical user interface. Although many different graphical operating systems have been created since the Lisa, very little has really changed in the implementation of the interface.

The Internet is such a large unstructured store of information that it would be almost worthless were it not for search engines. The development of more and more sophisticated methods of searching have allowed the Internet to become the useful tool it is now. Early search engines were Archie, WAIS and Gopher. Archie was developed in the early 1990s to help find files on the networks forming the Internet. It obtained directory listings of all the files on thousands of sites around the world and put them in a database. This database was searchable through a simple interface. WAIS was developed as a source of information for

big business, and was able to be searched by using questions in natural language (normal language such as English). Gopher was developed by the University of Minnesota to help its staff and students by distributing campus information. These search engines formed the basis for those that are available today.

New and exciting approaches

Many software applications that we take for granted now are the result of creative lateral thinking. Among these applications are spreadsheets, web browsers and presentation software.

The story of the spreadsheet began in 1978, when the program Visicalc was launched for the Apple II computer. Visicalc was based on the paper spreadsheets used by businesses, but it had the advantage that calculations could be made automatically. The functions provided in Visicalc were minimal when compared with modern spreadsheets. As the spreadsheet developed, further functions were added, increasing its application to tasks outside finance. Added to these features were the ability to turn tabulated material into graphs and the further ability to automate tasks by means of macros. The spreadsheet, more than any other application, took the personal computer from being an enthusiast's 'toy' to being an important part of the modern office.

The growth in the use of the Internet is almost entirely due to the ease with which it can be navigated. The applications that give us the ability to find our way around the Internet are the browsers. Before 1993 the Internet was basically an ASCII-text-based network. In that year the graphical browser Mosaic was first released by NCSA (The National Center for Supercomputing Applications) for a number of different computer platforms. The release of Mosaic allowed full use to be made of the World Wide Web (which was created in 1989 at CERN in Switzerland) and the hypertext links it provided. The original developers of Mosaic left NCSA soon after the release of Mosaic, to form the Netscape Communication Company. Although Mosaic and Netscape perform the same tasks,

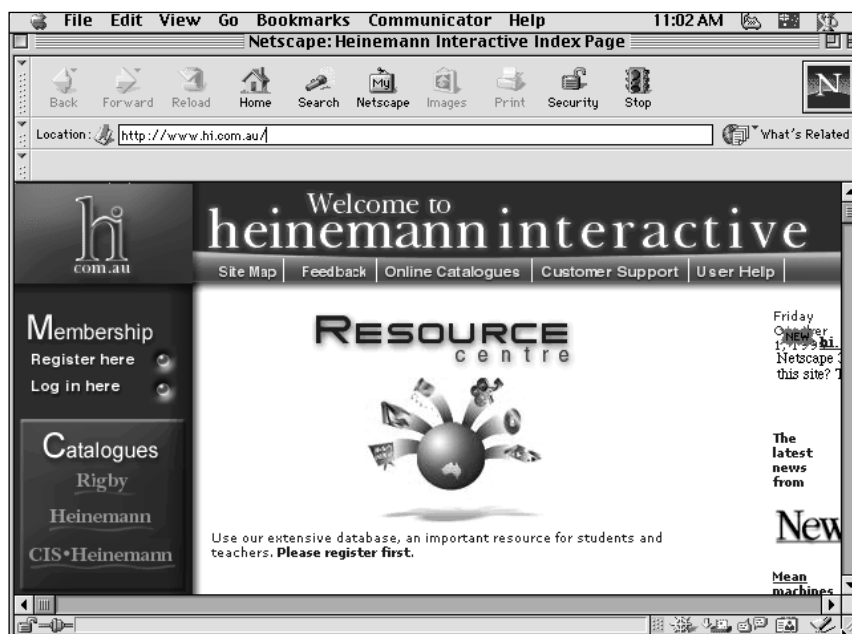


Figure 1.10 Netscape Navigator was produced by members of the Mosaic development team.

Netscape was written from scratch, solving a number of bugs that were inherent in Mosaic. Internet Explorer from Microsoft was developed from Spyglass' Mosaic browser for use with the Windows 95 operating system.

Most research seems to suggest that the first application to be used for presentations was the Hypercard programming environment released by Apple in August 1987 for use on its Macintosh computer systems. Although not specifically designed for presentations, Hypercard found favour due to the ease with which links could be formed between various elements of a presentation. There are now a number of differing software titles that may be used for presentations, some forming part of an integrated package, others being designed specifically for presentations.

Exercise 1.5

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Software was written for the early computers to solve _____ and _____ problems. As computers became more _____, business and commerce found uses for the computer. Now computers are used for a _____ variety of _____. Many of these are _____ of non-computer applications, for example _____ and _____. These _____ have now been _____ for the general _____ who often use them at home and at _____.

- 2 Use the Internet to investigate the evolution of a software application, for example spreadsheets, or of a particular application such as Hypercard. Use presentation software to help you with a talk to the class. Save your file as HISTORY.
- 3 Examine the operating system used by your school computers and suggest what that operating system may be able to do in five years' time. Use a word processor to present your report. You may like to take some screen dumps from the existing operating system and use a graphical package to modify the screens so that the graphics illustrate your points.

Events that have led to the need for software licence agreements

As we have seen earlier, a number of different factors have led to the need for agreements to cover the use and distribution of software. The main factors are the ease with which software can be copied, the collaborative approach to writing applications and the availability of software from the Internet.

In the early days of computers, software was usually written for a specific application on a specific computer. Copying of such software was generally not an issue since there was a relatively small number of computers around. With the advent of the personal computer and commercially produced applications aimed at the home and small business market, the acquisition of software has become a regular occurrence. The ease with which software can be duplicated or transferred from one computer to another has often encouraged people to 'borrow' software from various sources. In these circumstances, software developers have had little choice but to strengthen the licensing arrangements to ensure that they receive a fair reward for the thought, time and effort put into creating a software

package. The agreements drawn up now assign only a right of the purchaser to use the software and not ownership which remains with the developer.

One of the major costs associated with application software is that of the development team. Duplication and distribution of a software title are a very small proportion of the total cost in bringing out a new title. Many teams of very skilled people work for a long time to generate, test and debug commercial software. If the developers are to continue in their business, the rewards must be there for them. The only source of these rewards is by means of sales.

The Internet poses a large problem as far as copyright is concerned. The ease with which software can be made available to a global audience is the greatest worry. The openness of the Internet and the lack of control mean that anyone is able to publish anything without regard to the source of the material.

Exercise 1.6

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

_____ to cover the use and _____ of software have become necessary because software is now _____ to copy. Software developers employ large _____ of people to _____ the software. This is an _____ process and the _____ developer must be rewarded if more software is to be _____. Licence _____ ensure that the _____ receives payment for the _____ and effort put into _____ the software.

- 2 Explain what would happen to software development if licence agreements were not legally enforceable.
- 3 What difficulties do software developers have in enforcing licence agreements? What body in Australia helps software developers ensure that software is used legally?

Sources of code and conditions that apply

There are various sources of code: the Internet, books and magazines, freeware and shareware. The publication of a code in one of these forms does not necessarily give a person the right to use it. Each of these sources is covered by the laws of copyright, and the programmers need to be aware of their responsibilities in this regard. For example, in some cases, portions of code may be reproduced for personal use, but not distributed.

There are a number of freely accessible sites and newsgroups on the Internet that cater for programmers in many computer languages. Code samples obtained from these sources may be free of copyright, as these places are a forum for discussing



Figure 1.11 Books and magazines can be a source of code, but the laws of copyright still apply.

various problems. It is not hard to check on the status of any code on the Internet, as a short e-mail to the code's author will clarify the situation.

Books and magazines are subject to copyright. If the code within one of these sources is free to distribute, it will usually be noted in the copyright notice in the source. If you are in doubt about the copyright status of any of the code, then a letter should be written to the publisher to ask about the status.

Freeware and shareware are generally not sources of code, as they are covered by the same laws of copyright as commercial software. Some freeware licences do state that the code may be modified but not for profit. In these cases it is possible to use the code, as it is a part of the licence. However, if the code is modified, it is a general courtesy to send a copy of the changed application to the original author.

Exercise 1.7

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Sources of computer code are _____, _____, _____, _____ and _____. 1 of these sources are subject to _____ Sites and _____ on the Internet that are used by _____ ma provide coe that is free of _____. Freeware and _____ are gerally unable to be used as _____ of code, as the softwre tem is covered by a ac_____.

- 2 Investigate books, magazines and the Internet as sources of code and report on whether the code you find is free of copyright or not. Create a database of code sources that are able to be used. In this database, you will need to have fields that contain the source of the code, its purpose and the computer language that the code is in. Save your database as CODEDATA. This will be a useful database later in the course when you will need to find code samples.

Inclusivity

Software design and development need to allow for many human perspectives. These include cultural, economic, social, gender and disability perspectives. No member of society should be excluded from using a software application for any of these reasons.

The need for software design and development to be inclusive

Software is designed to be used by people. When developers set out to create a new software item, they need to ensure that it is available to the widest possible audience. The following perspectives should be taken into account:

- *Cultural.* Software should not assume any cultural heritage. The language of prompts and messages needs to be written in plain and simple English, for example. Images should not reflect any bias towards one culture.
- *Economic.* When writing software, a programmer should not assume that the user will come from a particular type of economic background.
- *Social.* Software should not emphasise any one particular social background.

- *Gender.* A software title should not be written in such a way as to appeal to either males or females. Software should also provide messages and prompts that are gender-independent.
- *Disability.* People with disabilities should have access to the same technology that the able-bodied have. Software should be structured in such a way as to encourage those with disabilities to take advantage of the features. For example, the incorporation of a magnifying feature in a word processor will assist the visually impaired in working with that program.

Exercise 1.8

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
The human _____ that software developers need to allow for include _____, _____, _____, _____ and _____ perspectives. This is done so that software is made _____ to the widest possible _____
- 2 Choose an application and examine it in terms of the five perspectives. How do you think that the application could be improved to allow it to be used by a wider audience?

The general strengths brought to the field of software design and development

Creation of software is not only beneficial to the user, but it also develops the strengths and skills of the members of the development team. Apart from the obvious skills in working with others, team members' experiences enhance their communication skills, exercise their creativity, sharpen their design skills, improve their problem-solving ability and refine their attention to detail.

Many projects are completed as a team effort and, as such, there is the need for different people to work together. Members of a team have to appreciate the skills of others as well as their own shortcomings.

Communication is really at the core of all software projects. Whether large or small programs are being created, the programming team has to provide documentation. These documents vary from memos and notes for the use of the development team through to the user manuals that accompany the finished product. One of the hardest tasks faced by a programmer is to create the clear and simple messages that are needed as part of the program interface.

Programming is a very creative process. A programmer needs to be able to make a product that not only functions well but also acts in a familiar way. Screen design is the obvious way in which creativity is shown, but there are many other ways in which it can be displayed. For example, a programmer may find a simpler and faster way of obtaining a result.

The design of a process that takes place within a program is no different from the design of an object such as a car. A software developer needs to be able to analyse the needs of a user and turn them into a useful product which fulfils all the requirements.

The process of software development is not an easy one. Many problems will occur during the process, some with the program and others with documentation

or final implementation. A software developer needs to be able to overcome these problems for the project to have a successful outcome. Problem-solving skills such as lateral thinking are developed to a high degree in this context.

Finally, the program being developed must fulfil all the requirements set down by the original specifications. This means that all the fine details must be attended to. An application is not going to be successful if it fails to work under a rare, but possible, set of circumstances. For example, a word processor is not much good if it works properly for all combinations of characters except for the three letters *i*, *o* and *u* following each other. This word processor will work correctly most of the time, but if the word *facetious* is used, then it will fail.

Exercise 1.9

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Computer programming can enhance the _____ of the members of the team. The skills used in program development are _____, _____, _____ skills, _____, living ability and attention to _____ working as a member of a _____ helps people to work with _____ as well as helping with _____. Development of software involves a lot of _____, some for the _____ team and some for the _____. This means that a developer needs good skills. _____

- 2 Each of us has differing skills in each of the areas mentioned in this section. For each of the skills listed, state whether you think you have that skill or need to develop it further. Give reasons for your answer for each of the skills. Choose the skill that you think is your weakest and plan a course of action to improve it.

Team Activity Ergonomics

A large business is moving its entire office staff to a new building. The business requires the best ergonomic standards, privacy for computer work, and facilities that promote 'teaming' within the office. Write a report that makes recommendations in the following areas:

- furniture design—desk and chair
- placement of computer equipment—screen, keyboard and mouse.
- work environment—lighting, indoor climate, placement of work areas and noise.

Review exercises

- 1 Measure a typical chair in one of the classrooms at your school. Explain why this chair is not ergonomically suitable for use at a computer desk.
- 2 Compare a number of different computer mice by looking at the features of each. Construct a database of mice which lists the features of each, rates the 'feel' of each as good, fair or poor and records the cost of each mouse. Using the information from your database, make recommendations as to the best value mouse and the best mouse ergonomically.
- 3 Investigate the causes, symptoms and cures for either carpal tunnel syndrome or tenosynovitis. Explain what can be done by a computer user to minimise the risk of these disorders.
- 4 Describe how the workplace environment can affect the health and performance of a computer user. Illustrate your answer with appropriate examples.
- 5 Explain, in your own words, what makes a computer program easy to use. Choose a public domain program and analyse how easy it is to use. Publish your report with the filename EASY2USE.
- 6 Explain why the laws of copyright are necessary. What would happen to computer software if it was not covered by these laws?
- 7 Describe the responsibilities of a computer programmer as regards the use of program code from outside sources.
- 8 Visit the website for the Business Software Association of Australia. Describe the steps that should be taken by a company to ensure that software theft by its employees is prevented.
- 9 Voice recognition software is becoming much better. What new applications, or adaptations of old applications, will occur as a result of this advance? Describe the tasks that the new software will perform.
- 10 You have been chosen to develop a library enquiry system for your local library. What features will you include to make it useable to the largest number of people possible?

Chapter summary

- Ergonomics is the study of the relationship between people and their work environment.
- The work environment includes the operator's furniture, equipment placement, work techniques, software interface and general working surroundings.
- Prolonged use of software may lead to RSI if the environment and work practices are not right.
- Furniture needs to be properly designed and placed for worker to avoid problems caused by poor posture.
- User-friendly software can assist in the prevention of work-related health problems.
- Software that is user-friendly will have acceptable response times, be easy to use, provide appropriate messages to the user and have a consistent interface.
- Objects such as books and computer programs which result from a creative process are covered by the laws of copyright.
- Software licence agreements spell out the conditions under which a software item can be used.
- Licences will specify how the software may be installed, the reproduction of the software and the manner in which backup copies can be kept.
- When installing a software title, the user automatically agrees to the terms of the licence.
- Software design ideas have come from a large number of different sources.
- When using a section of code, a programmer needs to ensure that all the laws of copyright are followed.
- Software should be available to the whole population, taking into account various perspectives.
- Software design benefits both the user and the designer, as the designer acquires or enhances skills in many areas.

chapter 2

→ *Hardware and software*

Outcomes

- describes the functions of hardware and software (P 1.1)
- describes the interactions between the elements of a computer system (P 1.3)
- describes developments in the levels of programming languages (P 2.1)
- explains the effects of historical developments on current practices (P 2.2)
- identifies the issues relating to the use of software solutions (P 3.1)
- describes the role of personnel involved in software development (P 6.1)

Students learn about:

Hardware

- the function of hardware within a computer system, namely:
 - input
 - storage
 - output
 - control
 - process
- the operation of a variety of input devices, output devices, storage devices and CPU components
- the current trends and developments in computer hardware

Software

- system software, including utility software
- applications packages and custom-designed software
- generations of programming languages, namely:
 - machine
 - higher level languages

- assembler
- declarative languages
- event-driven versus sequential approach
- the need for translation
 - compilation
 - incremental compilation
 - interpretation
- characteristics of different operating systems, including:
 - command-based or graphical user interface
 - multi-tasking
- current trends in the development of software and operating systems

The relationship between hardware and software

- processing of software instructions by hardware
 - the 'fetch-execute' cycle
- the initiation and running of an application
 - start fetch-execute cycle
 - display the start screen
 - locate on disk
 - wait for user input
 - load into RAM
- the existence of minimum hardware requirements to run some software
- elements of a computer system, including:
 - hardware
 - procedures
 - software
 - personnel
 - data
 and their role in software design and development

Students learn to:

- describe how data is captured, stored and manipulated on a variety of hardware devices
- competently use computer hardware, selecting appropriate hardware for specific tasks
- competently use a range of software
- describe the development of subsequent generations of programming languages
- appraise the effect of the operating system on the tasks that the system can perform
- interpret and use an ASCII table
- identify the elements of a computer system
- describe the significance of each element in the software solution using a case-study approach

Personal Profile—Charles Babbage (1791–1871)

Born in Teignmouth, Devon, on Boxing Day 1791, Charles Babbage was the son of a banker. Today he is acknowledged as the 'Father' of the modern computer. His analytical engine, which was never completed, was the first machine to be conceived as an automatic calculating device. The machine would have used punched cards to store both data and instructions, very much like the early-generation electronic computers. The other feature that this device had in common with the modern computer was the division of the machine into three basic units: the store, the mill and the control. These units correspond to the parts of a modern computer system: storage, process and control.



Little is recorded about Babbage's life before he entered Trinity College Cambridge in 1810. While at university he married Georgiana Whitmore, whose half-brother Wolryche was a member of parliament. He graduated from Cambridge in 1817 with a MA degree in mathematics.

In 1823 he started work on his 'analytical engine' which was designed to calculate and print mathematical tables. Finance for the project had been provided by a government grant of £1500. Babbage stopped work on the project in 1827 when his wife died, and travelled to Europe. By the time he returned in 1828 the money had gone. In 1829, with the help of some of his friends, Babbage managed to gain a further grant of £3000 from the Prime Minister, the Duke of Wellington. Disagreements with his construction engineer forced work on the engine to stop in 1830. In 1834 Babbage approached the new Prime Minister, Lord Melbourne, for funding for his machine, the analytical engine. However, the government was unwilling to give him more money as the difference engine was not complete. He continued to lobby the government for funds until 1842, when he was told that the funding would not eventuate. By 1851 he had come to the conclusion that the analytical engine would not be constructed.

During this period Babbage continued to work in other fields, having been appointed Lucasian Professor of mathematics at Cambridge University in 1830, although he never actually gave a lecture. He was a founding member of the Analytical Society (formed in 1823), the British Association for the Advancement of Science (in 1831) and the Statistical Society of London (in 1834). He was also involved with the construction of railways and published various books and articles. When he died in 1871, his passing was barely noticed. It was not until the advent of the modern computer that Charles Babbage's contributions were appreciated.

Hardware

The function of hardware within a computer system

A computer programmer writes the software that directs the hardware to perform a particular task and so solve a problem. The software consists of detailed instructions that control the hardware. When writing these instructions, programmers view hardware in terms of their five logical elements: **input**, **processing**, **control**, **storage** and **output**.

Input involves entering data into the computer for processing into information. The term 'data' refers to the raw facts used by the computer, such as letters and numbers. 'Information' is data that has been ordered and given some meaning. Information is the result of work on the computer and depends on the data entered. If the data entered is inappropriate, the information presented will be meaningless. A related computer saying is 'garbage in, garbage out' (**GIGO**). Input devices include keyboards, mice, disks and scanners.

Processing involves changing data to produce information by following a series of instructions. Processing is performing the required task and is carried out by the **central processing unit (CPU)**. The CPU takes the data entered from an input device, changes it in some way to produce information, and sends it to an output device to be presented.

Control coordinates the operations of the input, processing, output and storage. It is performed by the control unit which is part of the CPU. The control unit is the organiser that directs the flow of data in the computer, in the same way as traffic lights control the flow of vehicles at an intersection.

Storage involves receiving and retaining data over a period of time, allowing it to be accessed and retrieved when required. Storage may be classified as primary storage or secondary storage, and as permanent or temporary. **Primary storage** holds data and programs before and after they are processed by the CPU, and is called 'internal storage'. **Secondary storage** is more permanent and is called 'external storage'. It uses media such as hard disks, floppy disks and CD-ROM to store data.

Output involves the presentation of information to a person, or data to another computer. It includes the transfer of data from primary storage to an output device such as a monitor or printer. The information presented is the result of the operator's work on the computer.

All the elements of hardware work together. Data is entered using an input device and processed in some way before being presented using an output device. If necessary, data can be held on a storage device for later use (see Figure 2.1)

These five elements can be seen in the type of computer used at your school, called a **microcomputer** (see Figure 2.2). A microcomputer is only one member of a large family of computers and is also known as a PC (personal computer) or home computer. Data is entered using a keyboard (or mouse) and then changed in some way and presented on the screen. These tasks are performed under the control of a program that resides in the computer's memory. If required, the data can be stored on a disk. All types of computers, from large to small, have input, process, control, storage and output.

Peripheral devices

Peripherals are devices other than the CPU. They are often attached to the computer and include input and output devices, and secondary storage. For example, the keyboard, mouse, monitor, printer and disk drive are all peripheral devices.

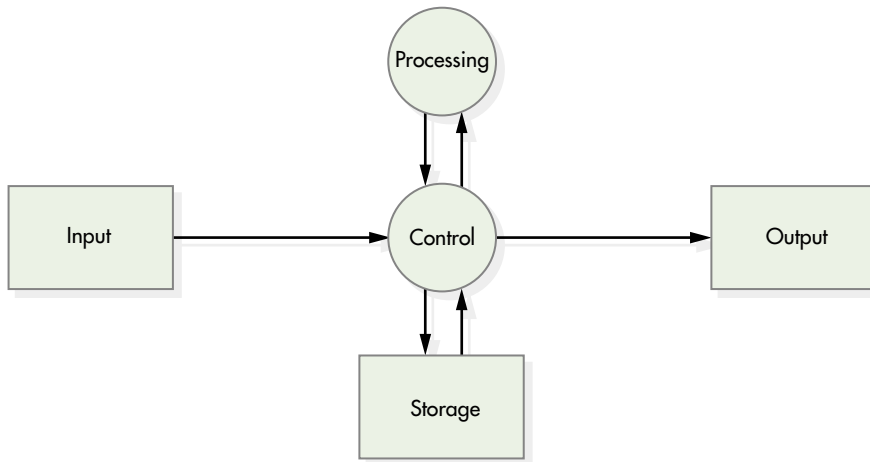


Figure 2.1 The elements of hardware.



Figure 2.2 A personal computer system contains all five elements of a computer system. Which ones can you identify from the photograph?

Exercise 2.1

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

All computer hardware systems consist of five logical elements. These are _____, _____, _____, _____ and _____. Entering data is called _____. Producing information from this data is done by _____. Coordination of events is carried out by the _____ of the computer. When data is kept or later used, its spaced in _____. Presentation of information is carried out by these _____ of the hardware system.

- 2 For each statement, select a matching word or phrase from the following list: control, data, hardware, information, input, microcomputer, monitor, output, peripherals, storage.
- Data that has been given order and some meaning.
 - Presentation of data or information to a person or another computer.

- c Receipt and retention of data.
 - d Another name for a personal computer.
 - e Entry of data into the computer for processing.
 - f Pieces of computer equipment other than the CPU.
 - g An output device.
 - h Coordinates the operations of the input, processing, output, and storage.
 - i The computer equipment that you can see and hold.
 - j Raw facts put into the computer system.
- 3 Name the five elements of hardware. Use a drawing program to illustrate how data passes between these elements. Save the file on disk and name it **HARDWARE**.
 - 4 Explain the role of storage in a computer. Use your creation of the file **HARDWARE** in question 3 to illustrate how storage is an important part of the hardware system. You can present your answer as a word-processing file called **STORAGE1**.
 - 5 Name the two types of storage. Use the example of your file **HARDWARE** to explain the similarities and differences between these storage types. You can present your answer as a word-processing file called **STORAGE2**.
 - 6 Create a small database with two fields called **DEVICE** and **TYPE**. Enter as many peripheral devices into this database as you can. Each peripheral should be classified in the **TYPE** field as an **INPUT**, **OUTPUT**, **PROCESSING**, **CONTROL** or **STORAGE** device. You can save your database as a file called **PERIPH**.

EXTENSION

- 7 The typical advice given to a person wanting to buy a personal computer is that they should first choose the major software needed, and then choose the hardware that will execute the software. However, a consultant strongly advised a person to buy an IBM-PC without considering the software. Evaluate the consultant's advice. Present your answer as a word-processing file called **COMPBUY**.
- 8 The introduction of a personal computer to an office makes some people feel insecure and inadequate. What remedies would you suggest to solve this problem? You may, if you wish, present this answer as a word-processed document.

Input devices

A computer needs data and instructions before it can work. Entry of data from outside the computer system is called **input**. Once taken from the outside environment, data needs to be changed by the input device into a suitable form to be processed. There are many types of input devices used for different purposes.

The keyboard

The keyboard is the most commonly used input device. It consists of a number of rows of electrical switches. Electronic circuits inside the keyboard convert the keypress into a code which represents a character. The code is then passed on to the processor.

There are many different layouts for keyboards. The most common, named the **QWERTY** keyboard after the first six letters on the top row, was developed for mechanical typewriters then adapted for computer use. The main disadvantage

of the QWERTY keyboard is that it was designed to slow down the typist in the days of mechanical typewriters. Other keyboard arrangements have been invented for the purpose of improving the speed of input, the most notable being the **Dvorak** keyboard. However, the cost of retraining typists and re-equipping computers with new keyboards makes these other designs uneconomical and they are not widely used.

Most computer keyboards have extra keys that are not found on typewriters. The purpose of these keys is to modify the code sent from the keyboard so that it can stand for another character. For example, commonly found modifier keys are the **control key**, the **alt key** and, on Macintosh computers, the **command key**. Other keys not normally found on typewriters are programmable **function keys**, an **escape key** and numerical keys arranged as a separate **numerical keypad**.



Figure 2.3 Computer keyboards are very much like a typewriter keyboard, but they usually contain more keys, which makes the keyboard more flexible.

Pointing devices

Pointing is an instinctive ability. We learn to point before we can stand, sit or talk. Thus it can be easier to point to something on the screen rather than to enter a command into the computer using a keyboard.

A **mouse** is an input device that can be used to move a pointer on a screen. It consists of a ball that is rolled over a flat surface (see Figure 2.4). The ball's movement on the surface causes the cursor to move to a corresponding position on the screen. Clicking or pressing a button on the mouse allows a character or command to be selected from the screen.

A mouse is now supplied with most PCs because people find them easier to use than keyboard commands. The mouse can move the cursor to any point on the screen with a single sweep of the hand.

The mouse is an important device used in a **graphical user interface (GUI)** where choices are made by pointing to an icon or menu, then clicking on the mouse button. It replaces the **command line interface** where the cursor was moved using nine separate keys on the keyboard (four arrow keys, tab, page up, page dn, home, and end) and choices were made by using a combination of one or more keys.

The movement of the mouse ball is translated into two signals, one representing the 'vertical' movement of the mouse on the mat and the other the 'horizontal' movement of the mouse. These two signals carry data about the distance moved by the mouse in each direction. The data is then converted by the computer program into movement of the pointer on the screen.



Figure 2.4 A mouse connected to a computer allows the user to 'point' to a wanted item.



Figure 2.5 A trackball is really only a mouse on its back.

A **trackball** is similar to a mouse, except that the ball is on top of the device instead of the bottom (see Figure 2.5). The operator moves the trackball with the palm, finger or thumb, which allows the cursor to be moved. Trackballs are frequently used in computer-aided design (CAD) applications because their movement is far more precise than that of a mouse. In addition, they do not require a flat surface, making them a convenient pointing device for many laptop computers.

Joysticks are mainly used for computer games. They are used to move the cursor or other object on the screen and, like the mouse and trackball, they translate physical movements into two signals which are interpreted as screen movements by the computer program.

A **light pen** is an input device that can change pictures or drawings into electronic signals. The pen is wired to the computer and can be used like an ordinary pen. It is used simply to 'draw' on the screen in the same way as drawing on paper. The light pen works by detecting the light produced by the scan at the point on the screen. The signal to the computer is translated by the computer program into a position on the screen.

Touch screens enter data by detecting the touch of a finger. Some of them have infra-red light beams shining horizontally and vertically across the face of the screen. The exact location of a finger is determined when the beams are broken. Other touch screens work by changes in capacitance across the screen or by ultrasonic reflections. In addition, there are devices that allow most monitors to be used as a touch screen. The computer program again receives a signal from the device, converting it into a screen position.



Figure 2.6 A digitising tablet can be used to 'draw' directly on a screen.

The **digitising tablet** is another input device which sends signals representing the position of a 'pen' on a special pad. There are several different ways in which this information is measured: by electrical resistance, magnetically by detecting current pulses, and by sonic tablets that use microphones. Many laptop computers use a small digitising tablet called a **trackpad** in preference to a mouse or trackball.

Character readers

Character readers are input devices that can accept text and send it to the computer.

A **barcode wand** is an input device that allows the computer to read the barcodes on packaged products. It usually looks like a thick pen and is rubbed lightly over the barcode. It works by detecting variations in the thicknesses of the bars. This data is then converted to a coded signal and sent to the computer program which changes the signal to a series of digits. Barcode readers in supermarkets are set into the surface of checkout counters.

A **document reader** is an input device that can read text. Many multiple-choice tests require answers to be marked on specially printed answer sheets using a pencil. The answer sheets are corrected using a computer which receives its information from a document reader. This input device consists of a series of lights and photoelectric cells which can read the lead from the pencil.

Optical character readers are used in shops to monitor sales. They are hand-held devices that can read specially printed numbers from a label attached to an item. They are usually attached to a terminal at the shop counter, which in turn is connected to a computer. The computer program detects either the shape of the character or the area of the character, comparing this with a character database. Optical character recognition (OCR) software can now be used to change printed text, such as from a typed page, into strings of character codes which represent the text as a word processor file. Programs which 'read' handwritten text are also becoming more widely available as they become more accurate.

Other graphical input devices

Scanners are input devices that can turn photographs and drawings into bit patterns able to be read by the computer. The bit patterns carry information about brightness and colour in the scanned document. These patterns can be stored, changed, and the pictures printed by a graphics program. In addition, character recognition software can be used to read text and store it in a word-processing program. A scanner works by shining a strip of light onto the page and measuring the reflected light with light sensors (see Figure 2.7).

Digital cameras create bit patterns similar to those produced by a scanner. These patterns are produced by the lens focusing the image on a light-sensitive grid. Digital cameras can send these patterns directly to the computer or store them in a memory for later transfer to a computer. Video cameras work in a similar way, but they can take a sequence of still images very quickly (about 30 each second for full-motion video).

Video input devices take in a signal from a video source such as a video recorder or television tuner. These video cards convert the analogue signal received into a digital sequence which can be interpreted by the computer in the same way as the input from a digital camera or scanner.

Sound devices

Sound from sources such as microphones can be turned into a digital sequence that can be processed by a computer. The process involved in digitising the sound is the same as is used when music is stored on compact disk.

Sounds can also be 'captured' from certain types of musical instruments by means of a **MIDI (musical instrument digital interface)**. The MIDI system does not actually record the sound but uses a code to represent the tones. These codes can be manipulated and saved, then sent to a musical instrument capable of processing them in order to reproduce the desired musical effect.

Data can also be input from another computer by means of disks, modems and network interfaces.



Figure 2.7 A scanner converts an image into a bit pattern which represents the picture.

Exercise 2.2

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
Probably the most common _____ device used is the _____ keyboard. The _____ keyboard is much more efficient than the _____ one. Pointers on screens are usually controlled by a _____ devices such as a _____ and _____ can also be used to control a pointer. Other devices that can be used to input text are the _____ and _____. Other data that can be entered into a computer includes _____, _____ and _____.
- 2 Explain why it is easier to point to a choice on the screen rather than type it into the computer.
- 3 Use a drawing program to illustrate how a mouse works as an input device. Save your work with the filename MOUSE.
- 4 Name all the different pointing devices. Discuss the similarities and differences in the ways they work.
- 5 Describe one task where a mouse is superior to the keyboard and one where the keyboard is superior.
- 6 Why are trackpads a convenient pointing device for many laptop computers?
- 7 List the input devices available to you in your school. Describe the schoolwork tasks that you can perform with each of these devices. Present your report as a word-processed document with the filename SCHOOLIN.
- 8 Scan a photograph of yourself and modify the picture using a graphics program.

EXTENSION

- 9 A punched card is a thin piece of cardboard on which information is recorded as small holes. This information is passed into the computer through a machine called a card reader. Why have punched cards become obsolete?
- 10 'Touch screens are easy to use and will eventually replace the mouse as a pointing device.' Comment on this statement.

Output devices

Output is the presentation of information to a person, or data to another computer. It enables results to be obtained. Computers can present data as text (letters and numbers), graphics (pictures) or as computer-generated/reproduced sounds.

Visual display devices

A **monitor** is a television-like device used to display images generated by the computer. Though similar to a television screen, it is much clearer and provides immediate feedback about what the computer is doing. The monitor is the most common output device and has several names, including the **screen**, **cathode ray tube (CRT)** or **visual display unit (VDU)**.

The **screen** itself is a cathode ray tube with electrons being bounced onto a phosphor coating at the front of the tube to make it glow. The beam of electrons rapidly scans across and down the screen to produce the image. This beam forms

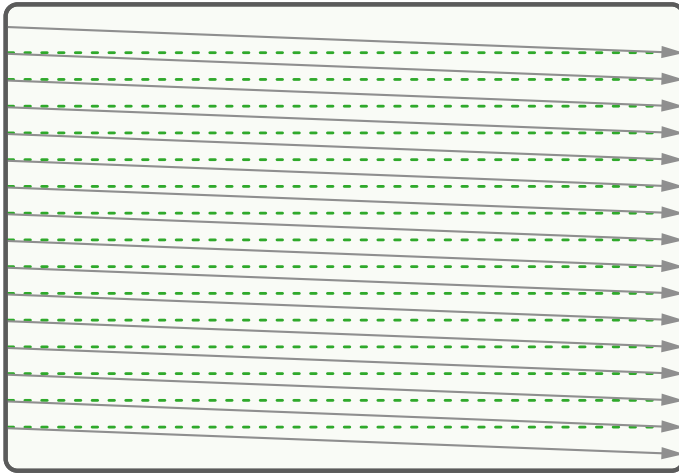


Figure 2.8 A monitor screen uses a raster scan to produce an image.

the picture on the left–right scan by being rapidly turned on and off. When moved from the right to the left to start a new scan line, the beam is turned off. This scan is known as a **raster**.

Liquid crystal displays (LCDs) are flat screens used in portable computers. They use the same type of display as digital watches and calculators. This technology provides displays that are very light, take up less room, produce no heat, have no glare, and create no radiation. Furthermore, LCDs require less power than a CRT, allowing them to run on batteries. These displays use the properties of ‘liquid crystals’ which change direction when subjected to an electric current. Polarising filters are used so that when the crystal turns light is prevented from being reflected back through the filter as its polarisation is changed. (Try looking at an LCD calculator display through polarising sunglasses while turning the calculator slowly from left to right. This will illustrate that the light is not reflected back.)

Screens are classified according to the clarity of their image or resolution. Resolution is measured by the total number of **pixels** (dots) that occupy the screen. A number of standards for computer screens have been developed, each offering a higher resolution and hence a sharper picture.

In addition, screens are available in monochrome or colour. **Monochrome** means ‘one colour’ and screens can be black and white or, in some cases, amber or green on black. Monochrome is cheaper but colour screens are widely used because most software now has many different features that make colour essential. The depth of colour (the number of different colours that can be displayed) varies from computer to computer. Depths usually vary from 256 colours through to millions of colours. The number of bits used to represent each pixel will have an effect on the number of colours that can be displayed.

Hard copy devices

A printer converts data in the form of electric signals to impressions on paper. The paper containing the data is called **hard copy** or **printout**. Printers can be classified as impact and non-impact printers. **Impact printers** make an image on the paper by striking it with a metal print head, while **non-impact printers** make an image using some other method. Impact printers include dot-matrix printers, and non-impact printers include laser and ink-jet printers.

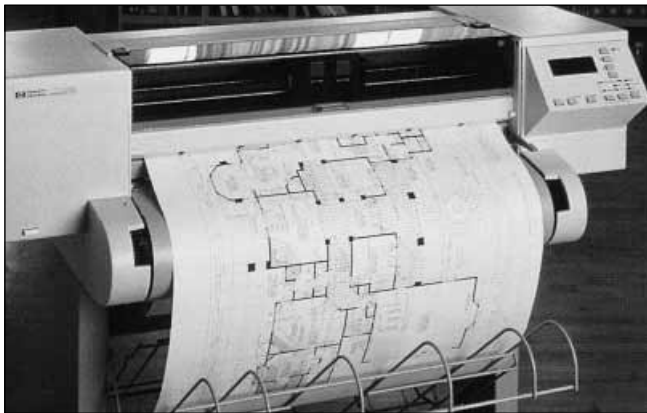


Figure 2.9 EFTPOS machines use dot-matrix printers to make duplicate copies.

Dot-matrix printers were the most common type of printer because they were reliable and cheap. Even though the cost per page of dot-matrix printing is at least four times less than that of any other printing, its popularity has decreased. Laser and ink-jet printers are now more popular because of their much lower unit price and far better print quality. The dot-matrix printer forms its characters as a series of dots, very much like the characters formed on the VDU. Each dot is formed by the impact between a pin, the printer ribbon and the paper. The pins are located in the print head and can range from 9 to 48 all arranged in a vertical line. Dot-matrix printers are also noisy. Impact printers such as the dot-matrix printer are still used in applications where an exact copy of the original is needed. For example, EFTPOS machines still use dot-matrix printers.

Laser printers use a laser beam to create an image on a specially sensitised drum. The drum then picks up the toner that creates the image on the paper in much the same way as a photocopier. Laser printers provide high-quality output, their speed is fast, and they can produce graphics as well as text of varying size and styles. Laser printers tend to be more expensive than other printers.

Ink-jet printers produce characters by spraying very fine ink jets onto paper, or by bursting tiny bubbles of ink by means of a small heater. They can produce colour or black output and are quiet, light and reasonably priced, which is making them increasingly popular.



A **plotter** is a specialised printer used in architectural and engineering design. It uses special pens that are moved around to create the desired drawing (see Figure 2.10). It operates by interpreting commands which move the pen across the paper in both the vertical and horizontal directions.

Figure 2.10 Plotters use pens to create accurate drawings.

Other output devices

Other output devices include speakers for the output of sound, and MIDI interfaces to output to electronic musical instruments and directly to video or television. Data can also be output to another computer by means of disks, modems and network interfaces.

Exercise 2.3

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

VDUs and _____ examples of _____ copy _____s the copy cannot be used _____ from the comute. Printers produce a _____ copy that can be used away fro the _____. The main ypes of printers are _____ printers, _____ printers and _____ printers. A plotter is also a _____ copy device but it uses _____ to draw on the _____.

- 2 For each statement, select a matching word or phrase from the following list: CRT, dot-matrix, hard copy, ink-jet, input, LCD, monitor, monochrome, output, printer, VDU.
- a A television-like device, used to display images generated by the computer.
 - b Converts data in the form of electric signals to impressions on paper.
 - c Abbreviation for cathode ray tube.
 - d A type of monitor that displays in one colour.
 - e A popular non-impact printer.
 - f Another name for a printout.
 - g Screen used by many portable computers.
 - h Presentation of data or information to a person or another computer.
 - i Abbreviation for visual display unit.
 - j Entry of data into the system for processing.
 - k An impact printer that makes an exact copy of the original.
- 3 What is the term given to the presentation of information to a person or data to another computer?
- 4 Create a database with three fields DEVICE, TYPE and ADVANTAGES. Enter each of the output devices into this database and classify its type as soft copy or hard copy. List the advantages that each type of output has over the others. Save your file as OUTDEVIC.
- 5 Find and describe five examples of where a display is better than a hard copy output. Explain why the display is better for this application. Use a word processor to answer this question, saving your answer with the filename DISPLAYS.

EXTENSION

- 6 Although futurist writers predict a paperless office, high-speed printers continue to print tonnes of paper every hour. How can a business reduce the amount of paper it is using?
- 7 The resolution of computer screens has increased in the past few years. Compare the resolution of a screen with that of a printout obtained from a laser printer.

Process and control

The **central processing unit (CPU)** is responsible for controlling and processing data within the computer. It is the 'brains' of the computer. The CPU accepts the data from any input device, changes this data according to the instructions given by the operator, and then sends the results to an output device. These results are the information required to solve the problem.

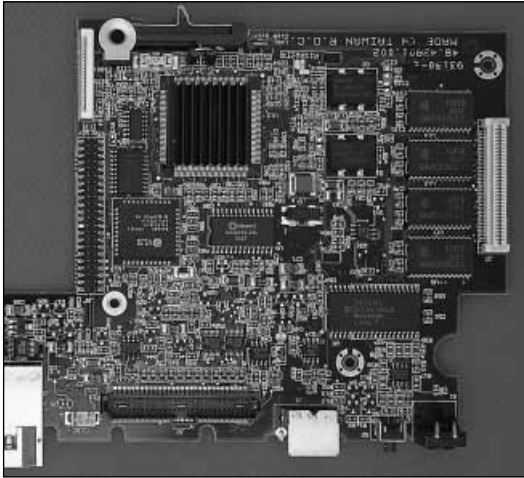


Figure 2.11 The main board (also called logic board or motherboard) of a computer showing the silicon chips that contain the electronic circuits needed to work the computer.

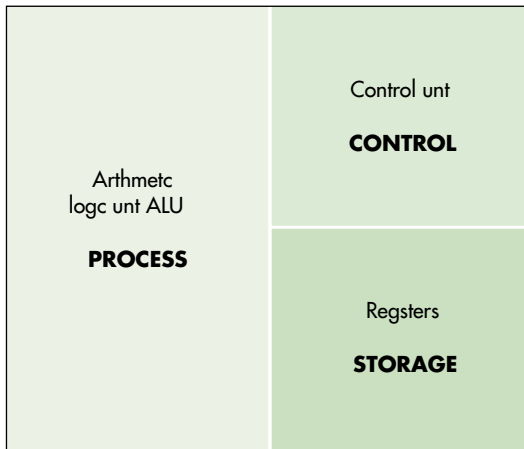


Figure 2.12 The central processing unit (CPU) is made up of three parts.

Arithmetic logic unit

The arithmetic logic unit (ALU) is the part of the CPU that does all the mathematical and logical calculations. It performs:

- basic mathematical operations, including addition, subtraction, multiplication and division
- comparison operations to make decisions using relational operators (=, <, >, <=, >=, and <=)
- logical operations to combine decisions using the logical operators (and, or, not).

The CPU consists of millions of electrical components located on a thin silicon wafer called an **integrated circuit (IC)** or **silicon chip**. In a microcomputer the CPU is contained on one integrated circuit and is called a **microprocessor**. In larger computer systems the CPU may consist of more than one integrated circuit. These integrated circuits are located on the flat printed circuit board inside the computer called the **main board**, **logic board** or **motherboard** (see Figure 2.11). The main board also contains other integrated circuits to store data.

With advances in technology, microprocessors are becoming more powerful and cheaper. This has allowed the microcomputer to do tasks that once required a computer the size of a classroom. The CPU is made up of the **control unit**, the **arithmetic logic unit** and **registers** (see Figure 2.12).

The control unit coordinates the operations of input, processing, output and storage. It is the organiser that directs the flow of data in the computer in the same way as traffic lights control the flow of vehicles at an intersection. The control unit selects and retrieves instructions from storage in sequence, interprets them, and starts the required operation.

Control unit

The control unit coordinates the operations of input, processing, output and storage. It is the organiser that directs the flow of data in the computer in the same way as traffic lights control the flow of vehicles at an intersection. The control unit selects and retrieves instructions from storage in sequence, interprets them, and starts the required operation.

Registers

A register is a temporary storage area for small amounts of data or instructions needed for processing. Registers are located within the CPU to provide faster access to data than primary storage. Some of the different registers are:

- the **accumulator**, for storing the result of the latest calculation
- the **storage register**, for storing data coming from, or being sent from, primary storage
- the **address register**, for storing information such as a number about the location of data in primary storage (This information is called the **address** of the data.)
- the **instruction register**, for storing the instruction or **operation code (op code)** to be followed
- other registers used to store data that can be used for comparisons, for example a **flag register** where a 1 is stored in a specified place in the register when an operation gives a particular result.

Exercise 2.4

- 1 Copy and complete the following sentences:
 - a The CPU stands for the _____ processing unit.
 - b The CPU accepts data from an _____ device.
 - c The CPU sends the results to an _____ device.
 - d An _____ circuit consists of a silicon wafer containing millions of electrical components.
 - e In a microcomputer, the CPU contained on one integrated circuit is referred to as a _____.
 - f The integrated circuit inside a computer is called the _____.
 - g The _____ is the device that directs the flow of data in the computer.
 - h The part of the CPU that carries out all the mathematical and logical calculations is called the _____.
 - i A register is a temporary _____ area for small amounts of data and instructions.
 - j A _____ regulates the speed of tasks in the fetch-execute cycle.
- 2 Describe the purpose of the CPU.
- 3 What effects have advances in technology had on microprocessors during the past ten years?
- 4 Name the three parts of the CPU and describe the purpose of each of the parts. Use a word processor to present your answer. Save the file as CPU.
- 5 If the speed of a CPU is given as 500 MHz, what does this mean?

EXTENSION

- 6 'A computer is more intelligent than any person because the CPU can carry out arithmetic operations many times faster'. Comment on this statement.
- 7 In 1959 Jack Kilby and Robert Noyce developed the first integrated circuit (IC). The integrated circuit is now the heart of the computer and has many applications. What precautions are taken during the manufacture of integrated circuits?

Storage

Primary storage

Primary storage is a part of the computer that holds data and programs before and after they have been processed by the CPU. Primary storage is directly accessible by the CPU. It is also known as **main memory**, **primary memory**, **main storage** or simply **memory**.

The amount of primary storage is very important in determining the capabilities of a computer. Computers with more primary storage can store more data and larger programs. Since many software programs require a specific amount of memory before they can be used, a computer with more primary storage can also use more powerful software. The unit of measurement of storage is a **byte** and it represents a single character such as a letter, a number, a punctuation mark or a space. The prefixes 'kilo', 'mega', 'giga', and 'tera' are then added to 'byte' and these words are more commonly used to measure data storage (see Table 2.1). Most microcomputers bought today have a memory measured in megabytes. Many computers have provision, if necessary, for either adding individual integrated circuits to the main board or adding expansion cards to increase their storage capabilities.

Sometimes there is not enough primary storage available in the computer, so **virtual memory** is used. Virtual memory uses a secondary storage device to simulate extra primary storage. Using virtual memory is slower than using main memory by itself, as large amounts of data need to be transferred between primary and secondary storage.

Unit	Symbol	Meaning	Value
byte	B		1
kilobyte	kB	thousand bytes	2^{10} Approx. 1 000
megabyte	MB	million bytes	2^{20} Approx. 1 000 000
gigabyte	GB	billion bytes	2^{30} Approx. 1 000 000 000
terabyte	TB	trillion bytes	2^{40} Approx. 1 000 000 000 000

Table 2.1 Units of measurements for data storage.

There are two main types of primary storage—permanent memory, called **read-only memory (ROM)**, and temporary memory, called **random access memory (RAM)**.

Read-only memory (ROM)

ROM holds data and instructions which are fixed at the time of production and cannot be changed by the operator or the computer. Clearly, permanent memory only allows data to be retrieved (read) and not entered into storage. The software that is stored in ROM is called **firmware**. Storage of software within ROM protects it from being damaged or changed. The firmware often contains part of the operating system of the computer so that the computer, or even software such as a word processor, can be started. The actual contents of ROM are set by the computer manufacturer. ROM is non-volatile. **Non-volatile memory** does not lose its contents when the power to the computer is turned off. There are different types of silicon chips used for ROM. **Programmable ROM (PROM)** chips allow data and instructions to be entered only once and cannot be reprogrammed. **Erasable programmable ROM (EPROM)** chips, as the name suggests, can be erased and reprogrammed by the computer manufacturer.

Random access memory (RAM)

RAM is where data and instructions are held temporarily and where they can be manipulated or executed. This type of memory allows us to read and write data. RAM depends on a supply of electricity to maintain data storage. When the power to the computer is shut off, everything in RAM is lost. In other words, RAM is **volatile memory**. Different types of silicon chips are used for RAM. **Dynamic RAM (DRAM)** chips are commonly used in microcomputers as they are small and relatively simple. **Static RAM (SRAM)** chips have faster access time than DRAM chips but require additional power which is usually supplied by a small battery. SRAM chips are used in small portable computers. A recently developed memory type is **ferroelectric RAM (FRAM)**. FRAM is non-volatile memory with fast access time that could replace the need for ROM and some secondary storage devices such as hard disks.

Cache memory

Cache memory is another type of primary storage used in many computers. It is located between the CPU and RAM and is used to speed up the access to program instructions and data.

Exercise 2.5

- Copy and complete the following sentences:
 - Primary storage is internal storage, as it is _____ the computer's CPU.
 - _____ memory is another name for primary storage.
 - Computers with more primary storage can store more _____.
 - The unit of measurement of storage is a _____.
 - Most microcomputers have their memory size measured in _____.
 - Expansion _____ can be added to some computers to increase their storage capabilities.
 - Primary storage that is permanent memory is called _____.
 - Primary storage that is temporary memory is called _____.
 - The actual contents of ROM are set by the computer _____.
 - _____ memory is used to speed up the access to program instructions and data.
- Describe the purpose of primary storage and explain why a program needs to use primary storage.
- Describe why the amount of primary storage is important in determining the capabilities of a computer.
- Convert the following measurements to the units indicated:
 - 4 MB = _____ B.
 - 6 GB = _____ B.
 - 140 kB = _____ B.
 - 20 MB = _____ B.
 - 40 MB = _____ kB.
 - 2000 MB = _____ GB.
 - 3 000 000 B = _____ MB.
 - 7000 kB = _____ MB.

EXTENSION

- 5 The storage capacities for microcomputers have increased rapidly in the past few years. What do you predict will be the primary storage capacity for a popular personal computer in three years' time? Explain your reasons. Use an appropriate computer program to present your answer as a file called STORAGE1.

Secondary storage

Secondary storage is a more permanent storage area than RAM, using a peripheral device such as a hard drive or a floppy disk. The majority of computers use secondary storage because primary storage is limited in size and its RAM is volatile. Secondary storage is also called **external storage**, as it stores data away from the computer's main board. Secondary storage media include magnetic tapes, magnetic disks and CD-ROM disks.

Magnetic tape

Magnetic tape is a very long, thin strip of plastic, coated with a thin layer of magnetic material. Data is stored on the tapes in frames, each frame consisting of one byte. The data is 'written' onto the tape by a read/write head which converts electrical impulses into magnetic impulses that change the direction of magnetism of the coating on the tape.

The main disadvantage with magnetic tape is that it uses **sequential access** to retrieve data. This form of access starts at the beginning of the tape and reads all the data until the required item is found. Sequential access to data is slow, making magnetic tapes unsuitable for data which is often revised or updated. Magnetic tapes are made in reel-to-reel, cassette and cartridge forms. Each form holds different amounts of data and accesses the data at different rates.



Figure 2.13 Magnetic tape drives are usually used for backup.

Magnetic disks

A magnetic disk consists of a circular piece of metal or plastic whose surface has been coated with a thin layer of magnetic material. Data is written and read using the same principle as for magnetic tape, except that the medium is a disk rather than a long strip of plastic. The disk is housed in a protective jacket or container, since a fingerprint, a spot of dust or a smoke particle can damage the disk and prevent access to the data. **Floppy disks** and **hard disks** are two types of magnetic disks.

Magnetic disks use **direct access** (or **random access**) to retrieve data. This form of access allows data to be found directly without accessing all the previous data. It follows that direct access allows data to be retrieved much faster than the sequential access used on magnetic tapes. In addition to attaining faster data retrieval, magnetic disks can hold more data in a smaller space.

A magnetic disk cannot be used until it has been formatted. **Formatting**, or **initialising**, prepares a disk to store data and organises the disk into concentric circles called **tracks** and pie-shaped wedges called **sectors**. The operating system determines the number of tracks and sectors. It labels each sector of each track with an address, so that the computer can go directly to a specific area (direct access). If a disk is formatted by one operating system, a different operating system may not be able to read its data.

Floppy disks may be single density, double density or high density. The higher the **density**, the greater the disk's data storage. Double density floppy disks can store about 720 kB of data, while high density floppy disks can store about 1.44 MB of data.

Floppy disks

A floppy disk is a magnetic disk made of flexible plastic and covered with magnetic material. Floppy disks cost only a few dollars, with the most common sizes being 3.5 inches (9 cm) and 5.25 inches (13 cm). These sizes measure the diameter of the disk, and the use of the unit 'inch' comes from the American influence on computer development. The 5.25 inch disks are flexible, as their name suggests, but the 3.5 inch disks are housed in a hard plastic jacket. To be used, a floppy disk must be inserted into the disk drive, which is either built into the computer or is in an external unit connected to the computer by a cable. The disk drive spins the disk at a constant speed and data is stored on, or retrieved from, tracks located on the surface of the disk. Most floppy disks in common use today can store between 140 kB and 1.44 MB of data, depending on the type of computer and disk drive.

Hard disks

A hard disk is a magnetic disk made of metal and covered with magnetic material (see Figure 2.14). It is located inside the computer's casing or in a sealed unit, and works on the same principle as a floppy disk, except that it is rigid and much thicker. The rigid construction of a hard disk allows it to be rotated faster than a floppy disk, giving faster access to data. Another advantage of its hard construction is that it permits data to be stored more densely. For example, hard disks attached to a microcomputer can store between 2 GB and 20 GB of data, which is much more than a floppy disk can.

Hard disk drives are available for all sizes of computers. The disk may be permanently installed in the drive, when it is called a **fixed disk**, or it may be in the form of a removable cartridge or disk pack that can be removed from the drive.

A fixed disk is enclosed permanently inside the sealed case for protection from the elements. Fixed-disk systems contain one or more hard disks and can be used on all types of computers. In large computers the fixed-disk system provides storage capacities in the gigabyte (billions of bytes) range.

A removable cartridge, for example a **Zip Disk** or a **Super Disk**, has a similar speed and capacity to a hard disk. These cartridges usually contain one or two disks. Many hard disks used with small computers are designed to use removable



Figure 2.14 A hard disk drive is still a peripheral even though it may be housed inside the main system box of a computer.

cartridges. The advantage of the removable cartridge is that it can be removed at any time, and a different cartridge inserted. For example, a separate cartridge can be used for a different application or to transfer data from one place to another.

A **disk pack** is another removable device in which several hard disks (a common number is eleven) are packed into a single plastic case. The disk pack drives are designed for large systems that require large storage capacities of hundreds of megabytes.

Compact disks

A **CD-ROM (compact disk read-only memory)** disk is an optical laser disk which stores digital data by using laser beams (see Figure 2.15). It is based on the technology of the CD audio disk. This laser technology provides very large storage capacities, and a CD-ROM disk is able to store up to 600 MB of data. Since you cannot write to a CD, it is not suited to applications where data changes, but it is very convenient for storing data that remains constant. In particular CDs are used for encyclopedias, reference material, educational titles, children's stories and games. They are also popular for multimedia applications to store video and audio data.

CD-R (compact disk recordable) technology allows data to be recorded once. With the appropriate software and hardware, audio, video and computer data can be recorded on each CD-R. **CD-RW (compact disk read write)** technology allows a CD-RW disk to be rewritten.

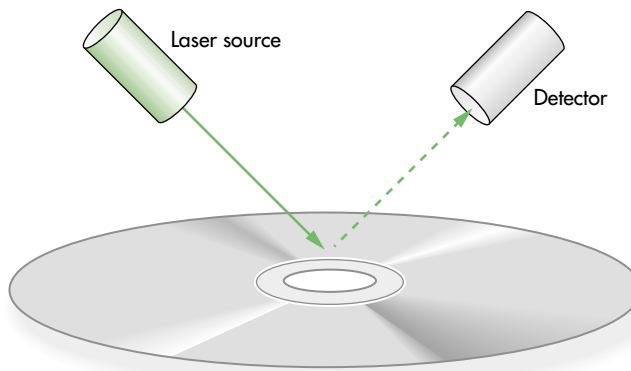


Figure 2.15 Compact disks work by using laser light reflected from the surface of the disk onto a detector.

Exercise 2.6

- Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

_____ storage is less volatile storage than _____ different methods of storage are used, some of them being _____, _____ and _____. Magnetic _____ ribbon coated with a _____ substance. When data is to be stored, the _____ changes the _____ pattern on the tape _____ and _____ disks also use a _____ coating. When disks are _____ they are made ready for use. CD-R disks can only be written to once and are read by means of a _____ beam.

- 2 Explain why it is important for computer systems to use secondary storage devices as well as primary storage.
- 3 Describe the similarities and differences between the three most common secondary storage devices. Choose three uses of secondary storage, one for each of the devices. Explain why that type of secondary storage is better for that application than the other two.

EXTENSION

- 4 A large primary storage enables the computer to retrieve and store data quickly and thus operate at high speed. However, primary storage is more expensive than secondary storage. What factors need to be considered when deciding on the amount of primary storage to be purchased?
- 5 'Laser technology will replace magnetic tapes and disks as the most popular secondary storage medium.' Comment on this statement.

Current trends in hardware

It is very difficult to make firm predictions about where technology will be in even three years' time. We can, however, look at trends and make some educated guesses. A great deal of time, money and effort are continually being expended in order to improve all aspects of computing. Technologies have physical limitations which researchers are constantly seeking to overcome by the invention of new technologies. For example, modem speeds over copper wire telephone lines are limited to 55 600 bps with the current technology.

Hardware technology is focused on improving processing speed, increasing storage and providing a better human–computer interface. Processing speed is improved by increasing the clock speed of the computer and increasing the number of bits processed at the one time (the **word length**). Increased storage allows a computer to manipulate data from more complex systems, for example real-time audio and video, as these types of data carry a large amount of information. Equally important in the storage of large data files is the need to rapidly move the files from secondary storage into primary storage. In order to provide a more friendly working environment for users, the interface with the computer is undergoing changes, especially with regard to the use of non-keyboard input devices such as a microphone.

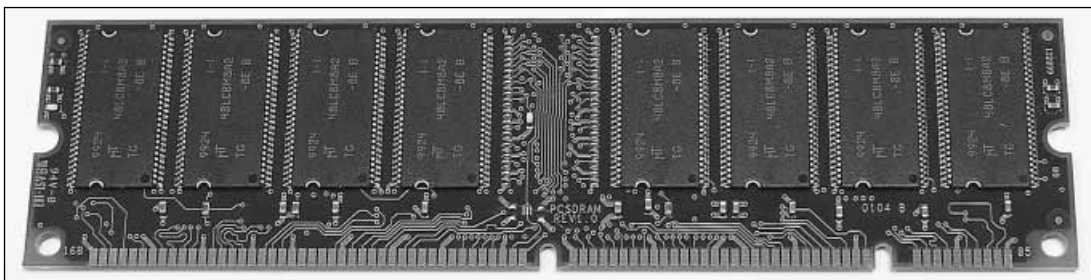


Figure 2.16 Computer speed and power have been increased by the design of integrated circuits that contain a larger number of electronic components, such as these that make up this memory module.

Exercise 2.7

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
Computer technology is continually being _____, which enables in _____, _____ allow the computer to process more complex _____ such as sound and _____. The _____ between the computer and human beings is also being improved. _____ voice recognition software now allows the _____ to communicate with the computer by its _____, _____ also let us view _____ in _____ time.
- 2 Use a word processor to compare the speed, storage and types of peripheral in use five years ago with those available today. Describe the main improvements during this time. Using the information you have gathered in this activity, make predictions about the future improvements in computer technology. Save your document as FUTURE.
- 3 Use newspaper articles, magazines and the Internet to research current trends in hardware. Internet sites that will prove useful are those run by hardware manufacturers. Don't concentrate just on the trends in computers, but also look into peripherals such as printers and cameras. Your research should be summarised as a word-processed document named TRENDS.

Software

Society's acceptance of computer technology is due to its adaptability to a very wide range of tasks. For example, the same hardware item can be used to monitor the security of a factory at night and prepare the week's payroll the next day. This adaptability is made possible by software.

Software can be classified as **system software** or **application software**. System software provides the algorithms for the computer to manage its resources and to communicate with peripherals. Application software is responsible for the instructions which allow the computer to perform a particular task.

System software

System software falls into two groups: the **operating system** and **utility software**. The operating system provides the computer with programs that allow it to communicate with the outside world and manage its resources. Utility software allows the user to perform common tasks such as formatting disks, deleting files and searching through files.

The main tasks of the operating system are in the management of resources and to isolate the user from the direct use of hardware items. The operating system's task starts as soon as the computer is turned on and does not finish until it has been turned off. At startup, the computer needs a program to follow so that it is ready for the user. This program is provided by the operating system, which ensures that the input and output devices are made ready for use and the primary memory is set up to receive to receive programs and data. The user sees the computer go through a predetermined set of steps which will leave it in a

predictable state of readiness for the user to perform whatever tasks are required.

While the computer is on, the operating system usually provides the link between the application program and the input and output devices. When programs are written, input and output are usually described in general terms, for example PRINT 'Hell. It is up to the operating system to communicate at a machine-to-machine level with the required peripheral. In the case of printing to a printer, the operating system will communicate with the printer's **driver** (the program which controls the operation of the printer). This allows the same application program to be used on computers with different printers.

Some utilities are often included within modern operating systems, for example utilities to copy files, delete files and transfer files from one directory to another. Other utilities may be added to the operating system to perform less common tasks, such as virus checking or file recovery.

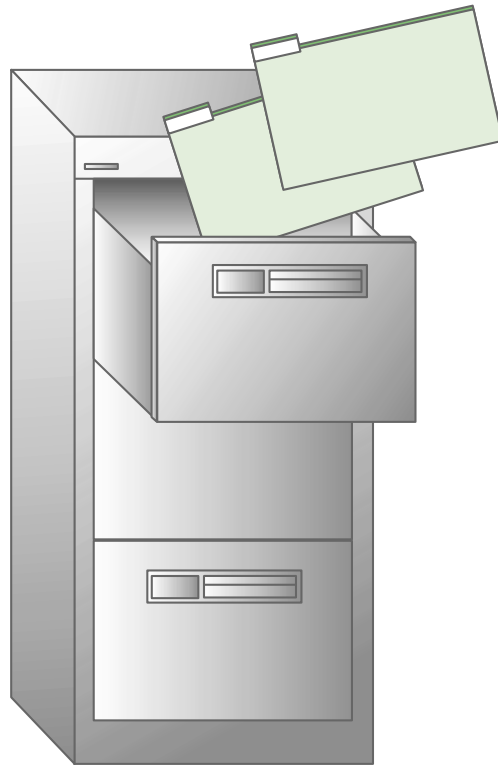


Figure 2.17 System software manages the computer and its resources.

Application software

System software is responsible for the operation of the computer hardware and the general interface with the outside world. The second class of software, application software, is responsible for the computer being able to perform specific tasks. If we look at a factory as an analogy, the operating system corresponds roughly to management and the application software to the workers who actually produce the items made by the factory. Application software can be classed as mass-produced software or custom software.

Mass-produced programs are designed for general use. They are written with a specific task in mind but also with a wide group of users in mind. Application software of this type includes word processors, spreadsheets, database management systems, games, and graphics and multimedia applications. Some application packages allow the program to be customised for a particular user. The user is able to set personal preferences for the way in which the program works or to create a personal set of templates that are appropriate for their own use. This type of software, although very expensive to produce, is relatively cheap to purchase, since the cost of development is shared among the purchasers of the program.

Custom software, on the other hand, is developed for a single customer. This type of software is specialised for the task the customer requires. In some applications, for example an airline's booking system, this leads to a large team being employed over a long period of time. However, since the application has a specific purpose, the customer is forced to meet the total cost of development.

Exercise 2.8

- 1 Note down the tasks that your school computer performs as it starts up. Use a word processor to list these tasks. Save your file as STARTUP.
- 2 When your school computer is running, the operating system 'looks at' peripheral devices waiting for data to be entered. Describe how this might take place. Present your work as a word-processed document with a filename WAITING.
- 3 Run an application such as a spreadsheet and list the tasks you see being performed by the computer. Some of these are the responsibility of the operating system and some are the responsibility of the spreadsheet program. Write down on your list whether each task is carried out by the operating system or the application program. Save the word-processed document as WORKING.
- 4 To your document WORKING, add the tasks the program uses as you program the spreadsheet to display the numbers 1 to 100 in the cells A1 to A100 by using the formula =A1+1 in cell A2 then filling down in the spreadsheet. You can then save the spreadsheet as COUNTING, and again note down the tasks that are performed as the program saves to disk.
- 5 Use a database program to create a database of all the applications you can find on your school computer. This database should name the application in one field and its purpose in a second. Save your database with the filename PROGRAMS.

Generations of programming languages

The modern use of computers in so many diverse fields stems from their adaptability. Before computers were invented different jobs required different devices—many tasks could not be performed by a single device. What makes the computer unique is its ability to be programmed with a set of instructions which can be stored and then reused many times, often without human intervention. The same device can be used to carry out different tasks by changing its set of instructions or program. This program needs to be in a form, or language, that the device can use. However, people create computer programs, and as there is not a common language which is easily understandable by people and computers, an intermediate language, or **programming language**, is needed.

The development of computer programming languages closely follows the development of computer hardware. As the capabilities of computers increased, so did the need for more sophisticated programming methods.

Computer languages can be categorised in five generations. First- and second-generation languages are known as **low-level languages** and are processor-dependent; that is, they are used to develop programs which are specific to a particular type or series of processor. Third-generation and later languages are known as **high-level languages** and are processor-independent. They can resemble either natural languages such as English or symbolic languages such as mathematics. Languages of the third generation and later are used to develop programs in terms of the problem being solved rather than the hardware on which the solution is implemented.

First-generation languages were, by necessity, those that could be directly 'understood' by computers—they were in **binary form**. Early computers were programmed by using paper tape with holes punched to represent 0s and 1s or they had their instructions wired by means of plugboards, wires and switches.

Both methods presented problems when a new program was required and a high level of skill was needed to create the programs. Since the instructions are in binary form, these languages are known as **machine languages** and are specific for the type of computer being programmed.

Second-generation languages or **symbolic assembly languages** replaced the sequences of binary digits with **mnemonic codes** (or short code words) to represent instructions. Like machine languages, assembly languages are specific for the type of processor. However, they offer great advantages over machine code as the mnemonics are easier to remember and read. Their development meant that programming accuracy was improved, since the instructions could be coded using normal written characters. Furthermore, a program did not have to be rewritten if the physical location of a variable or instruction needed to be changed, as the memory locations used to store values are addressed by symbolic names instead of locations. The assembler provides suitable physical memory locations when the program is assembled into machine code for execution. Assembly languages were given names such as AUTOCODER or SAP (symbolic automatic programming).

Third-generation languages provided a great leap forward as they allowed programmers to write programs that were independent of the machine being used or the arrangement of registers and the instruction set of the processor. BASIC, COBOL, ALGOL and FORTRAN are the most widely known third-generation languages. These languages are distinguished from later generations by their structure which consists of a sequence of steps, branches and loops. Unlike second-generation languages where one coded step became one machine instruction, third-generation instructions are usually compiled to several machine instructions.

Fourth-generation languages are more difficult to separate from their third-generation ancestors as they may contain some of the same structures. However, in addition to these structures they employ other mechanisms such as screen interaction, form filling and computer-aided graphics. Many fourth-generation languages depend on a database and its data dictionary, as well as extensions of the dictionary which contain logic and business rules. (This extended form of the data dictionary is often known as an encyclopedia.) Fourth-generation languages are concerned more with what needs to be done than with how it is going to be done, often accomplishing this by forming a software application which can be customised by a user with very little technical expertise. The most common of these languages are sophisticated spreadsheet packages such as Excel and Lotus 1-2-3, and database management systems such as FoxPro, Quattro Pro and Dbase.

Fifth-generation languages allow the programmer to code complex knowledge which the computer can draw inferences from. The languages of this generation use the disciplines of knowledge-based systems, expert systems, inference engines

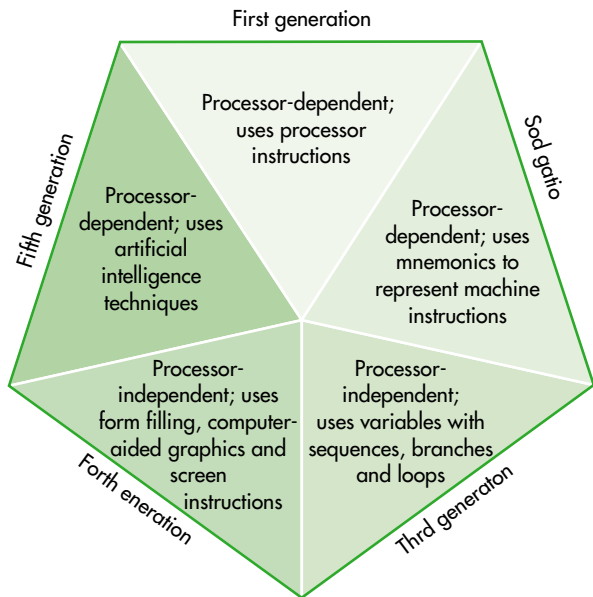


Figure 2.18 The five generations of programming languages.

and language processing originated in the field of artificial intelligence. Programs written in a fifth-generation language often appear to be highly intelligent or to possess an expertise much greater than that of most people. Japan's fifth-generation project, ICOT, is an attempt to use artificial intelligence techniques together with new hardware designs, such as massive parallel processing, to rapidly advance the power of computing technology. This project is important in the development of the fifth generation as there is a coordination of efforts in all research areas, so that new hardware development is made available to the software researchers and vice-versa.

Event-driven versus sequential approach

Different problems require different methods of solution. Some, such as a cooking recipe, can be solved by a chain of steps; others, such as a computer-controlled machine, need to react to different circumstances as they change. Programming languages can cater for either of these methods. Programming which uses the recipe formula is known as **sequential programming**, whereas programming which caters for reactions to different happenings is known as **event-driven programming**.

Sequential programming

Sequential programming is used to create a sequence of steps which, it is hoped, will solve the given problem. A majority of languages in the first, second and third generations support this model. There are several reasons for this, the major one being that the processors supporting these languages were designed to perform one operation at a time, so implementing the translation is a reasonably easy task.

There is a wide range of problems suited to solution by sequential methods, most of which involve the processing of data collected from other sources.

This imperative programming paradigm is easiest to implement on the Von Neumann computer. The basis of the Von Neumann computer is the separation of the processing components from memory. In contrast, the human brain operates in a manner which combines memory and processing. In the Von Neumann computer, the storage of instructions and data are in the same memory locations, and processes within the processor govern whether the value stored in a particular location refers to instructions or data. This architecture lends itself to **imperative languages** which employ variables to model memory locations and assignments which model the processes of the transfer of data from location to location. When **iteration** (the process of repetition of a number of steps) is added to the language structure, a versatile and powerful language is produced.



Figure 2.19 Playing a tape on a VCR is an example of sequential programming.

Event-driven programming

Event-driven programming is well suited to 'real time' applications of computer technology, since this type of programming determines what will happen after a certain external event has been communicated to the system via an interface such as a sensor, mouse, joystick or touch screen.

The main applications, at present, are in the realms of computer-controlled systems and games software, although many general applications can also be written using this programming method. The method parallels the way human beings think, in that we solve problems by reacting to events. For example, if a car driver senses that there is the chance of a collision with an object, the driver will take what actions are necessary to avoid the collision. The driver's mind has been 'programmed', through experience, to deal with this particular event in a certain way.

The most common of the event-driven programming languages available to general computer users are the 'hyperscript'-type languages, one of which is Hypertalk, the scripting language for Hypercard which is available for the Macintosh range of computers. Hypertalk begins an event handler with a statement such as 'on mouseUp' (when the mouse button is released) or 'on openCard' (when a card in the stack is displayed on the screen); this corresponds to an event (something which is sensed by the computer). After this statement comes a series of steps which are executed.

A sample Hypercard script will display the card called 'TitleCard' with a visual effect (in the case of Hypertalk an iris closing) when the mouse button is released. (The reason that the release of the mouse button is used rather than depressing the button is that it allows the user to change his or her mind and not invoke the steps that follow.) The event which triggers the sequence of steps in the program can be changed; for example, the event 'on mouseDown' would start the chain of events when the mouse button is depressed.

```
on mouseUp
  visual effect iris close
  go card 'TitleCard'
end mouseUp
```

Event-driven programming is of particular advantage when there are external devices attached to the computer, each device being able to trigger a subprogram. A word-processing program may be written that uses a keyboard for text entry and a mouse for various editing commands and cursor placement. Some of the events 'captured' by the program could be:

- keypress (depression of one of the keys on the keyboard)
- mouse position (the position of the mouse pointer on the screen)
- mouseclick (one click of the mouse button)
- mousehold (depression of the mouse button and holding it).

Each of these events would then lead to the subprogram which handles what is to happen once the event occurs—including further event-driven programming.



Figure 2.20 The program which runs the VCR is an event-driven program, with the VCR performing a task corresponding to the button or buttons pressed.

Exercise 2.9

- 1 Complete each of the following statements with the most appropriate word from the list (one word is used twice):
event-driven, handler, sequential, variables
 - a The programming method known as _____ programming employs variables and assignments.
 - b A cooking recipe is a form of _____ programming.
 - c _____ programming is appropriate when the computer has to react to some form of happening.
 - d In the Von Neumann computer model, _____ are used to represent memory locations.
 - e When an object receives a message, a _____ processes that message.
- 2 Explain why a majority of first-, second- and third-generation languages are examples of sequential programming languages.
- 3 Explain the three main components of any imperative programming language, and illustrate your answer by choosing examples of each component from a language of your choice.
- 4 Examine the operating instructions for an electronic device such as a video recorder. Choose one section of the instructions which illustrates sequential programming and one which illustrates event-driven programming. Justify your choices.
- 5 Explain why an event-driven system will, with current technology, still require some sequential programming.

The need for translation

Programs written in any of the higher-level languages (second-generation and above) cannot be directly understood by a processor. The instructions contained within the program have to be converted from the human-readable form in that higher-level language (called the **source code**) into a machine-readable form (the **object code**). Source code is usually created using a **text editor** which forms part of a program development system. However, a text file which has been created by other means, for example in a word processor, may also be translated. The process of translation from the source code to the object code is accomplished by using one of three translation methods: compilation, incremental compilation or interpretation.

Compilation, using a compiler, involves translating the whole of the source code into object code and storing the object code to be executed later. Compilation is similar to the translation of a book from one language to another, in which the whole book is translated before being read. A program which has been compiled will be executed quickly, since the computer can understand the instructions directly.

In **interpretation** the source code is translated to object code which is immediately executed. Interpretation can be likened to an interpreter who stands beside a foreign dignitary and translates the sentences into the known language as they are spoken. Interpreters produce code which is immediately executed and therefore has the advantage of being able to identify errors within a statement at

the time of execution. A second advantage of the interpreter over the compiler is that the program can be tested for both syntax and run-time errors. A common use of interpreters was in the early personal computers such as the Commodore 64 and Apple II, which had a BASIC interpreter in ROM. An interpreted program will be executed more slowly than compiled object code since the translation process has to be carried out each time a program is executed. This is most evident where there are a number of loops within the program. A compromise is to use an interpreter during the development stage of a program and compile the resulting source file when all errors have been identified and corrected.

In **incremental compilation**, an interpreter is used to translate the high-level code. Sections of code which are used repeatedly are translated and stored for use by the interpreter when needed. Use of this method means that the interpreter does not have to repeat the translation process for these sections of code, thus speeding up execution of the program.

The language translator is, itself, a computer program. Input for this program is the source code which consists of high-level language that has been entered as text. Output from the translation system is the executable object code. The language structure determines the processes used within the translator.

Characteristics of different operating systems

As we have seen, the operating system is responsible for communication between the user and the computer system. Two types of operating system are in common use: the **command-based interface** and the **graphical user interface (GUI)**.

The command-based, or command-line, interface relies on the use of text to communicate with the computer. The basis of the command-based interface is the use of special words or phrases to initiate an action by the computer. In a command-based interface, the display consists of a predetermined number of lines of text of a fixed length. (A common standard is to have 24 lines each of 80 characters.) Text is stored in a part of main memory called a **screen buffer**. Each of the character positions on the screen is represented by one memory location in the screen buffer. The values in this screen buffer, usually conforming to the ASCII set of characters, are translated into displayable characters by a character generator. The command-based interface may use one or more lines on the screen to accept input text and may also use menus and keyboard combinations for frequently used operations. Text displayed in this way is very limited as there is little that can be done to a character by way of emphasis. Early computers used this kind of interface, as both processing power and mass storage were expensive. There are still peripheral devices, such as automatic teller machines (ATMs) with LED (light-emitting diode) displays, which can only display characters.

The graphical user interface uses **windows**, **icons**, a **mouse** (or other pointing devices such as a trackpad) and **pointers** (thus it may also be referred to as a **WIMP interface**) to present information on the screen and make selections. The graphical user interface uses a greater amount of main memory as a screen buffer, as each of the memory locations represents one **pixel** (dot) on the screen. This type of user interface became prominent only when the cost of processors decreased and their processing speed increased. The graphical user interface uses the concept of having items on a desktop. Items are usually represented by icons and may be selected by using the mouse to point and click. Choices within the program may also be presented graphically by means of palettes and choice boxes as well as by menus. The GUI is also able to display documents in a form which closely resembles their final form. This form of display is known as **WYSIWYG** (what you see is what you get).

Current trends in the development of software and operating systems

The development of software closely parallels the developments in hardware. As processors become more powerful, these capabilities can be applied to software. Currently there are two main areas of interest for software developers—those focusing on the tasks performed by an individual and those focusing on tasks performed as a group.

For the individual, the most exciting development is in the area of voice recognition. Many potential users are put off using computers as they find the keyboard intimidating. Development of voice recognition in both application software and operating system software is well under way and it should not be too long before we see systems which, once trained, will perform most tasks with little or no keyboard input. Further development in multimedia will also see the computer and entertainment systems become integrated, with the television becoming not just a passive device presenting to the viewer but an interactive device able to perform many tasks now done by a computer.

Collaborative working (i.e. a group of people working on a single task) also presents an exciting future. The problems involved with collaborative working on a single file are being addressed. Teleconferencing by computer is also well developed and further development will see it integrated into group work on a file.

Exercise 2.10

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Programs written in a _____ language cannot be understood by a _____ unless it has been translated. High-level code is called the _____ and the machine-readable form is known as _____. There are three methods of translation: _____, _____ and _____. The process of _____ translates the whole of the _____ level code into _____ before it can be run. An _____ section of the code and the resulting _____ is immediately _____.

- 2 Describe the similarities and differences between the three translation methods.
- 3 Examine the operating system on your school computer and explain whether it is command-based or graphical. Give reasons for your choice. Present your answer as a word-processed document called SCHOOLOS.
- 4 Use the ASCII table on page 105 of the book to convert the phrase *Software Development* into a string of decimal ASCII values. Don't forget the difference between the uppercase and lowercase letters in the name, as well as the space. Use a spreadsheet to copy each of these values into cells A2 down to A21. In cell B2 type the formula =CHAR(A2) and fill down to cell B21. If your codes are right, you should see the words *Software Development* written down column B. Save your document as ASCIIONE.
- 5 Convert the following set of ASCII values in decimal into characters:
3, 32, 108, 105, 107, 101, 32, 86, 101, 103, 105, 109, 105, 116, 101
- 6 Using a second blank spreadsheet, type your name down column A from cell A2, with one character per cell. In cell B2 type the formula =CODE(A2), then fill it down to the same length as your name. Column B will contain the ASCII codes for

your name. Check the values with the ASCII table on page 105. Save your file as ASCII TWO.

- 7 Describe the differences between command-based and graphical user interfaces. Give one example where a command-based interface would be appropriate and one where a graphical user interface is appropriate. Give reasons for your choice.
- 8 Examine two application packages which perform the same task, for example two word processors, one current product and one from five to ten years ago. Explain the differences between the applications. Using these applications as examples of the evolution of software, describe the changes in word processing you might expect within the next ten years. Present your answer as a word-processed document called FUTUREWP.

The relationship between hardware and software

Software provides the instructions for the hardware to follow. Program instructions need to be performed in the designed order if the computer is to achieve the desired results. The control part of the processor ensures that the instructions are followed in the correct order. It is helped in this task by a special register, often known as a **program counter**, which stores the memory location of the next instruction to be followed. As each instruction is performed, the processor passes through what is known as the **fetch–execute cycle**.

Processing of software instructions by hardware: the ‘fetch–execute’ cycle

To process data, the CPU performs a cycle of events on a single instruction. This cycle is called the fetch–execute cycle, or **machine cycle**. The fetch–execute cycle, as the name suggests, has two parts: the fetch cycle (or **instruction cycle**) and the execution cycle. The fetch cycle involves the control unit getting the instruction (fetch) and working out what to do (decode). The execution cycle involves the ALU carrying out the instruction (execute) and the control unit sending the results to storage (store) (see Figure 2.21).

The fetch–execute cycle can be divided into nine steps:

- fetching the instruction from primary storage
- decoding the instruction into an operation code and data addresses
- copying the operation code into the instruction register
- copying the addresses of the data into the address register
- using the address register to copy the data into the storage register

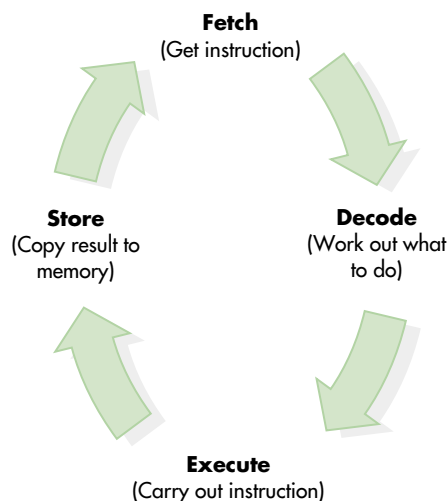


Figure 2.21 The fetch–execute cycle.

- sending the operation code and data to the ALU
- executing the instruction on the data
- sending the result to the accumulator, ready for the next instruction
- storing the results in primary storage.

A system clock regulates the speed of the tasks in the fetch–execute cycle by sending out electrical pulses at a certain rate. The number of electrical pulses per second is the speed of the CPU and is called its **clock speed**. New personal computers have clock speeds between 300 and 500 MHz (megahertz). A speed of 500 megahertz means the clock is generating 500 million electrical pulses per second.

The initiation and running of an application

We can see the fetch–execute cycle in operation if we look at the events that occur when we choose to run an application. As soon as the computer is turned on, the fetch–execute cycle begins. The instructions which load and run the operating system come from hardware (ROM) and from a secondary storage device such as a hard disk. The ROM contains the program that loads the operating system. This program loads the operating system program into main memory and then executes the operating system.

Once the operating system has been loaded, the computer is ready for use. While waiting for the user to choose an application to run, the operating system continually monitors the input devices for actions. When the user has chosen the application to run, the operating system locates the start of the application on the external storage device and begins to load it into main memory. The operating system continues with loading the program into main memory, finding the appropriate sections of the program on the disk as they are needed. When the program has been loaded into main memory, control is passed from the operating system to the now-loaded application. The application usually has to set up memory locations for its own use and perform other **initialisation** activities so that it can run effectively. While this set of steps is being performed, the application will usually display a startup screen to indicate that it is working. Once the startup activities have finished, the program will display a start screen giving the user options to commence work using the application. Finally, the program will again cycle through various input devices, looking for an instruction from the user to perform a task.

The existence of minimum hardware requirements to run some software

Application programs are developed to perform a specific task or group of tasks. The complexity of a program or the task it is designed to perform means that there will often be a minimum set of hardware requirements that need to be met before the application can run successfully.

Many factors might contribute towards running an application, including:

- files requiring a large amount of main memory for their manipulation, for example graphics or sound files
- applications where the speed of processing is critical, for example in the production of full-motion video
- particular requirements of peripherals, for example the need of a sensor for a certain type of interface with the computer

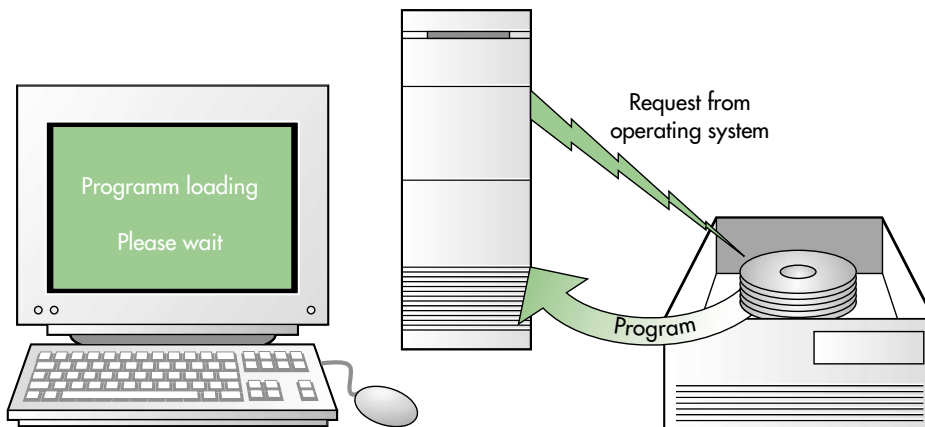


Figure 2.22 Running an application involves loading the program from external storage into memory and starting it.

- processors which can interpret a particular set of instructions, for example when special machine code instructions have been written to perform a specialised task
- external storage devices with a particular capacity to store intermediate results or parts of the program
- particular types of input or output devices, such as touch screens or network interface cards.

Exercise 2.11

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Hardware flows the _____ provided by software. The CPU moves through the steps of the _____, executes the instructions one by one. The first stage in the _____ cycle is for the _____ unit to _____ the instruction. This step is followed by the process of _____, where the _____ unit works out what to do. The last two steps are called the _____ cycle. This cycle begins with the carrying out of an instruction, following which the _____ unit sends the result to _____.

- 2 Use a drawing program to create a series of diagrams that show the fetch–execute cycle in operation on an instruction that takes a value of 3 from memory location 8 and adds it to the total in the accumulator register. Call your file FETCHXE.
- 3 Start the word processor on your computer and examine the visible steps your computer carries out as the file loads. List these steps as a word-processed file called BEGINWP. Compare these steps with the steps in the section above.
- 4 Examine a number of software titles for any minimum hardware requirements. For each of these items, give reasons why you think these restrictions exist. Present your findings as a database file with fields for *software title*, *minimum hardware* and *reasons*.

Elements of a computer system

A computer system can be viewed as containing five elements all working together to achieve a purpose. These five elements are hardware, software, data, procedures and personnel. When designing a software solution to a problem, all of these elements must be kept in mind, as no one of them is any more or less important than the others.

- **Hardware** consists of the physical devices needed to perform the required job. These devices perform the tasks of input, output, process, control and storage.
- **Software** consists of the steps, coded in a way that can be interpreted by a computer, required to perform the task. These instructions may be given to the computer in a machine-understandable form, such as a word-processing program, or in a human-understandable form, such as a program written in a language such as Visual Basic or Pascal.
- **Data** is the raw facts that are manipulated by the computer system in order to present us with information.
- **Procedures** are tasks that are performed by, and rules put in place for, the users of the computer system. These procedures are put in place to ensure that the people using the system perform the correct tasks at the appropriate times with the required data.
- **Personnel** are all the people involved with the computer system, not just the direct users of the computer hardware. These people include:
 - **direct users**—people who use the hardware
 - **indirect users**—people who are affected by the system
 - **computer operators**—people who look after the computer resources in a large computer system such as a mainframe computer
 - **data entry operators**—people who enter data into the terminals
 - **information systems managers**—people who plan and oversee the computing resources within an organisation

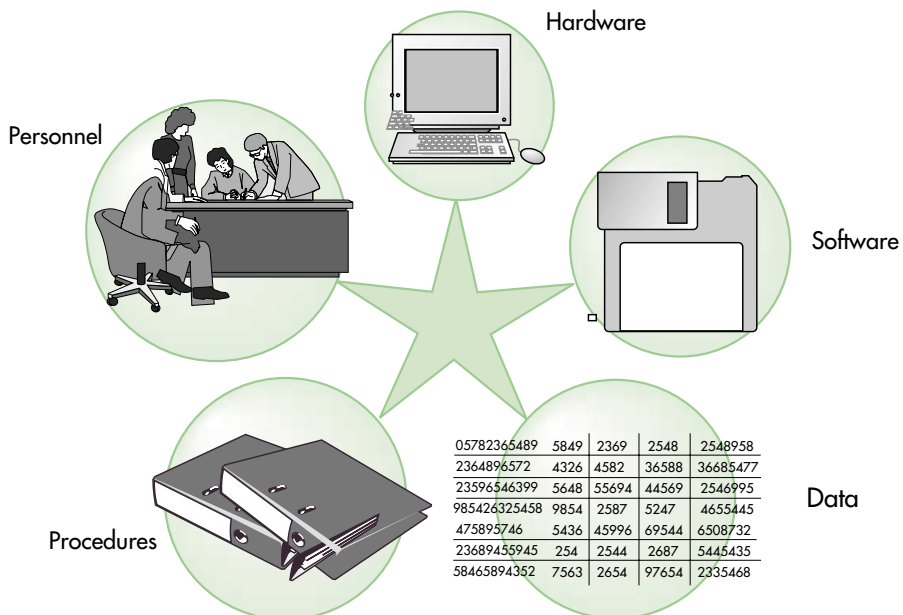


Figure 2.23 A computer system has five elements which work together to achieve its purpose.

- **maintainers**—people who support the computer system by providing technical help in installation, maintenance and repair of computer equipment (**computer technicians**), assistance to users (**technical support staff**), training to users in how the system operates (**training specialists**) and advice on how problems can be solved (**computer consultants**)
- **developers**—people concerned with the design and development of the system, such as **systems analysts**, **computer programmers** and **hardware engineers**.

Case study

Computers in the supermarket

Each of the elements of a computer system are examined in more detail in the following case study.

A supermarket uses a computer system to operate its checkouts and manage its banking. Barcodes are scanned at the checkout, the items are identified from a central database and the prices and item descriptions are printed on the customer's docket. When all the customer's items have been scanned, a total price is calculated and displayed for the checkout operator to see. Once the cash has been tendered by the customer, the change is calculated, and the total is added to the day's takings of the checkout as well as to the store's takings on the central computer. The elements of this system can be classified as follows:

- Hardware consists of the main computer, printer and networking hardware together with a barcode reader, cash register, cash register printer and display for each of the checkouts.
- Software includes the operating system, backup software, network management system for items stocked, accounting software loaded onto the master computer, and cash register software loaded onto the registers.
- Data used by the system includes barcodes, item descriptions, item prices, customer totals and daily totals for the store and for each cash register.
- Procedures performed by the checkout operator include scanning individual items, calculating the total to be paid

by the customer and giving the receipt and change. Procedures carried out by management include banking the day's takings, balancing the cash registers and backing up the data.

- Personnel can be viewed as belonging to the groups outlined above: direct users, indirect users, maintainers and developers. Direct users in this example would include the checkout operators and store managers. Indirect users are the customers. Maintainers include the technical repair staff, trainers and help-desk staff. Developers were involved with the design of the system and would also be needed if, for example, the supermarket wanted to add an EFTPOS ability to each checkout.



Figure 2.24 Cash registers are one of the hardware items in a supermarket system.

Developing software

When software is being developed to solve a problem, each of these system elements has its part to play. Hardware needs to be obtained that will perform the tasks required. The necessary software is either bought off the shelf or developed. Data entering and leaving the computer system is examined to work out the processing and storage needed within the system. Procedures are designed to ensure that the correct data is entered and the appropriate output is obtained from the system. Personnel are involved in all phases of the development of the new system.

Managers and users are observed and questioned to find how the system works and what improvements are wanted. Developers create the new system, with programmers writing or modifying any computer code required. Technicians and trainers are employed to install the new hardware and train users in the operation of the system.

Exercise 2.12

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
A computer _____ consists of five elements. These are _____, _____, _____, _____ and _____. The physical devices are the system's _____. Programs are classified as _____. Raw facts form the _____. Tasks performed are the _____ and people involved with the system are the _____.
- 2 Examine your school library system and identify and list the hardware, software, data, personnel and procedures used for this system. Present your report as a word-processed document called SYSLIB.
- 3 Examine a computer system of your own choice, identifying the elements of that system in the same way as in question 2 above. Present your report as a word-processed document called MYSYSTEM.

Review exercises

- 1 Name the five elements of computer hardware and describe, in your own words, the purpose of each of these elements within the computer system.
- 2 You have been given the task of publishing the school magazine using the computers in the computer room. Describe the input devices you will use for this task and the purpose of each.
- 3 Choose an application of computer technology used in a business and investigate the input and output devices used by that business. Your investigation should focus on the devices used and the purpose for using them. You can publish your report as a word-processed document.
- 4 Explain the tasks of each of the three parts of a CPU when data is being processed. Use diagrams to help with your explanation.
- 5 Describe the similarities and differences between primary and secondary storage. Explain the role of each in the processing of data. Use an example to help with your explanation.
- 6 Describe the reasons why the speed of processors has increased and the amount of primary storage in computer systems has increased. How have these increases been achieved?
- 7 Explain the differences between system and application software. Describe the role of each in the operation of a computer system.
- 8
 - a List the features of each of the generations of programming languages.
 - b Extension: Choose one application where each of these languages would be appropriate and give reasons for your choice.
- 9 Find one example of sequential programming in a device at home and one of event-driven programming. Explain why you chose the devices you did and briefly describe the actions of each program.
- 10 Explain the reasons why code written by a computer programmer needs to be translated for the computer.
- 11 Explain why a word-processing program written for an operating system such as the Macintosh OS will not run on a computer running Windows.
- 12 Choose an application of computer technology in business or commerce. Describe each of the five elements of computer systems as they apply this application. Don't forget any indirect users as well as direct users of the system. You can publish your report as a word-processed document.

Chapter summary

- Hardware consists of input, process, control, storage and output.
- Input involves entering data into the computer for processing.
- Processing consists of the steps required to change input into the required output.
- Control involves the coordination of the processes involved in input, processing and output.
- Storage is the holding of data over a period of time so that it can be used when required.
- Output consists of the presentation of processed data to a person or another computer.
- Peripherals are devices other than the central processing unit (CPU).
- Input devices consist of items such as keyboards, mice, trackballs, trackpads, light pens, touch screens, digitising tablets, character readers, scanners, cameras and microphones.
- Output devices include visual display devices, hard copy devices, speakers and devices that output to another computer.
- Process and control are performed by the CPU.
- The CPU consists of the control unit, the arithmetic logic unit (ALU) and registers.
- The control unit coordinates the processes of input, processing and output.
- The ALU performs the tasks given to it by the instructions.
- Registers are memory locations inside the CPU that are used to store data and instructions that are needed for processing.
- Storage can be classed as primary storage and secondary storage.
- Primary storage is directly accessible by the CPU and is usually in the form of ROM and RAM.
- Secondary storage is more permanent than RAM and it stores data away from the main board.
- Examples of secondary storage devices are magnetic tape, floppy disks, hard disks and CD-ROMs.
- Trends in hardware are focused on increasing power by using faster processors with greater primary storage.
- Software is classified as system software or application software.
- System software consists of the operating system and utilities.
- The operating system allows the computer to communicate with the outside world.

Chapter summary

- Utilities allow the user to perform common tasks such as copying files and formatting disks.
- Application software performs specific tasks such as word processing.
- Mass-produced application software titles are designed for general use.
- Custom software is designed for a particular user.
- Programming languages can be categorised by their generation.
- First-generation languages are machine codes directly understandable by the processor.
- Second-generation languages are assembly languages, where each mnemonic code is translated into one machine instruction.
- Third-generation languages are independent of the processor used and use steps, branches and loops of instructions.
- Fourth-generation languages contain some of the same structures as third-generation languages but use form filling, screen interaction and computer-aided graphics as well.
- Fifth-generation languages employ artificial intelligence techniques to code complex knowledge from which the computer can draw inferences.
- Sequential programming uses instructions arranged as a set of steps that can be followed.
- Event-driven programming uses input from an external device to determine how the system will react.
- High-level languages need to be translated so the processor can understand the instructions.
- The three methods of translation are compilation, interpretation and incremental compilation.
- Compilation is the process in which all the source code is translated into object code for later execution.
- Interpretation involves the source code being translated and executed line by line.
- Incremental compilation uses an interpreter, but frequently used sections of code are compiled and used whenever needed.
- Operating systems can be classed as either command-based or a graphical user interface.
- The command-based interface uses commands in the form of text to communicate with the computer system.
- The graphical user interface uses windows, icons, mice and pointers to make selections and use the system.

Chapter summary

- Software development trends are focusing on tasks performed by the individual and tasks performed by groups.
- Software instructions are performed by the fetch–execute cycle.
- The main steps in the fetch–execute cycle are fetch the instruction, decode it, execute it and store the result.
- When an application is run, the operating system locates the application on disk, loads it into RAM, displays a start screen and then waits for user input.
- Many applications need a minimum hardware setup in order to run.
- The elements of a computer system can be classified as hardware, software, data, procedures and personnel.
- Hardware is the set of physical devices needed.
- Software provides the computer instructions.
- Data is the raw facts needed by the computer for the task.
- Procedures are tasks performed by the users of the system.
- Personnel are all the people involved with, or affected by, the system.

Team Activity

You have been given the task of designing a system to run the school's interactive tour for prospective students. Investigate and list the hardware and software you would need to perform this task. Choose the items from the list that you think are

most appropriate for the task and give reasons for the choice. Include the minimum hardware you will need for this task. Present your report as a word-processed document. Also use a spreadsheet to prepare a costing for these items.

chapter 3

→ *Software development approaches*

Outcomes

- explains the effects of historical developments on current practices (P 2.2)
- identifies the issues relating to the use of software solutions (P 3.1)
- analyses a given problem in order to generate a computer-based solution (P 4.1)
- investigates a structured approach in the design and implementation of a software solution (P 4.2)
- uses a variety of development approaches to generate software solutions and distinguishes between these approaches (P 4.3)
- describes the role of personnel involved in software development (P 6.1)

Students learn about:

The structured approach to software solutions

- program development cycle for the structured approach, including defining the problem, planning, building, checking and modifying
- characteristics of the structured approach, including:
 - long time periods
 - large-scale projects
 - large budgets
- involvement of personnel, including analysts, designers, programmers, users and management
- team approach

The prototyping approach to software solutions

- characteristics of the prototyping approach, including:
 - non-formal
 - shorter time period
 - small-scale projects
 - small budgets
- involvement of personnel, including programmer and users
- links with the structured approach

The rapid applications software development approach

- characteristics of the rapid approach, including:
 - lack of formal stages
 - coding languages used
 - relationship of programmer to end user
 - short time period
 - small-scale projects
 - low budgets
- involvement of personnel, including developer and end user

End-user approach to software development

- characteristics of the end-user approach, including:
 - use of standard software packages
 - lack of formal stages
 - short time period
 - potential long-term, small-scale project
 - low budgets
- end user as the developer

Students learn to:

- identify sound ergonomic practices when using computers
- identify each of these stages in practical programming exercises
- design and develop a limited prototype as a demonstration of a solution to a specified problem
- use an existing software package to develop a customised solution
- select appropriate software development approaches for specific purposes

Personal Profile—Konrad Zuse (1910–1995)

Konrad Zuse was born on 22 June 1910 in Wilmsdorf, a suburb of Berlin. After completing his schooling, Konrad entered the University of Berlin, Charlottenberg, in 1927 to study civil engineering.

On graduation, Konrad joined the Henschel Aircraft Company and was set to work investigating the stresses caused by the vibrations of aircraft wings. This work involved a great deal of calculation. To help with these calculations he designed and built a mechanical computer, named Z1, in his parents' living room. Work on Z1 began in 1934 and finished in 1938.

The beginning of World War II in 1939 brought a temporary halt to Zuse's plans to build a bigger and better machine. He was called up to serve in the infantry, but was eventually able to persuade the army to allow him to go back to building computers.

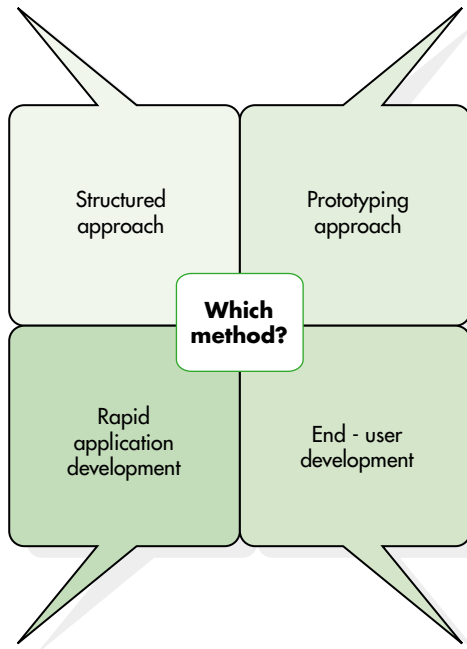
By 1941 Konrad had built two further computers, called Z2 and Z3, for the German Aerodynamic Research Institute. These were electromechanical computers which used relays to perform calculations. Z3 was the first programmed control calculating machine in operation. It was able to store 64 numbers each consisting of 22 bits and to multiply two rows of digits in 3 to 5 seconds. However, his plans for an electronic computer using valves were rejected as not necessary since the authorities believed that Germany would soon win the war.

In 1942 he commenced work on Z4. Towards the end of the war, the Allied bombing of Germany forced Zuse to move the computer which was nearly completed. The first move was to Göttingen, but the threat of the Z4 being captured forced Zuse into making a further move, this time to a small village in Bavaria called Hinterstein. In 1950 Z4 was installed in the ETH in Zurich, Switzerland, where it remained operational until 1955.

In 1950 Konrad started his own computing company, which was taken over by Siemens in 1967. He continued to work for Siemens as a consultant in computer research. In 1966 he was appointed an honorary professor at the University of Göttingen.

Zuse also made a great contribution to software in 1945 by developing the first algorithmic programming language, known as 'Plankalkül'. Using this language, he designed a chess-playing program.





Introduction

There is no one correct way of producing a computer solution to a problem. Many different factors will affect the way in which a software package is developed. These factors will include the time, budget and resources available, the nature of the problem and the expertise of the developer. Four of the more common approaches are the **structured approach**, **prototyping approach**, **rapid application development (RAD)** and **end-user development**.

Figure 3.1 There are four common approaches to software development. Many factors will determine which is used in a particular case.

The structured approach to software solutions

This software development approach is based on a tried and tested design method. It can be divided into five stages—defining the problem, followed by planning, building, checking and modifying the solution. These steps form what is commonly known as the **software lifecycle**. The steps can be thought of as forming the ledges of a waterfall, the flow from one stage to the next following a downhill path. Figure 3.2 shows these stages diagrammatically.

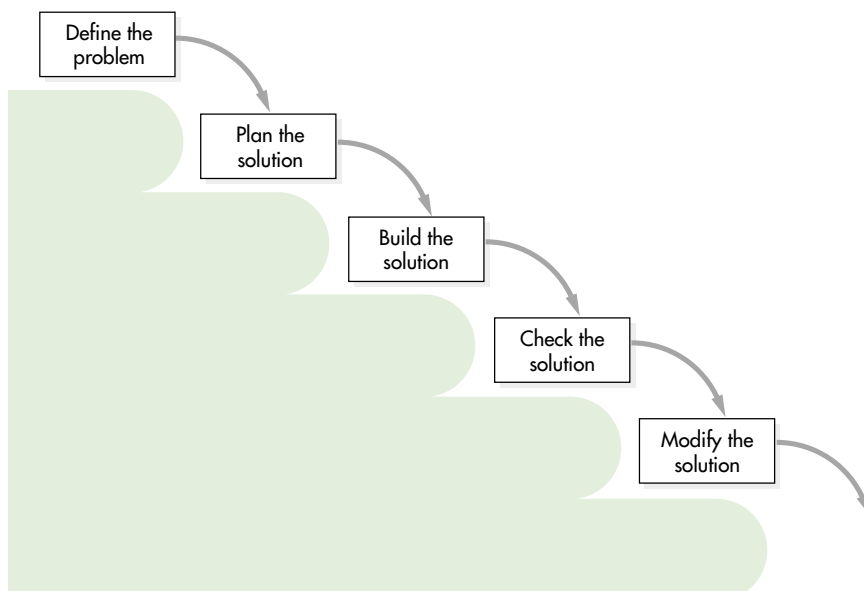


Figure 3.2 The structured approach to program development can be likened to a waterfall.

We will follow these stages in the development of a program for a small coach company called Cumfy Coaches. At the present time, Cumfy Coaches uses a card system to keep a record of the bookings for their trips. This system is slow and inefficient, so the company has decided to use a computer system instead.

Defining the problem

Before any solution to a problem can be created, the problem itself must be understood. At the beginning of the process, effort is put into determining exactly what is wanted from the software. Statements such as 'I want to computerise my office' are not of any use when it comes to solving the problems that exist. In this first stage no thought is given to the computer technology. All effort is put into an examination of the needs of the user.

At this stage in the development cycle, the managers of the business and a representative of the software development team, usually a systems analyst, discuss the nature of the business as well as the problems faced with the current system. The analyst needs to have a clear indication of what is to be done by the software. From these discussions comes a document which clearly states the requirements of the new system.

The requirements should be expressed in terms of what must be done, not how it is to be done. The requirements should also be clear and sufficiently detailed to allow the final solution to be measured against them. Finally, they should be complete, covering all the existing needs as well as any new ones.

Cumfy Coaches—defining the problem

In discussions with Ian, the managing director of Cumfy Coaches, the systems analyst, Kara, has found out the following about the company. Its purpose is to provide tours for a wide range of people within the local community. The company has two coaches which are regularly booked for tours of varying lengths. The current system consists of one card file box for each of the tours being offered. Each card contains the details of one passenger and the seat allocated to that passenger. When a tour has been completed, the cards are filed in an archive, with the file box being reused for a new tour.

Jan has indicated that the system, although it works quite well, has a number of faults:

- With the card file system it takes quite a while to find the details of a passenger, as cards are stored in seat order in the file box.
- There is often duplication of data as passengers from the same family are each allocated a seat and therefore their details are repeated on each card.
- The preparation of passenger lists for each tour is a mundane task and there have been embarrassing mistakes made recently, when cards were filed in the wrong box.



Figure 3.3 Comfy Coaches system will not cope with the arrival of two new coaches.

- The system is completely independent of the accounting system which again leads to duplication of tasks.
- When travel insurance claims are made by returning passengers, it takes staff quite a long time to locate the details in the archive.
- The archive is taking up more and more space as more tours are completed.
- The system will pose further problems when the company takes delivery of two more coaches which it has ordered to cope with increased business.

Kara has therefore set out the following requirements for the replacement system:

- The system will allow new customer details to be added, existing ones to be edited and cancelling customers to be replaced.
- The system will allow tours to be sorted on customer name, tour date, tour destination and seat allocation.
- Once a tour is completed, the customer records will be stored in an archive file for the calendar year in which the tour occurred.
- Family details will be able to be automatically duplicated for members of the same family.
- The system will be able to produce reports, including passenger lists for the drivers, and tour and customer details from archived files.
- The system will be able to manage 500 tours within a 12-month period.
- The software will be able to communicate with the accounting procedures in use.

Exercise 3.1

- 1 Present the set of requirements for the Cumfy Coaches system as a word-processed document. Add any other features that you think may be needed. Create a folder (subdirectory) on your disk called CUMFYCOACH, and save the report in this directory as CUMFYREQ.
- 2 Choose a manual system that you are using at home and list the functions of that system together with the problems that it poses for you. Describe the requirements you have for a replacement system. You should present this exercise as a word-processed document called REQUIRED. Examples of the types of problem that are suitable for this exercise are address books, birthday books, management of a hobby collection such as stamps and the organisation of your photographs.

Planning the solution

Once the requirements of the system are understood, attention passes to the planning of the solution. This phase has to identify the needs of the user. The process is usually carried out by a systems analyst who examines the inputs and outputs of the present system. Processes and procedures of the existing system are also observed in order to determine how the inputs are going to be converted into outputs.

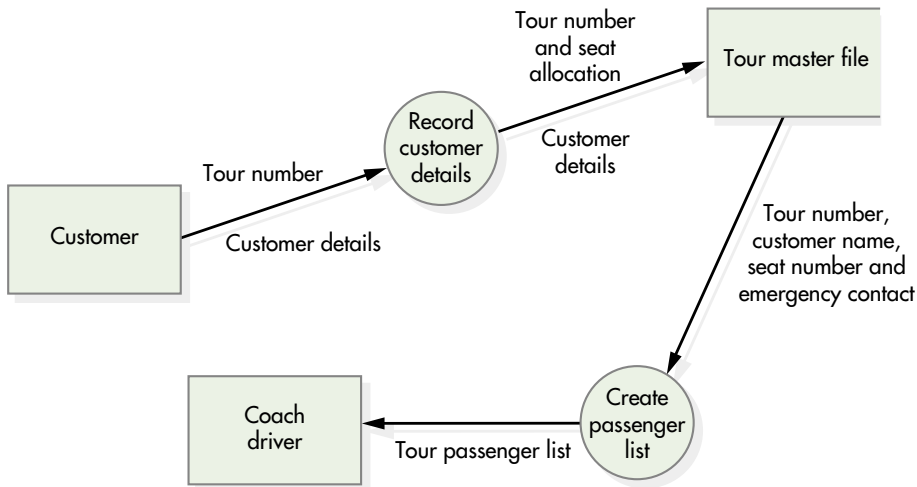


Figure 3.4 An example of a dataflow diagram which is used to show the way in which data passes between parts of the system.

Varying analysis tools are used to help with this information-gathering process. Interviews, questionnaires and observation are used to gather information about the functioning of the current system. The results are put together and presented as a report. The workings of the system may also be presented in a graphical form in this report.

Cumfy Coaches—planning the solution

Now that Kara has a clear set of requirements, it is time to look at the way in which the Cumfy Coaches company operates its business. The first person to provide her with information is the secretary, Janine, whose task is to run the office. Janine’s work involves looking after the booking system, creating the passenger lists and bookkeeping. Next the two drivers, Marie and Angus, provide information about the running of the company. Finally, Kara contacts some of the past passengers to see whether they can provide her with any more information.

In this investigation, Kara concentrates first on the inputs and outputs of the system. The inputs include the customer’s name, address, telephone number and emergency contact details. She has found that the coaches are all fitted with 45 seats numbered from 1 to 45 and that the coach company assigns a four-digit code to each of the tours. As the information-gathering process continues, Kara also discovers that the coach drivers’ passenger lists contain the seat number, the passenger name and an emergency contact number.

Kara then compiles a report which details all her findings, listing the data items input into the system and the outputs from the system. Processes and procedures needed to convert input data to output data are also described. In order to make the workings of the system clearer for the designers, she creates a number of diagrams showing how the system works. One of these, a **dataflow diagram**, is shown in Figure 3.4. (You will learn how to understand and construct dataflow diagrams in the HSC course.)

Exercise 3.2

- 1 From the description above, create a report that might be presented by Kara once she has examined the workings of the system. Save your document in the CUMFYCOACH directory as a word-processed document called CUMFYPLN.
- 2 Examine the system you described in Exercise 3.1 and present a plan for the solution of the problem. Identify the inputs, outputs and processes that are needed for the solution to your problem. Save your document as a word-processed document with the filename PLANNED.

Building the solution

The next step in this method of software development is to hand over the specifications to the design team. This team will look at the areas of design and programming. Sometimes both tasks are performed by the same individual, but at other times these tasks are assigned to specialists. The design staff also use the specifications to create a set of test data that can be used later to test the program and its parts.

Designers have the responsibility of first breaking the required processes down into smaller parts, known as **modules**. This procedure ensures that those responsible for the design of the processing steps have a small, understandable process to work on. Modularisation also allows for these modules to be saved and reused in other, similar applications. The third great benefit comes in the stages of testing and modification, as these smaller units can be tested and modified without affecting those sections known to work well.

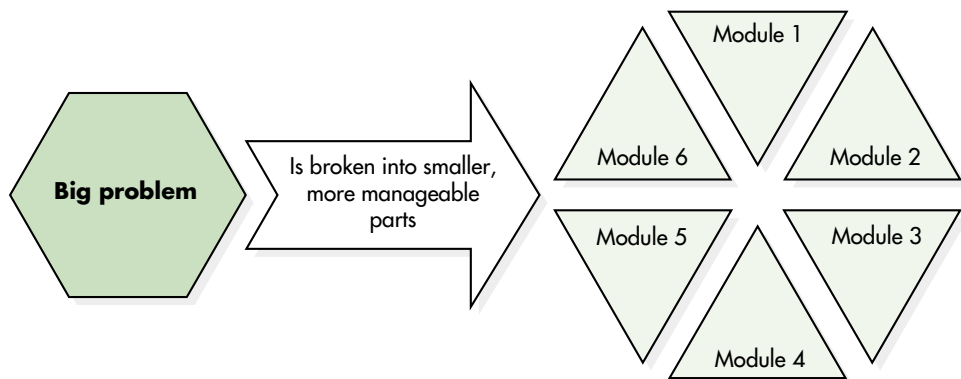


Figure 3.5 A problem is much easier to solve if it is broken into smaller, more manageable parts.

Each of the modules is then turned into a set of steps, known as an **algorithm**. It is the algorithm that is then passed on to the programmer for coding into an appropriate computer language.

Cumfy Coaches—building the solution

The design specifications for the new system are now passed to Stephanie and Katharine, the programmers. For this small system, they are acting as both designers and programmers.

They decide to first divide the system into two large units: one to handle the processing of customer details and the other to create the passenger list for each of the tours. These two major system parts are shown in the dataflow diagram in Figure 3.4. However, the ‘Record customer details’ module is still too large to be comfortably handled, so it is broken down further.

While Katharine is working on decomposing the customer details module, Stephanie starts to create a set of test data that can be used to test the modules. This test data consists of details of a number of customers and a couple of tours. She carefully chooses these items so that she can predict what the outputs will be.

When they are happy with the sizes of the modules, the programmers work on creating each module’s computer program. First they write the steps as an algorithm (the necessary processing steps arranged in an order that will solve the problem), then they test the algorithm and finally they translate it into a language that can be run on a computer. (These processes will be examined in a later chapter.)

Exercise 3.3

- 1 Decompose the customer details module of the Cumfy Coaches example into smaller modules. Describe the task that each of your modules should perform. This exercise should be word-processed and saved as CUSTMODL in your CUMFYCOACH directory.
- 2 Break the system you examined in question 2 of Exercises 3.1 and 3.2 into a number of smaller modules. Present your answer as either a word-processed document or a diagram (using a drawing program) with the file name MODULED.

Checking the solution

Once designed, the modules can be checked using the test data. Checking is usually carried out both before the module is coded into a programming language and after it has been coded. The checking procedure is designed to detect errors, which can come from different sources.

Checking the algorithm design is carried out to make sure that the processes chosen will properly perform the required task. This makes the task of finding errors in the later stages much easier.

When the coded version of the module is tested with the test data, the programmer is looking for errors that might occur when the module is run on the computer. The two steps to this type of checking are to check the program code manually first (called **desk checking**) and to test the module when it is being used on a computer (called **run-time checking**). Errors in a module will lead to modifications that will overcome the problems.

Once this stage of the cycle is complete, the application is ready for use by the customer.

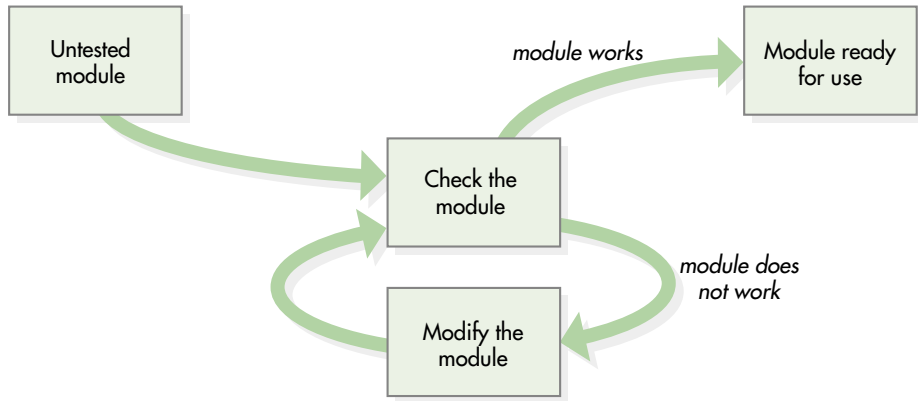


Figure 3.6 The modules go through a cycle of testing and change until they perform the tasks they are supposed to.

Cumfy Coaches—checking the solution

Katharine and Stephanie set to work on the design of each of the modules. As each module is completed as an algorithm, it is tested manually. When they are satisfied that the module will work as planned, they move on to the coding step. This involves choosing a language and converting their module steps to those in the chosen language.

On completion, each of the modules is first checked for mistakes in the coding. (These are called **syntax errors**—you will look at these in more detail in Chapter 5.) The next step is to take the test data and perform a manual check on the code.

The coding stage is followed by the process of entering the code into the computer and again running the program using the test data. After the modules are complete, they are gradually put together to form the whole program. As each of the modules is added, the program is tested again with the test data to ensure that it performs the tasks it is supposed to.

The program is then ready for use by Cumfy Coaches.

Exercise 3.4

- 1 From the description of the system in the previous sections, design a database that will store the customer details for one coach tour. Use a simple database management system such as the one that is included in Microsoft Works or Appleworks. Save your file with the filename TOURDATA in the same directory as the other exercises. Fill up at least ten records in the database with some test data.
- 2 Describe the tasks of one of the modules in your own case study. Use an appropriate computer application to create a working module. Choose some data to test your module.

Modifying the solution

When the new system is in operation, continual evaluation by users takes place to determine whether it meets the requirements. Users identify any changes that may be needed to the system. These changes are made by again following the steps in the software lifecycle.

This approach to software development is very structured, although stages tend to overlap. At any stage in the cycle it is possible for development to be backtracked to an earlier stage of the process. This occurs when problems are found with the output of a previous stage. For example, during the design stage a problem may be found with the analysis. This problem must be fixed before the development continues.

This approach allows teams of people to be involved. As the steps are distinct, each of them may be performed by a different specialist individual or group. This is why it is so important that clear and concise documentation is provided at each stage of the process.

Cumfy Coaches—modifying the solution

After using the program for a short while, the staff make some suggestions as to how the software might be improved.

In the office, Janine has discovered that along with the details of each passenger there is a need for a record of payments to be kept. Many of the passengers are paying for their trips in two or more instalments. She would like a module that keeps track of these payments and generates letters of reminder that could be sent six weeks before the departure of each tour.

The drivers would like the system to be able to identify any special needs of the passengers, such as special dietary requirements.

The managing director would like the system to be able to manage the rostering of coaches on trips, so that they spend as little time as possible off the road.

With these new requirements in mind, Kara is now drawing up a new set of specifications that could be used in the modification of the software.

Exercise 3.5

- 1 Write a set of specifications that could be used to modify the Cumfy Coaches application described above. Save the document with the filename CUMFYMOD.
- 2 Examine your case study and suggest modifications to this application that might be necessary. Present your document in an appropriate form.

Usefulness of the structured approach

Major disadvantages of this approach are that it is a time-consuming process and an expensive one. Each of the stages in the process cannot be undertaken until the previous one is completed or is close to completion. In some applications time is critical, and so the structured approach is not always the best option to take in development. However, for large and/or complex applications it provides a structure within which efficient development can take place. This is especially important when a team of developers is employed, as each member of the team needs to have a clear understanding of where the process is up to.

Exercise 3.6

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
The structured approach to software development consists of five steps _____, _____, _____, _____ and _____. The first stage focuses on _____ the problem and lists the _____ of the software. In the second stage the _____ and _____ of the existing system are analysed so that the development team knows how the system works. Stage 3 involves _____ the solution, followed by the _____, after which the software is placed in operation. While in operation, there is a constant evaluation, the results of which are used to _____ the program.
- 2 Using your answers to question 2 in Exercises 3.1 to 3.5 as an example, describe, in your own words, the processes involved in the structured approach to software development.
- 3 Examine the processes and procedures in your school library and develop a set of specifications for a new software application to run the library.

EXTENSION

- 4 Describe the roles of each of the individuals involved in the development of the Cumfy Coaches software application. Are any of the people involved more important than the others? Give reasons for your answer.

The prototyping approach to software solutions

A **prototype** is a working model that is usually used to gain information about how the elements of the system work together. For many years industries have used **throwaway prototypes** to test new design features which are incorporated in the final product. The car industry creates so-called 'concept vehicles' to test new technology and customer reaction to design features. The information gathered in this way is then used in the design of new car models. The concept vehicles themselves are either consigned to the scrap heap or kept in a museum for viewing; they are not used.

There are software development tools that can produce a prototype that can be refined into a fully working solution. This type of prototype is known as an **evolutionary prototype** as the application evolves from a cycle of use, evaluation and modification.

Prototyping as an information-gathering tool

When a prototype is used as an information-gathering tool, it is often at the beginning of the structured approach to software development, described earlier. When the user is unclear about exactly what is required from the application, a systems analyst will develop an initial set of specifications that are quickly implemented by the programming team. Prototypes of this form concentrate on inputs and outputs. Little regard is paid to error checking. The prototype is passed

to the user for evaluation under working conditions. On receiving the evaluation, the systems analyst incorporates the user's suggestions into the revised prototype. When the user is satisfied that the prototype performs the required functions, this information is incorporated into the requirements of the new system and the structured approach is continued. The prototype in this situation is a 'throw-away', as it is discarded when the requirements phase of development is finished.

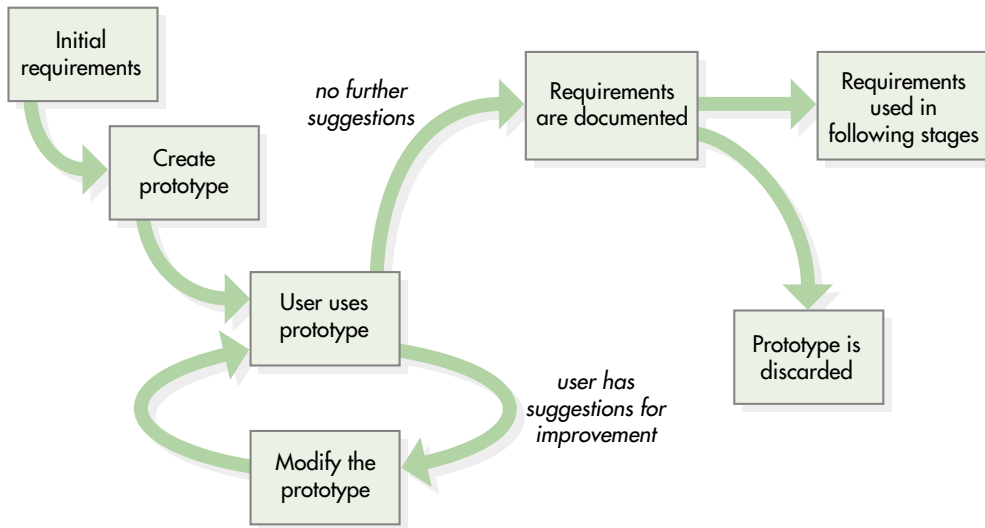


Figure 3.7 A throwaway prototype is used to gather information and is then discarded.

Although prototyping appears to offer a shortcut in the normal system development cycle, a prototype should be developed systematically with appropriate documentation. Particular attention needs to be paid to documenting modifications made as the prototyping process proceeds. If the prototyping process is used without care, a working system may be constructed which does not provide all the information needed.

This development tool is not suitable for complex processes or those that involve a large amount of mathematical manipulation. In addition, a prototype will often run more slowly than is needed for the final application, often as a result of the prototyping language. Prototyping is a suitable tool for applications such as multimedia applications and online enquiry systems. It is becoming more common for software developers to prototype user interfaces in order to gauge the users' reactions to the proposed design.

North Sydney Airport Corporation—prototyping for information gathering

The North Sydney Airport Corporation wishes to install an interactive tourist guide at its airport. The guide is to provide information about a number of tourist attractions and an accommodation booking facility that will accept credit cards for payment. The airport, which will serve both interstate and international travellers, is not yet in operation, so the systems analyst decides to prototype the system. The initial prototype will be developed with the emphasis on input and output modules; a reduced number of tourist attractions can be used to trial the system, and more added when the final system is implemented.

The initial prototype consists of a main menu module which leads to modules explaining local attractions, accommodation offerings and an accommodation booking service which allows credit cards to be used to pay a booking deposit. This system is a particularly good candidate for prototyping as it is an interactive system which will be used by untrained people. The interface needed for an application of this type needs to be intuitive and well tested. The system can be evaluated under realistic conditions, with a reduced set of data.

Prototyping as a software development approach

Prototyping can also be used as an approach to the development of a final solution. As with the 'throwaway' prototype, an initial set of specifications is supplied to the programming team. Using this set of requirements, the team uses a set of software tools to create a solution. This solution is passed to the user for use and evaluation. Again, following evaluation the prototype is modified and passed back to the user for further operation and evaluation. This process continues until the user is satisfied with the operation of the software. However, this time the prototype is not discarded, but is used as the final solution. This approach is commonly known as evolutionary prototyping.

This form of software development is not suitable for large-scale projects, as detailed feedback from the user is important to the success of the project. With a large number of users or a complex system, this method would produce problems in gathering and analysing feedback from the users.

Evolutionary prototyping will generally produce a software application in a shorter time and at a reduced cost to the client. The reduction in time and expense occurs because the process is less formal than the structured approach. To ensure that the prototype is able to be maintained, it should still undergo the stages of defining the problem, building and checking, although the distinction between the stages will not be as clear as in the structured approach.

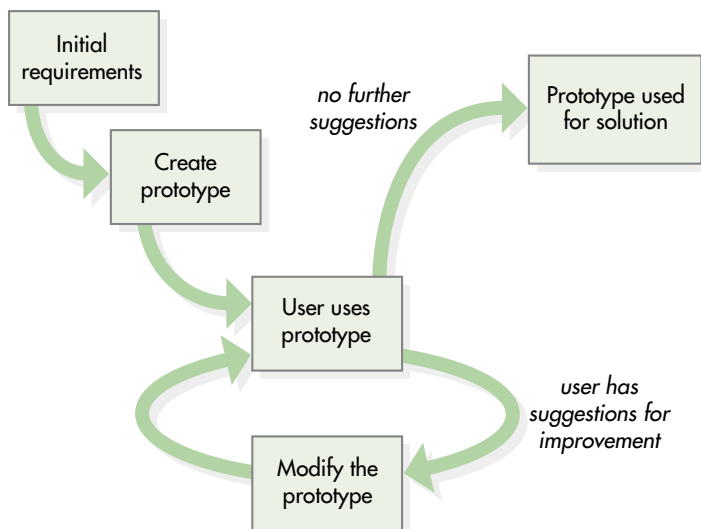


Figure 3.8 Evolutionary prototyping leads to a fully working application that continues to be used.

North Sydney Airport Corporation—evolutionary prototyping

The North Sydney Airport Corporation's interactive tourist guide can also be developed as an evolutionary prototype. The plan in implementing this prototype system is again to develop one which lists a limited number of attractions. Emphasis in the initial design is on input and output modules. The difference is that, in this case, the prototype system is installed in the airport. As it is used, the development team uses feedback from both visitors and attraction operators to modify the features of the system.

Exercise 3.7

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
A prototype is a _____ that is used to gain _____. When used in this way it is often _____ once it is no longer needed for this use. It is used in this way when the user is _____ of what the system _____ are. As the user works with the _____ it is evaluated. The results of this _____ are used to refine the prototype. This method can also be used as a tool. In this _____ case, the _____ is continually _____ until it is able to perform all the _____ tasks.
- 2 Design suitable input and output screens for the tourist information system to describe three motels in the area and to accept bookings. Use a graphics program or a hypertext system for this question. Save your file as TOURGUID.
- 3 Explain how you would use a simple flat-file database system, such as that found in Microsoft Works or Claris Works, to prototype a stock-control system for a small general store.
- 4 Create a prototype for a general store stock-control system. The store owner wishes to keep a database which details the name of each item in stock, the supplier, the quantity, the level when it should be reordered, the wholesale price, the percentage markup and the wholesale value of the stock in the shop. Test the prototype, making suggestions for additional features.

EXTENSION

- 5 Choose an application you think is suitable for prototyping, and give a brief description of the system requirements and your reasons for using prototyping as a development tool. Design at least one input and one output screen for this system. Present your answer as an appropriate set of documents.

Rapid application development (RAD)

The term **rapid application development (RAD)** is used very loosely to describe any software engineering process design that leads to faster application development. A number of different approaches may be used in RAD, including those of CASE (computer-aided software engineering), the reuse of code and the use of templates. Software tools used would include development environments such as Visual Basic, Hypercard, Hyperstudio, Access, Filemaker Pro and REAL Basic.

One of the main features of the RAD approach to software development is the lack of formal stages. In the more structured approach, the lifecycle stages can be clearly identified by the processes taking place. Less formal stages are more suited to the development of smaller, low-cost projects, as the development teams are also small, often consisting of one programmer.

When a RAD approach is taken, the user is often directly involved with the programmer. The tools allow a programmer to develop a solution which can then be tried by the user.

Barton's Books—a RAD approach

Barton's Books is a small bookshop catering for a wide variety of tastes. At present the staff use a cash register and a card system to keep track of their stock. However, this system is becoming unmanageable, so they have decided to upgrade to a computer system.

After discussing the requirements with the store manager, the software developer, Marie, decides to implement the system using a relational database management system, such as Access or Filemaker Pro. This will allow her to create and link files to manage the stock, the suppliers and the customers.

The chosen development tools allow for rapid development, since input and output form design are functions that are included in these development packages. The use of one of these database management systems also gives Marie the ability to customise some of the operations as macros which can be used by either keystroke or screen buttons.

Exercise 3.8

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
Rapid application development, or _____, can be used to describe any _____ process that leads to a development. This approach has _____ stages that are suitable for _____ and _____ cost projects. Often the _____ and programmer work closely together.
- 2 Describe a situation in which a developer might decide to use a RAD approach to software development. Explain why the RAD approach should be used in this case in preference to the structured approach.
- 3 Use a database management system, such as Access or Filemaker Pro, to develop an application to catalogue a compact disk collection. Reports should be created to allow the user to view the data in these ways: by artist, by music type and by CD name. Buttons, or macros, should be used to allow the user to move from one view to the next. When moving between views, part of the macro should sort the database records on the field that is requested. Save the file as CDCOLL.

End-user development

As hardware and software have become more powerful, so has the ability of users to customise applications to their own situations. The adaptation of software tools to varying situations by users has enabled many of them to create software solutions to their specific problems quickly and cheaply.

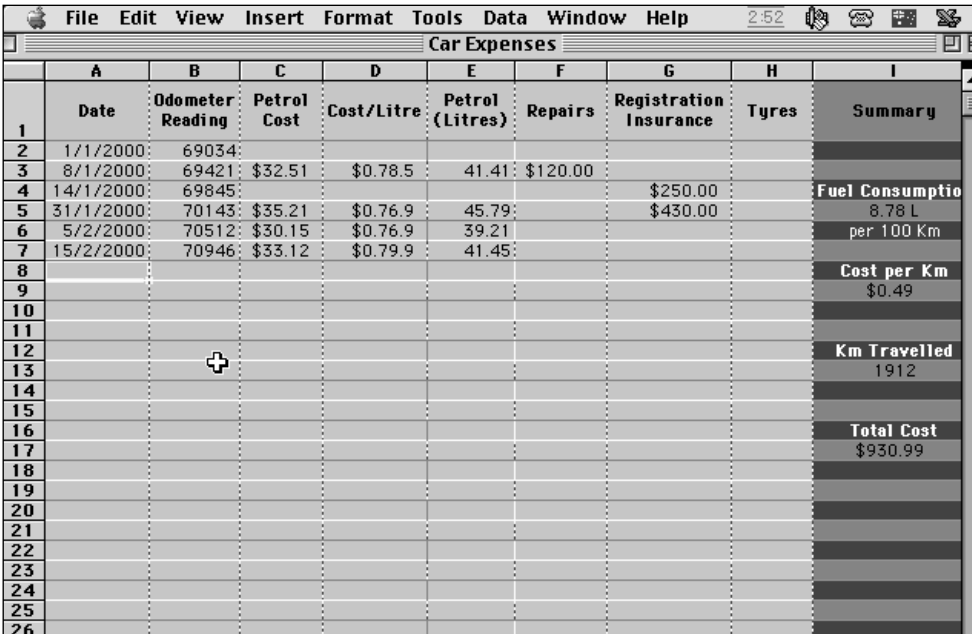
The emphasis in this type of development is on rapidly arriving at a solution. There are no formal stages in this type of development, the user taking advantage of the application's capabilities and customising them for use. These projects are often small-scale projects initiated for the sole use of the person doing the customising. It is not unusual for these projects to be used for a long time, as the end user has created a tool that reflects his or her own ways of working. As the user finds further tasks for the system to perform, it can be modified to meet these new needs.

Because there is little or no involvement of outside personnel, the end-user application can be produced at a fraction of the cost of a custom-built software package. A further advantage of end-user development is the familiarity the user has with the package, thus removing the need for further training in its use.

The off-the-shelf application packages that end-user development uses generally consist of a fourth-generation language such as a database management system or spreadsheet application. These tools supply a number of features which simplify the interface with the computer and the language required. Within these applications will be found tools such as a report generator, a query language and a screen design.

Jessica's motoring records

Jessica wants to keep track of her car expenses, so she decides to use a spreadsheet application to help her. In this spreadsheet she wants to be able to store the odometer reading when the expense occurred. The expenses she has listed are for



	A	B	C	D	E	F	G	H	I
1	Date	Odometer Reading	Petrol Cost	Cost/Litre	Petrol (Litres)	Repairs	Registration Insurance	Tyres	Summary
2	1/1/2000	69034							
3	8/1/2000	69421	\$32.51	\$0.78.5	41.41	\$120.00			
4	14/1/2000	69845					\$250.00		Fuel Consumption
5	31/1/2000	70143	\$35.21	\$0.76.9	45.79		\$430.00		8.78 L
6	5/2/2000	70512	\$30.15	\$0.76.9	39.21				per 100 Km
7	15/2/2000	70946	\$33.12	\$0.79.9	41.45				
8									Cost per Km
9									\$0.49
10									
11									
12									Km Travelled
13									1912
14									
15									
16									Total Cost
17									\$930.99
18									
19									
20									
21									
22									
23									
24									
25									
26									

Figure 3.9 Jessica's spreadsheet is an example of end-user development.

petrol, repairs, registration and insurance, and tyres. She sets out the spreadsheet so that it calculates the total cost as well as the cost per kilometre.

Using the spreadsheet's help facility, Jessica has no problem devising her spreadsheet. In her design, she uses each 'page' of the spreadsheet workbook to hold the data for a year. The application takes her a little over an hour to complete. Since the sheet is for her own use, Jessica does not even bother to document how it works. When the following year comes, she will copy the formulas from one sheet to the next.

Jessica could also create a macro that will automatically set up a new sheet for the new year.

Exercise 3.9

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

End-user development involves the _____ of a system creating a solution to a _____ without the aid of a _____ team. More _____ computers and application _____ mean that a software solution can be created _____. The _____ of the solution is also reduced as _____ help is not used. The users also enjoy _____ with the application because he or she developed it and so needs no more _____ in its use.

- 2 Using Figure 3.9 as a guide, develop the spreadsheet solution to Jessica's problem. Save your file as JESSCAR.
- 3 Use an appropriate software package to develop an application to manage your budget. Format the document to make it useable and create a button, or macro, to print a copy of it. Save your file as BUDGET.

EXTENSION

- 4 What are the advantages and disadvantages of an application created by the user? Use an example to illustrate your answer.

Team Activity

As a team, investigate the requirements for a school report program that could be used within your school. Write a report which details the requirements of such a system. Examine the inputs and outputs of

the report system, and use interviews and other information-gathering processes to plan a system that could be implemented. Present your report in a suitable form on disk.

Review exercises

- 1 Describe the types of project which are best suited to each of the four development approaches described in this chapter. Word-process your answer and save it as a document called DEVAPPCH.
- 2 Name the development approach best suited to each of the following applications, giving reasons for your choice.
 - a An airline wants to develop a program to manage aircrew rosters.
 - b A school student wants a program that can manage her study timetable.
 - c A radio station needs a computer application to manage its advertising schedules.
 - d An interstate coach company needs a computer application to manage its booking system.
 - e A large shopping centre requires an interactive guide to its facilities.
- 3 Use a hypertext development tool, such as Hypercard or Hyperstudio, to develop a prototype of a school directory. Your directory should have a main page and at least three other linked pages.
- 4 Use a suitable application to develop a budgeting program for a family.
- 5 Investigate the requirements of a service station for a software package that can be used to manage the business.
- 6 Investigate the requirements of a primary producer for a software package that can be used to manage the business.
- 7 Describe the features of a development tool such as Visual Basic that make it suitable for rapid application development.

Chapter summary

- Four of the more common approaches to software development are the structured approach, prototyping, rapid application development and end-user development.
- The structured approach consists of five steps: defining the problem, planning the solution, building the solution, checking the solution and modifying the solution.
- The structured approach is used for large-scale projects with a long development time and a large budget.
- The structured approach is used when a team is working on the project, as the structure gives direction for each of the members of the team.
- The personnel involved in the structured approach are analysts, designers, programmers, users and management.
- Prototyping is the creation of a working model of the software solution.
- The prototype is used as a tool to gain information, after which it is thrown away, or it is progressively refined into a working solution.
- The throwaway prototype is used when the users of a system are unclear as to their requirements.
- The evolutionary prototype is created and utilised, with user evaluation being used to further refine the prototype until it performs the required tasks effectively.
- Prototypes are used for small-scale projects with small budgets and short development times. The process of prototyping is also less formal than the structured approach.
- The process of evolutionary prototyping usually involves a small number of people. The personnel involved include programmers and the users.
- Rapid application development (RAD) is a broad approach to software development. CASE tools are often used, as are development environments such as Visual Basic.
- RAD is identified by the lack of formal stages, and is used in small, low-cost projects with small development teams involved.
- End-user development is characterised by the use of readily available software packages to develop a small-scale project.
- The end user is often the only person involved with developing the system.

chapter 4

→ *Defining the problem and planning software solutions*

Outcomes

- describes and uses appropriate data types (P 1.2)
- describes the interactions between the elements of a computer system (P 1.3)
- explains the effects of historical developments on current practices (P 2.2)
- identifies the issues relating to the use of software solutions (P 3.1)
- investigates a structured approach in the design and implementation of a software solution (P 4.2)
- uses a variety of development approaches to generate software solutions and distinguishes between these approaches (P 4.3)
- uses and develops documentation to communicate software solutions to others (P 5.2).

Students learn about:

Defining the problem

- understanding the problem
- identification of inputs and required outputs
- determining the steps that, when carried out, will solve the problem

Abstraction/refinement

- the top-down approach to solution development
- refinement of a proposed solution
- modification of an existing solution

Data types

- data types used in solutions, including:
 - integer – string – floating point
 - Boolean – date and currency formats

- data structures, including:
 - one-dimensional array – record
 - sequential files
- limits of particular data types
- integer representation in binary, decimal, octal and hexadecimal forms
- the impact of hardware/software limits on different data types

Structured algorithms

- methods for representing algorithms
 - pseudocode – flowcharts
- control structures
 - sequence
 - selection (binary and multiway)
 - iteration (pre-test, post-test including FOR..NEXT loops)
- software structures
 - subroutines – modularity
- use of standard algorithms, including:
 - load and print an array – process records from a sequential file
- checking the algorithm for errors
- historical events that led to the development of a structured approach to algorithm design

Students learn to:

- understand a problem by using an IPO chart
- work out a way to solve the problem and describe the solution using an algorithm description method
- check the solution to a problem by means of a desk check
- convert numbers to and from their decimal representation to binary, octal and hexadecimal forms
- distinguish between the programmer's, or abstract, view of data and the physical representation of that data within a computer system
- identify simple data types and describe the ways in which the different types are represented within a computer system
- differentiate between simple and structured data types
- differentiate between the structure of an array and the structure of a record
- reference an element within a data structure;
- match the data requirements of a problem to appropriate data structures and justify the choice
- create a paper representation of appropriate data structures for storing data
- identify the different constructs used in an algorithm
- read and understand algorithms written as pseudocode and as a flowchart

Personal Profile—Grace Hopper (1906–1992)

Grace Hopper was born Grace Murray in 1906 in New York. She graduated from Vassar College in 1928 with a BA in mathematics. Remaining at Vassar, she earned an MA in 1930, a PhD in 1934 and married Vincent Hopper, who died in 1945.

In 1943 Grace left Vassar, where she was an associate professor, to join the US Naval Reserve. After Midshipman's school, she was sent to the Bureau of Ordnance Computation Project at Harvard's Cruft Laboratories where she worked on the Mark series of computers. Grace became the third person to program the Mark I computer, receiving awards for her pioneering work.

In 1949 Grace joined the Eckert-Mauchly Computer Corporation which produced the UNIVAC I, the first large electronic computer. She encouraged programmers to share libraries of code, thus reducing errors and the duplication of tasks. She is best known for her contribution to the

design of compilers which allow programmers to write programs in languages more closely approaching normal language. She was an active participant in the design of the COBOL business language. She remained with the company until 1967, during which time it was purchased by Remington Rand which in turn merged with the Sperry Corporation. Grace again joined the military where she remained until retirement in 1986, having risen to the rank of Rear Admiral. She remained active, consulting to the Digital Equipment Corporation, working well into her eighties. Grace died in her sleep on 1 January 1992.

The term 'bug' is supposed to have been coined by Grace Hopper when a moth was found to have caused a malfunction in one of the early computers.



Defining the problem

There are three main steps in producing a solution to a problem. They are:

- *Understand the problem.* Before a solution can be designed, the problem must be fully understood.
- *Work out a way to solve the problem.* How a solution is arrived at depends a lot on past experience. For example, a chef would have no trouble describing how to make French onion soup. If a similar problem has been solved, it is an obvious advantage. The plan of the solution may involve the use of an algorithm.
- *Check the solution to the problem.* After the solution to the problem has been designed, it needs to be tested. If the results are unsatisfactory, it is modified or discarded. There is usually more than one correct solution for any problem and the best solution is a matter of personal choice.

It is important to remember that after a solution has been produced it has to be implemented. A computer-based solution could be implemented using a software application or it could be converted into a programming language.

The following sections deal with some methods that can be used to help us understand the problem.

IPO chart

Before a computer solution to a problem can be implemented, it is necessary to understand the processes and data interactions within the system. Paper representations of the system allow the programmer to better understand the ways in which its components work together. An **IPO (input, processing, output) chart** is used to describe the data elements which will enter the system (or subsystem), the processes which will occur and the data elements which will leave the system. Although the formats of IPO charts might differ, they will contain a heading, a list of inputs, a description of the processes and a list of outputs.

The IPO chart in Figure 4.1 illustrates the action of a railway ticket vending machine. The traveller inserts a banknote and presses a destination key. The destinations with their fares are read from a fares master file and the particular destination is matched with the fare. The machine prints and issues the ticket as well as calculating and giving the change due to the traveller. At the end of the transaction the machine's transaction file is updated on the central system.

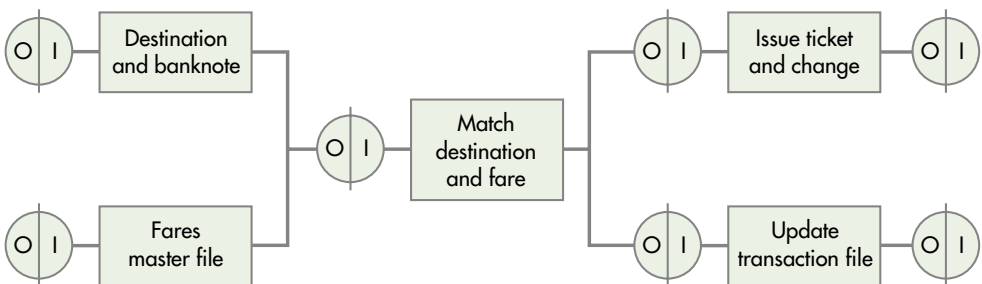


Figure 4.1 Ticket system IPO chart.

A different IPO chart format is shown in Figure 4.2. This chart describes the processing of an employee's wages to determine the amount of tax to be deducted, the outputting of the net wage (after-tax wage) and the amount of tax payable. The form that the processing takes is shown in more detail than in the previous type of chart.

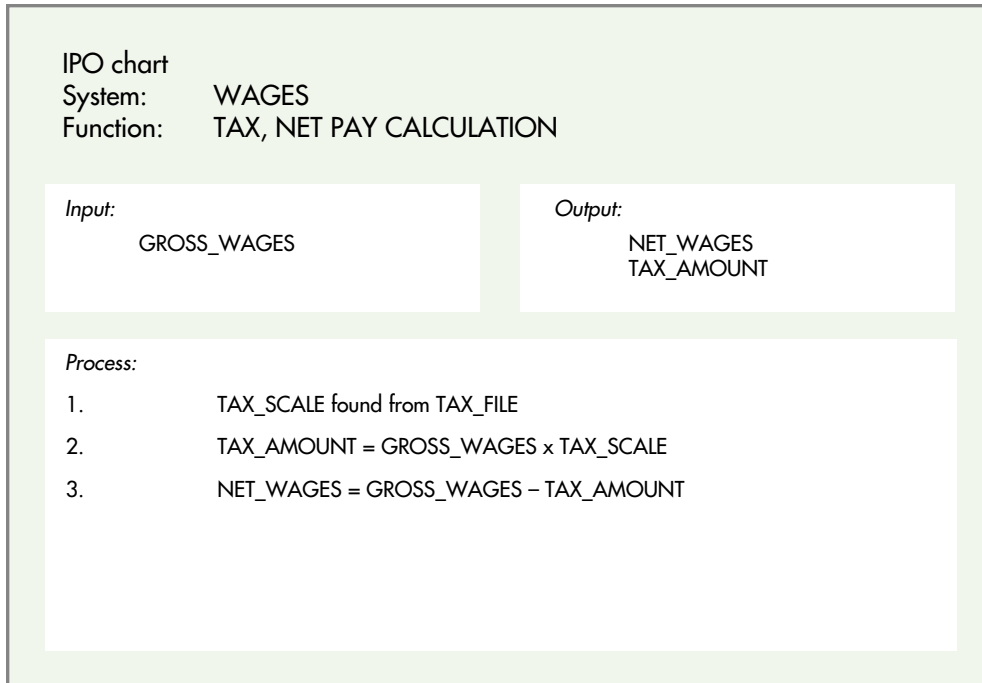


Figure 4.2 Wages calculation IPO chart.

In creating an IPO chart, it is important to look at the problem in the following way.

The first step is to look at the outputs that are required in order to solve the problem. We need to decide what it is that the system is to do before we look for the necessary inputs and processes. Designing a solution is very much like baking a cake. We need to know what type of cake we want before buying the ingredients and following the recipe.

After deciding on the outputs that are required, we then look at the inputs needed for those outputs. In terms of baking a cake, we are listing the ingredients that are needed for the type of cake to be baked.

The final step in creating an IPO chart is to decide on the processes that are needed to turn our inputs into the required outputs. In the cake example, this corresponds to the recipe.

Example

Terry Cotta's Tiling Company has decided to computerise its quoting system. A salesperson will enter the area's dimensions and the tile type that has been chosen by the customer. The system is to output a quote for the job, subdivided into costs for materials and for labour.

In this example, it is quite clear that the required outputs are the total cost of the job and the individual costs of the materials and labour. After this stage, the IPO chart will look something like the chart in Figure 4.3.

INPUT	PROCESS	OUTPUT
		Total cost of job Cost of materials Cost of labour

Figure 4.3 Stage 1 in producing an IPO chart. The outputs are specified.

In order to obtain these outputs, we will need the details as entered by the salesperson as mentioned in the design brief (that is, the tile type and the dimensions of the area). However, these are not the only inputs that are needed for this

INPUT	PROCESS	OUTPUT
Dimensions of area to be tiled Tile type Cost of tiles per square metre Amount of other materials (grout, adhesive) needed per square metre Unit costs of other materials Area of tiles that can be laid in and hour Tiler's hourly rate of pay		Total cost of job Cost of materials Cost of labour

Figure 4.4 Stage 2 of the production of an IPO chart now brings in the required inputs to achieve the outputs.

task. In addition to these data items, the system will need to be able to provide the cost of the tiles per square metre and the cost of the other materials needed, such as adhesive. As well as these costs, the labour charge needs to be calculated. This charge is also dependent on the inputs of the area of tiles that can be laid in an hour as well as the hourly rate of pay of the tilers. These amounts may depend on the type of tile chosen by the customer. We have now built up our IPO chart as illustrated in Figure 4.4.

Finally, we can look at the processes required to produce the customer's quote. These would include:

- calculating the cost of the tiles by multiplying the area by the cost of the tiles per square metre
- calculating the amount and cost of other materials such as adhesive and grout
- calculating the time that a tiler needs to do the job
- calculating the cost of the labour.

Our IPO chart is now finished and can be seen in Figure 4.5.

INPUT	PROCESS	OUTPUT
Dimensions of area to be tiled	Calculate area.	Total cost of job
Tile type	Calculate cost of the tiles.	Cost of materials
Cost of tiles per square metre	Calculate cost of the other materials.	Cost of labour
Amount of other materials (grout and adhesive) needed per square metre	Calculate cost of all materials by adding the cost of the tiles to the cost of the other materials.	
Unit costs of other materials	Calculate time required by the tiler.	
Area of tiles that can be laid in and hour	Calculate cost of the tiler's labour.	
Tiler's hourly rate of pay	Calculate total cost of the job by adding the cost of all materials and the cost of the labour.	
	Output total cost of the job, cost of materials and the cost of the labour.	

Figure 4.5 Stage 3, the processes required to achieve the outputs are included in the table.

Exercise 4.1

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

When we solve a problem, there are three steps we should follow. The first is to _____ the problem, the second is to find a _____ the problem and the last is to _____ the solution. One of the ways we can _____ the problem is to draw up a(n) _____ which lists the _____ required to produce the wanted _____ as well as the _____ required.

- 2 Describe the purpose and advantages of an IPO chart during the software development cycle.
- 3 A subsystem is to calculate the postage of a parcel according to the following rule:
The postage on a parcel is calculated as being \$2.00 plus \$1 for each kilogram or part of a kilogram.
Draw an IPO chart for this system.
- 4 Describe in words the system represented by the IPO chart in Figure 4.6.

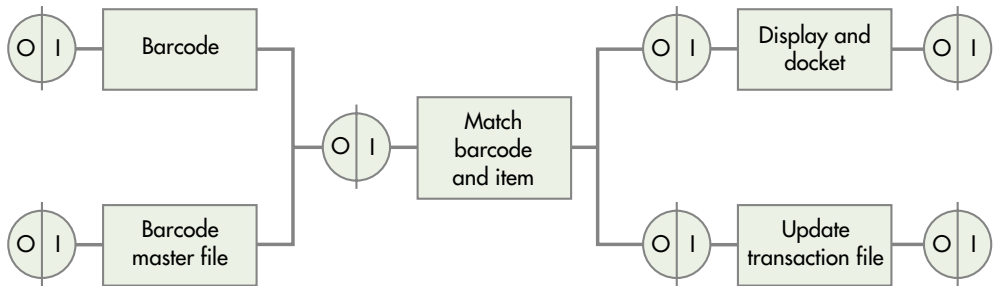


Figure 4.6

- 5 Describe in words the system represented by the IPO chart in Figure 4.7.

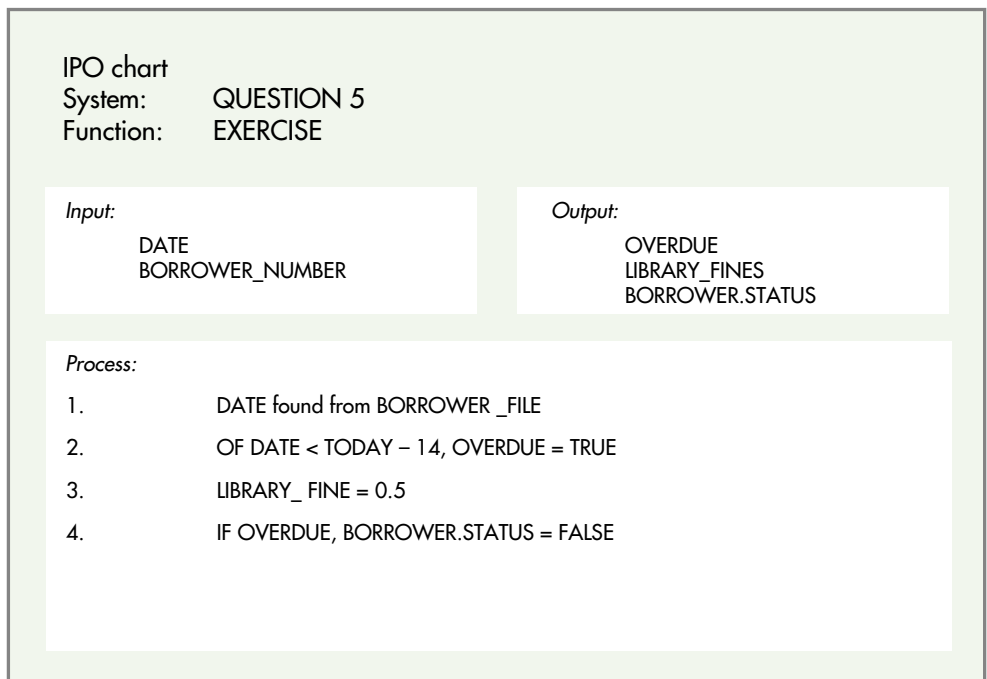


Figure 4.7

- 6 Using a word processor that has the ability to produce a table, create a blank table with the headings 'Input', 'Process' and 'Output' in one row together with a blank row. Use the 'Format Border' menu item (or similar) to create a border

around your table and between the cells. Save this document with the filename IPOBLANK. You can then use copies of this document to draw the IPO charts for the following exercises.

- 7 A wages system is to input the hours worked by an employee and the rate of pay, and then calculate the week's wages and output the gross pay (pay before tax). Draw an IPO chart for this system.
- 8 Choose a small subsystem and construct an IPO chart to describe its operation. In addition, describe its operation in words.

Abstraction/refinement

Top-down design

When designing an algorithm to solve a problem, many computer programmers use a **top-down design**. In this approach, large complicated problems are broken down into a series of smaller, easier-to-solve problems. The solution to one of these smaller problems is sometimes called a **module**. Clearly, a large complicated problem could consist of many modules. Each module is written separately before they are brought together as a whole. Top-down design results in computer programs which are easier to write and correct. In general, if an algorithm is not going to fit onto one page it should be broken down into modules.

The basic concept underlying the process of top-down programming is the **decomposition** of a solution into smaller and smaller units until each of the units can be expressed as one instruction. An alternative name for this method is **stepwise refinement**. This design method will produce a program which clearly shows the structure. Figure 4.8 illustrates the decomposition of a program into modules and sub-modules. The modules and sub-modules are sometimes given a numerical naming, each of the numbers in the name denoting a level in the

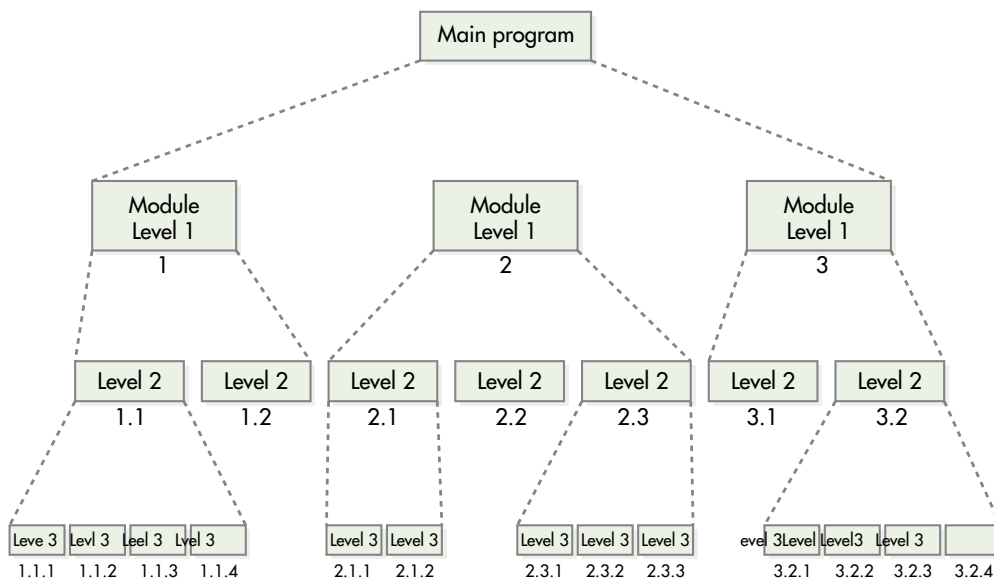


Figure 4.8 Top-down programming produces smaller and smaller program units.

hierarchy. As an example, module 2.1.2 is at the third level of the hierarchy as it contains three numbers. Each of the numbers represents its relationship to the other modules. We can see immediately that module 2.1.2 is a part of module 2.1, which, in turn, is part of module 2.

One of the main advantages of adopting this programming approach is that the logic of the program can be examined and tested at each stage of development. A main program, for example, can be constructed and tested before any of the modules have been created. As each of the new modules is created, it can first be tested to determine whether it works. Once this testing and modification process is complete, the module is put in place and tested with the other written modules. This simplifies the testing procedure, since errors which occur can be directly attributed to the new module. When the new module is in place and working properly, the next one can be written. The main program or algorithm is often known as the **driver module**.

Example

Suraya has been asked to create a program which will act as an address book. In the first analysis, she decides that the program will contain an initialisation module, an update module, a search module and a closing module. This decomposition has divided the total program into four smaller units.

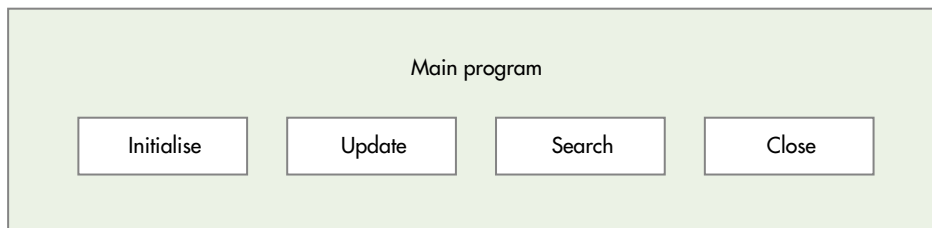


Figure 4.9 First decomposition of the address book program produces four modules.

The four modules are each decomposed into smaller units. For example the update module can be decomposed into three sub-modules which allow the user to add a new name to the book, change an existing entry and delete an entry from the book.

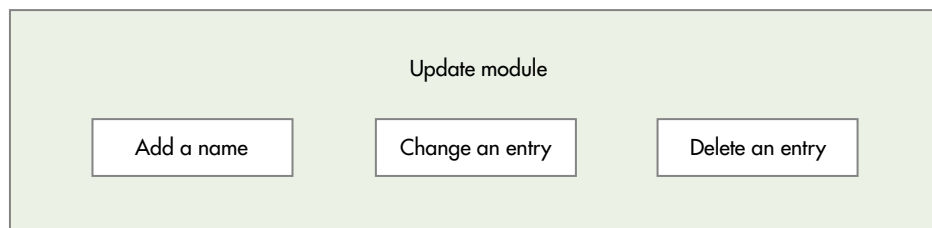


Figure 4.10 Decomposition of the update module leads to three smaller modules.

Suraya will then further decompose the modules at this level, repeating the process until the modules are expressible as a sequence of instructions.

When writing the program, Suraya can easily write and test the main controlling program. Once it has been checked and tested, the modules one level down the hierarchy can similarly be created.

Modification of an existing solution

Quite often an existing system is able to perform a number of the tasks required by its replacement. In these situations it is sensible to use those parts of the old system that still work satisfactorily. In other cases, the existing system may be performing well, but the users have identified a further function they need. Again, it is uneconomical and wasteful to design a completely new system and throw out the old one.

In modifying a solution to cope with new needs of the users, we still move through the steps of problem solving (understand the problem, design the solution and check the solution). However, we can use, or change, parts of the existing solution in our design. Those parts we use without modification should still take part in the testing of the new solution, as the manner in which they interact with the new parts of the system might affect the whole new system.

Exercise 4.2

- 1 Choose the word or phrase from the list which best completes the statements below:

decompose, driver module, hierarchy, modules, stepwise refinement, top-down design

- a The process of breaking a problem in smaller problems is known as _____ or _____.
 - b A program containing _____ allows the structure to become clearly seen.
 - c The numbers given to steps are used to show the _____ of those steps.
 - d The programmer will _____ a problem into smaller problems.
 - e The main algorithm is often known as the _____.
- 2 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Computer programmers use a ~~design method~~ to solve a problem. The term _____ is also used to describe this process. By _____ a problem into _____, more manageable parts, it becomes to ~~solve~~. The smaller parts are known as _____.

- 3 James has decomposed the instructions for a washing machine into the following set of modules (Figure 4.11). Decompose the initialise module into a number of sub-modules.

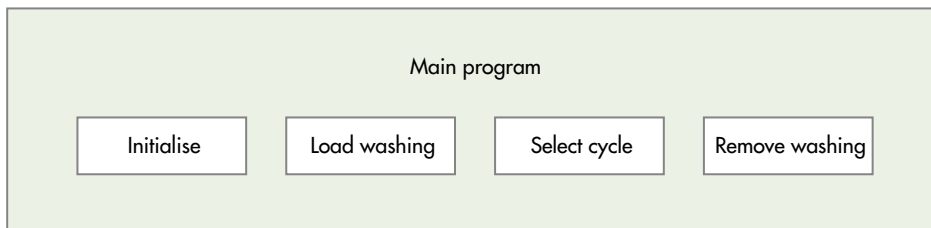


Figure 4.11 The first-level modules of James' washing machine program.

- 4 An algorithm for preparing and serving breakfast is to be designed. Decompose this problem into a number of smaller steps at the first level of decomposition; that is, list the modules that will form the main program.
 - 5 Take one of the steps from question 3 and decompose it to the second level. Are these steps small enough to now write an algorithm? Justify your answer.
 - 6 You have been asked to write an instruction manual for a video recorder. Your first job is to take the problem and decompose it into smaller steps. (These will form the main 'chapters' of your instruction manual.) Each of these steps is then to be further decomposed to a second level. (These would form the sub-chapters of your instruction manual.) You do not have to write any algorithms for this problem. You can use the 'outline' feature of a word processor to present your decomposition. Save the document as VIDSTEPS.
 - 7 Choose one of the second-level modules from your video recorder problem (such as recording using the timer) and write an algorithm that describes the steps of this module. You can use the word processor to present your algorithm, using the 'TAB' key to indent the instructions. Save the document as VIDMODUL.
 - 8 Write an algorithm for the driver module of the breakfast problem in question 4. You can use the word processor to present this algorithm. Save it as BREAKALG.
-

Data representation

Digital computers use the **binary number system** to represent data items and instructions. The binary number system requires only two digits to represent all numbers (we use 0 and 1). The medium we use to store data will determine how these digits are represented. Common media are electronic, magnetic and optical:

- *Electronic media*, such as the RAM inside a computer, can be thought of as using the electrical states 'on' and 'off' as representing the digits 1 and 0.
- *Magnetic media*, such as disks, represent data as magnetic 'spots'. If we want to represent a 1 magnetically, we can magnetise the spot. To represent a 0, the spot is left unmagnetised.
- *Optical media*, such as fibre-optic cables, use pulses of laser light to represent 1s and no light to represent 0s.

Unfortunately as human beings we do not think of numbers in binary form. Our counting numbers are based on a system where we use ten digits (0 to 9) in our representation of numbers. So in order for us to understand what happens inside a computer, we will need to be able to convert numbers from our decimal form into the binary form used by the computer.

Conversions to and from the binary number system

As mentioned above, the binary number system uses only the digits 0 and 1 to represent all numbers. This means that, once we have counted up to one, we have used up all the single-digit numbers and so we need to think of a way of representing two. In our normal counting, once we have reached the last single digit (9), we increase the number of digits to two to represent the next counting number (10). The same happens in binary. Thus the number two can be represented in binary by the digits 10_2 (this is read as 'one zero' not as 'ten'). The following number, three, can now be represented by 11_2 (one more than our

representation of two). Now we have used up all our two-digit binary numbers, so we have to use three digits to represent the number four (i.e. 100_2). This idea can be used to count to any number we like using the binary number system. The first ten counting numbers in binary are listed in Table 4.1.

Using a table like this is not a very practical way of converting larger numbers from our base ten counting system into binary. A division method is much more practical as it can be used to convert any counting number into its binary equivalent. The steps are as follows:
Step 1. Write down your chosen number and divide it by two (for the example we will convert 103_{10} into binary). *R* is used to show the remainder once the division has been performed:

$$\begin{array}{r} 2)103 \\ \hline \end{array} \quad R$$

Step 2. Repeatedly divide the answer by two until you reach 0, each time writing down the answer (quotient) and writing the remainder in the *R* column. (If there is no remainder, write 0.)

$$\begin{array}{r} 2)103 \\ \hline 2) 51 \\ \hline 2) 25 \\ \hline 2) 12 \\ \hline 2) 6 \\ \hline 2) 3 \\ \hline 2) 1 \\ \hline 2) 0 \end{array} \quad \begin{array}{l} R \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{array}$$

Step 3. The digits in the remainder column are then read from the bottom to the top to give the answer. This means that 103_{10} is equal to 1100111_2 .

When we have a large number of digits, it is often more convenient to group the digits in fours from the right-hand side. The binary equivalent of 103_{10} would be written as $110\ 0111_2$.

When it comes to converting binary numbers back to base ten, we use the fact that each digit in the binary number represents a power of two. The powers increase from right to left, starting with 2^0 (which is 1) as the value of the right-most binary digit. The digit on the right is also called the **least significant bit**, as it is the bit with the least value. Similarly, the digit on the extreme left of a binary number is called the **most significant bit**.

To see how to convert a binary number back to decimal we will look at the conversion of $110\ 0111_2$ back to decimal.

Step 1. Think of the digits with the powers of two above like this:

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Step 2. Write down the decimal values of the powers of 2. (Don't try to remember these powers as they are easy to calculate.)

$$\begin{array}{ccccccc} 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Step 3. Add up the numbers with 1s below them and ignore those with 0s below.

$$\begin{aligned} 110\ 0111_2 &= 64 + 32 + 4 + 2 + 1 \\ &= 103_{10} \end{aligned}$$

Number	Binary representation
one	1
two	10
three	11
four	100
five	101
six	110
seven	111
eight	1000
nine	1001
ten	1010

Table 4.1 Binary representation of the first ten counting numbers.

As most numbers we work with are fairly large, the binary number system is not really a suitable way to manipulate values. For this reason, two other number systems are used, both based on powers of two so that we can group binary digits together. These systems are **octal** (base eight) and **hexadecimal** (base sixteen).

Conversions to and from the octal number system

The octal number system is based on eight, and so uses eight digits to represent numbers. These are the digits 0, 1, 2, 3, 4, 5, 6 and 7. Rather than convert numbers directly from decimal to octal, it is usually much easier to change the decimal number to binary and then use these digits to find the octal equivalent. As an example, we will again use 103_{10} , which we already know is $110\ 0111_2$.

Step 1. Group the digits in threes from the right-hand side of the binary number, adding zeros if necessary to create the last group of three:

$110\ 0111_2$ becomes $001\ 100\ 111_2$

Step 2. Using the equivalents already known for the digits 0 to 7 in binary (see Table 4.1), replace each group of three with the appropriate digit. In the example, 001_2 represents 1, 100_2 represents 4 and 111_2 represents 7, so our octal equivalent to 103_{10} is 147_8 .

Again to convert an octal number back to a decimal number we need to look at the value of each of the digits in the octal number. This time each place represents a power of eight, moving from right to left in increasing order.

To change 147_8 back to a decimal number, we follow similar steps to the conversion from binary to decimal.

Step 1. Write the number with the power of eight above each digit.

8^2	8^1	8^0
1	4	7

Step 2. Change the powers of eight to their decimal equivalents.

64	8	1
1	4	7

Step 3. Add up the values.

$$1 \times 64 + 4 \times 8 + 7 \times 1 = 64 + 32 = 7 \\ = 103_{10}$$

As can be seen, the octal number system reduces the number of digits needed to represent 103_{10} from seven in binary down to three, but since eight is a power of two, we can still match up the digits with their binary equivalents. This feature makes octal a convenient way of rewriting binary numbers. However, it has fallen into disfavour somewhat, since most binary data these days is stored in bytes, consisting of eight bits, which are a little difficult to break into groups of three.

Conversions to and from the hexadecimal system

A far more convenient way of representing the binary strings that represent values stored within a computer is to break the one-byte units into two halves each of four bits. This means that there are sixteen different combinations of 0s and 1s which can be represented by these half-bytes (sometimes called **nybbles** or **nibbles**). This leads us to the number system based on sixteen, called the hexadecimal number system.

The next problem, once the number base of sixteen is chosen, is to find a way of conveniently writing the sixteen different digits that are going to be used in the representation of numbers in this base. (Remember that base two uses two digits, 0 and 1, base eight uses eight digits, 0 to 7, and base ten uses ten digits, 0

to 9.) The problem is solved by using the ten normal digits (0 to 9) together with the first six uppercase letters of the alphabet (A, B, C, D, E and F). Thus, if we were to count to fifteen in hexadecimal we would count 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Once we reach sixteen, we need to create another digit place, so in base sixteen (hexadecimal), sixteen is represented by the number 10_{16} .

Number	DeimalBinary	Octal	Hexdecimal
one	1	1	1
two	2	10	2
three	3	11	3
four	4	100	4
five	5	101	5
six	6	110	6
seven	7	111	7
eight	8	1000	10
nine	9	1001	11
ten	10	1010	12
eleven	11	1011	13
twelve	12	1100	14
thirteen	13	1101	15
fourteen	14	1110	16
fifteen	15	1111	17
sixteen	16	10000	20
seventeen	17	10001	21
eighteen	18	10010	22
nineteen	19	10011	23
twenty	20	10100	24

Table 4.2 The first twenty counting numbers in decimal, binary, octal and hexadecimal.

Bnary it pattern	Hexdecimalequivalent
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

The first twenty numbers in each of the number bases are shown in Table 4.2.

As with octal, it is much easier to convert a decimal number into binary form first before attempting the conversion to hexadecimal. Once the value is in binary form, the digits are grouped into sets of four from the right-hand side and then the bit patterns are changed to their hexadecimal equivalents (see Table 4.3.) The process is very similar to the conversion to octal.

Table 4.3 The hexadecimal equivalent for each of the sixteen bit patterns that can be formed from four bits.

To convert 103_{10} to hexadecimal, first convert it to binary (see the example on p. 99). The bit pattern $110\ 0111_2$ is now written as two groups of four bits by putting an extra 0 at the front to make it up to eight bits. Now, using the conversions in Table 4.3 we find that the bit pattern 0110_2 represents 6 and 0111_2 represents 7, so $103_{10} = 67_{16}$.

To convert 67_{16} back to base ten, we again look at the values of the digits. The right-hand digit again stands for the units, the next digit to the left stands for the sixteens and so on, with each digit to the left representing sixteen times the previous digit. So $67_{16} = 6 \times 16 + 7 \times 1$ or 103_{10} .

Notice that the largest value that can be represented as two digits in hexadecimal is FF_{16} . Doing the conversion, remembering that F_{16} stands for fifteen, FF_{16} represents $15 \times 16 + 15 \times 1 = 240 + 15$ or 255_{10} . Thus two hexadecimal digits can represent up to two hundred and fifty-five in our decimal system.

Exercise 4.3

- Complete each of the following statements with the most appropriate word from the list: binary, bit, decimal, hexadecimal, octal
 - The _____ system of numbers is used to store values in a computer.
 - We normally count using the _____ number system.
 - One digit in a binary number is called a _____.
 - Because binary numbers are difficult to work with, a system which contains the digits 0 to 7, called _____, is used.
 - The _____ number system uses the letters A to F to represent some of the digits.
- Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
 Digital computers store data as _____ and _____. Since there are only two bits in this system, it is known as the _____ number system. The _____ number system is inconvenient for us to use, so the _____ system based on eight and the _____ number system based on sixteen are also used by programmers.
- Convert the following decimal numbers into binary form:
 - 9
 - 17
 - 30
 - 89
 - 248

Extension: Design a spreadsheet that will perform the change of base from decimal to binary. Save it with the filename DECBIN. Check your answers to question 3 by using this spreadsheet.
- Convert the decimal numbers in question 3 into octal form.

Extension: Modify the DECBIN spreadsheet so that it will perform the change of base from decimal to octal. Save this spreadsheet as DECOCT. Check your answers to question 4 by using this spreadsheet.
- Convert the decimal numbers in question 3 into hexadecimal form.

Extension: Modify the DECBIN spreadsheet so that it will perform the change of base from decimal to hexadecimal. Save this spreadsheet as DECHEX. Check your answers to question 5 by using this spreadsheet.
- Convert these binary strings to decimal numbers.
 - 101_2
 - 1101_2
 - $1100\ 1011_2$

Extension: Create a spreadsheet that will perform the change of base from binary to decimal. Save this spreadsheet as BINDEC. Check your answers to question 6 by using this spreadsheet.

7 Convert these octal numbers to decimal numbers.

- a 15_8 b 37_8 c 206_8 d 777_8

Extension: Create a spreadsheet that will perform the change of base from octal to decimal. Save this spreadsheet as OCTDEC. Check your answers to question 7 by using this spreadsheet.

8 Convert these hexadecimal numbers to decimal numbers.

- a 15_{16} b $4B_{16}$ c $C0_{16}$ d $10E_{16}$

Extension: Create a spreadsheet that will perform the change of base from hexadecimal to decimal. Save this spreadsheet as HEXDEC. Check your answers to question 8 by using this spreadsheet.

9 First write each of these octal numbers in binary, then rewrite each number in hexadecimal.

- a 15_8 b 37_8 c 206_8 d 777_8

Extension: Create a spreadsheet that will perform the change of base from octal to hexadecimal. Save this spreadsheet as OCTHEX. Check your answers to question 9 by using this spreadsheet.

10 Explain why the binary number system was chosen for number representation in computers rather than the decimal system.

11 Show, using examples, why the hexadecimal rather than binary representation of numbers is preferred by computer programmers.

Data types

Data and instructions are stored in a computer system as strings of binary digits, with no distinction between their meanings; it is the program which decides the interpretation of the binary string. It is quite likely that a particular binary sequence stored in a particular location will represent part of an instruction on one occasion, a character on the next occasion, and part of an integer on another occasion. When looking at the physical storage of data in the computer system, we can move one step further and add the fact that the eight bits which make up a byte may not even be physically stored in the same integrated circuit. It is common practice for eight different chips to be used as primary storage, each of the eight bits of a byte being stored in the same location in each of a number of memory chips. These physical representations of data do not correspond to the ways in which the same data is represented in a non-computer system. Human beings think in terms of characters, numbers, words, sounds and so on.

The manner in which people represent problems to solve does not necessarily match the way in which the computer represents the same problems. People might think of problems in terms of lists, letters and numbers, pictures and sounds, graphs, etc. All these have to be represented in a computer as patterns of bits stored in the computer's memory. In first- and second-generation languages it was the programmer's role to translate people-oriented representations into computer-oriented representations. Third-generation languages and above have been designed to more closely resemble human languages and as such need to

offer data structures that more closely resemble those found in non-computer systems. A programmer employs a logical view of the data which more closely resembles the non-computer representation of the way we talk about problems.

Data often needs to be represented on paper so that we can understand what needs to be done to process it. Known as **abstract data types**, these structures are beneficial to the programmer, as the definition of their properties is independent of the computer system being used. Abstract data structures can be classified as simple data types or structured data types. The more basic of these two forms is the **simple data type** and it consists of the Boolean, character, integer, floating point and string data types, also date and currency formats. **Structured data types** are used to represent related data elements in a form that is manageable by the programmer, the compiler or the interpreter taking care of the physical representation within the computer system. The most common structured data types are the array, the record and the file. Files often take the form of a database.

Simple data types

Boolean data type

The **Boolean data type** is the simplest data type available as it can be used to represent one of only two possible values: true and false. For example, the result of an addition or a subtraction may be either positive or negative. A processor will store a bit in a register (often given the name **flag register**) after an addition or a subtraction, the value of this bit being set to 0 if the result is positive and 1 if it is negative. By using the stored value of this bit it is possible to make comparisons between two values as well, giving a computer the power to make decisions. Boolean data types have many uses in applications. For example, a Boolean container may be used to store whether a person is male or female within a database application, to determine whether a search has been successful, or to flag whether a particular data item should be included in a selection or not.

Character data type

The **character data type** is widely used as it represents the smallest item of data that an individual normally uses. Figures 4.12 and 4.13 show two common character sets. A character usually occupies one byte in primary storage, which gives a maximum of 2^8 or 256 different characters within one set. A larger number of characters within one set is possible by using two or more bytes to represent each character. Languages such as Chinese, which contains thousands of different characters, will employ a multi-byte representation. Since each character is represented by a string of bits, a code is needed to match the character with its representation. The most common code used at present is the ASCII code. Another popular code is EBCDIC, which is used mainly by IBM in their larger computers.

Integer data type

There are two commonly used simple numerical data types: the integer and the floating point. The **integer data type** is usually stored as two bytes (sixteen bits) and so has 2^{16} or 65 536 different combinations of bits. In order to cater for negative values, these numbers are stored in a form known as 'two's complement'. By using this method with two bytes, a range of -32 768 to 32 767 is achieved.

Binary				b6	0	0	0	0	1	1	1	1	
				b5	0	0	1	1	0	0	1	1	
				b4	0	1	0	1	0	1	0	1	
				Hex	0	10	20	30	40	50	60	70	
b3	b2	b1	b0	Dec	0	16	32	48	64	80	96	112	
0	0	0	0	0	0	NUL	TC7 DLE	SP	0	@	P	`	p
0	0	0	1	1	1	TC1 SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	2	TC2 STX	DC2	"	2	B	R	b	r
0	0	1	1	3	3	TC3 ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	4	TC4 EOT	DC4	\$	4	D	T	d	t
0	1	0	1	5	5	TC5 ENQ	TC8 NA K	%	5	E	U	e	u
0	1	1	0	6	6	TC6 ACK	TC9 SYN K	&	6	F	V	f	v
0	1	1	1	7	7	BEL	TC1 ETB	'	7	G	W	g	w
1	0	0	0	8	8	FEO BS	CA N	(8	H	X	h	x
1	0	0	1	9	9	FE1 HT	EM)	9	I	Y	i	y
1	0	1	0	A	10	FE2 LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	11	FE3 VT	ESC	+	;	K	[k	{
1	1	0	0	C	12	FE4 FF	IS4 FS	,	<	L	\	l	l
1	1	0	1	D	13	FE5 CR	IS3 GS	-	=	M]	m	}
1	1	1	0	E	14	SO	IS2 RS	.	>	N	^	n	
1	1	1	1	F	15	SI	IS1 US	/	?	O	-	o	DEL

Figure 4.12 Binary numbers can represent characters. This table shows the most common coding, known as ASCII.

Binary				b7	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	
				b6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
				b5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
				b4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
				Hex	0	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	
				Dec	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	
b3	b2	b1	b0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	NUL			SP	&											0
0	0	0	1	1	1					/			a	j			A	J			1
0	0	1	0	2	2								b	k	s		B	K	S		2
0	0	1	1	3	3								c	l	t		C	L	T		3
0	1	0	0	4	4								d	m	u		D	M	U		4
0	1	0	1	5	5	HT	NL	LF					e	n	v		E	N	V		5
0	1	1	0	6	6		BS						f	o	w		F	O	W		6
0	1	1	1	7	7	DEL							g	p	x		G	P	X		7
1	0	0	0	8	8								h	q	y		H	Q	Y		8
1	0	0	1	9	9								i	r	z		I	R	Z		9
1	0	1	0	A	10				¢	!											;
1	0	1	1	B	11				.	\$,										#
1	1	0	0	C	12				<	*	%										@
1	1	0	1	D	13				()	-										.
1	1	1	0	E	14				+	;	>										=
1	1	1	1	F	15		BEL				?										"

Figure 4.13 The EBCDIC code also was devised to encode characters as binary strings.

An Aside

Twos complement representation of integers

This method of representing negative numbers will be more fully explained in the HSC Course, but we need to know how this representation method will affect the range of integers we can have for a number of bits.

As we have seen, there are 2^{16} (that is, 65 536) different patterns of 0s and 1s when we use two bytes to represent an integer. Twos complement representation evenly divides these representations into two by representing equal numbers of positive and negative integers. That is 32 768 positive numbers and 32 768 negative numbers. Since 0 is taken to be a positive number, the range of positive numbers that can be shown in this way is 0 to 32 767. The negative numbers that can be shown are -1 to $-32 768$. Combining these ideas, this gives us a range of $-32 768$ to 32 767 as the values that can be stored in two bytes.

Floating point data type

The integer data type is not suitable for applications that require values which lie between integers; the **floating point data type** is designed to represent these values. In computing terms floating point numbers are dealt with as fractions. This is not quite the same definition as found in mathematics, but each number

which can be stored by a computer needs to be able to be represented as a finite number of digits, hence the above definition. Floating point numbers are stored in two parts: an exponent and a mantissa. The **exponent** is stored as one or more bytes and represents, as an integer, a power of two. The **mantissa** is expressed as a sum of binary digits and again is stored as a number of bytes.

For **single precision** floating point numbers, it is common practice to use one byte for the exponent and three bytes for the mantissa. **Double precision** floating point numbers consist of one byte for the exponent and seven bytes for the mantissa. In order to simplify the explanation and demonstrate the effects that the method can have on calculation, we will consider how a floating point number can be shown using only two bytes: one for the mantissa and one for the exponent.

The number 320_{10} may be stored in two bytes in the following manner: 0000 0100 0101 0000. The first byte represents the power of two (in this case 2^3 , so this byte has the same value as 8 in binary; in this case, it represents 2^8 or 256_{10}). The second byte represents the mantissa (the digits from the left to right in the mantissa represent first the sign of the mantissa in two's complement form, then decreasing powers of two from 2^0 down to 2^{-6}). The value 320_{10} is therefore represented in this form as $256 \times 1\frac{1}{4}$

The accuracy afforded by this example is low, since it is difficult to obtain an exact representation of even simple floating point numbers such as $3.\frac{1}{10}$

The larger the number of bytes allocated to the mantissa of the floating point number, the better the accuracy. The representation in this simple example has a range of $1.984\ 375 \times 2^{127}$ to $1.984\ 375 \times 2^{-127}$, 0 and $-1.984\ 375 \times 2^{127}$ to $-1.984\ 375 \times 2^{-127}$. As decimals, this range becomes $3.376\ 239\ 109 \times 10^{38}$ to $1.166\ 310\ 801 \times 10^{-38}$, 0 and $-3.376\ 239\ 109 \times 10^{38}$ to $-1.166\ 310\ 801 \times 10^{-38}$.

This range looks impressive, but due to the relatively small number of bits used to represent the mantissa, there are many values within this range that are not able to be represented exactly. Increasing the mantissa to three bytes gives a useful degree of accuracy for calculation, and this representation is in common use. A further problem with floating point numbers is that, as calculation progresses, especially in solutions with a large number of operations on the same data, the degree of accuracy of the result may be reduced due to **truncation** or **overflow**. Accuracy may be improved by rounding off results or, better still, using a larger number of bytes to represent floating point numbers. (This type is often known as double precision floating point representation.)

String data type

Data is often in the form of words. The character data structure is unsuited for this representation as each character in a word would have to be treated as a different data element. The **string data structure** is used to represent a sequence of characters, such as a word, but keeps its identity as a single data element. The abstract view of a string is as a number of adjacent locations, each containing a single character. This structure can be thought of as equivalent to writing a sequence of characters on a single line. The method of storing strings varies, but some of the more popular methods are:

- the use of one or more bytes at the beginning of the string to represent the size of the string
- setting the size of the maximum string length when declaring the variable at the beginning of a program module

- the use of an 'end of text' character to mark the end of the stored string of text.

Each of these methods has its advantages for various applications. For example, a word-processing program will create strings of varying lengths (each document may be stored as a single large string) and may use either the first or the last of these methods of storage. The word-processing program may even use the first method to write a string to disk and the third method when storing the text in the primary storage area. A database, however, may be written with string fields having a fixed length, so the second method would be suitable. These decisions are not usually made by the programmer but are built into the programming language being used.

Date and currency formats

Two special simple data types appear regularly in the solution of problems: date and currency.

A number of different methods of storing a **date** have been devised, each with its own merits. All need to incorporate the day, the month and the year. Some applications may also need the day of the week, but since it is possible to calculate that from the day, the month and the year there is no need to store it. Some storage methods use the following methods:

- Separate bytes are used to store the day, the month and the year. (This led to the so-called Millennium Bug at the end of 1999 as the year was stored as a two-digit value to save on memory, which was expensive during the early period of the computer revolution.)
- The date is stored as a serial number which counts the number of days from a particular known date (for example 1 January 1900). This method is useful as it makes the calculation of periods between dates very simple. Also, by using a floating point number, a time and date can be represented by the one serial number.

Currency may be represented by a floating point number stored as described earlier. However, this method is not an efficient way of doing it as there is a fixed number of digits after the decimal point (usually two). Thus we can think of the currency data type as being different. Storage of these data items can be improved by using a byte to represent the cents and two or three bytes to represent the dollars. In this way the cents can be represented exactly, unlike using a floating point representation where the cents would be represented approximately.

Structured data types

The simple data structures discussed above have limited usefulness. In some situations the use of simple data structures with related data items is unwieldy since each item needs to be accessed by a unique **identifier**. For example, the average maximum temperatures in Sydney for each of the twelve months of the year are related data items in that they are temperatures, but each item is different as it relates to a particular month. A convenient way of representing this data would be to name the collection of data as an identifier (for example TEMPERATURE) and use some way to identify each of the items individually. The structured data types called arrays and records overcome this problem by allowing data to show logical relationships by using the same identifier.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
25.6	25.6	24.4	21.7	18.9	16.1	15.6	17.2	19.4	21.7	23.3	25.0

Figure 4.14 Representation of a one-dimensional array to hold average maximum monthly temperature.

Arrays

The array is the easiest of the data structures to understand as it relates data of the same type. An **array** can be thought of as a number of separate storage locations arranged in a pattern. The simplest of these is a straight line, rather like a line of numbered boxes, with each data item being stored in its own numbered box. The data items can then be accessed by naming the box that it occupies using a number called an **index**. For example, Sydney's average April temperature is stored in box number 4 and so may be accessed from that box. This structure is known as a **one-dimensional array**. A one-dimensional array extends the use of a variable by allowing it to be used for a number of related data items.

Different languages define arrays in differing ways. However, the array identifier together with the indexing must be included for the interpreter to set aside the appropriate amount of memory for storage. We can illustrate how two different languages handle the definition of an array, using the one-dimensional array illustrated in Figure 4.14. In BASIC the array is declared by the statement `DIM AVERAGE_TEMP(12)`. In Pascal the array is identified in the declaration part of the program by the statement `temperature = array[1..12] of real;` appearing in the type declaration and the statement `average_temp : temperature;` being placed in the variable declaration.

Records

Arrays are used to store multiple data items of the same type. However, we often wish to link items which are related but are of different types, and the record structure is used for this purpose. A **record** consists of a number of data items, called **fields**, which may be logically linked. Individual fields within a record can be accessed by naming both the record and the required field. For example, the details of an employee will contain differing data elements such as the surname, first name, employee number, pay rate, number of years of service and department identification. These elements are linked by the fact that they hold data which identifies certain features about the employee. However, processing would be difficult if each of these fields was stored using the same data type, for example strings, since some of the fields are integers (for example the employee number), others may be represented by floating point numbers (for example the pay rate), and it may be possible to represent others by characters (for example the department). A major area in which records are used is that of database management; the terms record and field are both identified within this context.

Since a single record contains a number of related data elements, its use is limited. But when the record data structure is combined with the array structure, a very powerful tool is created to use in data processing. An array of records gives a programmer the opportunity to create large databases with a structure that is easily represented on paper.

As with arrays, the interpreter needs to be able to allocate memory to a record, so it needs to be declared. In Pascal the array declaration is made in the following manner:

```
type
  name = packed array[1..20] of char;
  person =
    record
      surname : name;
      first_name : name;
      employee_number : integer;
      pay_rate : real;
      department : char;
    end;
var
  employee : person;
```

Files

The term 'file' was used well before the advent of computer technology to describe a set of papers arranged in such a way that they could be easily referenced. In computing, the term **file** is generally used to describe a collection of data held in a secondary storage device. A file may well bring together a number of different data items in various forms. For example, a markbook file may consist of a string representing a class name, an array of records which represent students' names together with the results of a number of assessment tasks, and a one-dimensional array of floating point numbers which represent the average score for each of the assessment tasks. However, a file is typically a collection of data subdivided into records, each of these records, in turn, consisting of fields. The physical structure of a stored file (in terms of sectors and blocks) is quite different from the logical file structure that the programmer develops to represent the data in the problem.

Some sophisticated **database management systems** allow users to create their own applications. These systems can be categorised as fourth-generation. The file structure used in such systems can be complex, especially where **relational databases** are concerned. A relational database management system can be used to link several files by using one or more fields common to each file. For example, one school administration file may consist of student information, whereas another may consist of class information. If each student is identified by a student number, this field can then be used in the class file to identify the student. A further benefit is that it provides a link with all the other data relating to the student. If a student's details change, the link allows that change to be reflected in the linked data too. A program called a **report generator** can be used to create output from the linked files.

The simplest form of file structure is one known as a **sequential file**. In this structure all records are stored in a way that allows access to a record only as long as all previous records have been processed. This is very similar to the storage of a number of songs on an audio tape, where to listen to the third song on the tape you must at least pass by the first and second songs (assuming that you are starting from the beginning of the tape). The end of a sequential file is always indicated by some form of 'marker', known as an **end-of-file (EOF) mark**.

The most common uses for this are those in which all data elements of the structure are processed at each execution of a program and the order of processing is irrelevant in providing the correct output.

For example, a company's pay records may be stored as a sequential file. In this structure, each data element is processed in turn, as each of an employee's details is entered for the new pay period. In this case it does not matter when an employee's pay is processed in the sequence as long as it is done.

Exercise 4.4

- 1 Complete each of the following statements with the most appropriate word from the list (one of them is used twice):

abstract, array, Boolean, character, exponent, field, index, mantissa, physical, record, sequential, string

- a The _____ structure of data is independent of the computer system in which it is stored.
- b The storage of data as a magnetic pattern on a disk is an example of the _____ structure of stored data.
- c The _____ data structure could be represented as it within a computer system.
- d Data of the same type can be represented as a(n) _____.
- e Related data with different data types may be represented by a(n) _____.
- f Each element within an array may be individually accessed by using a(n) _____.
- g A is the smallest unit of data normally used by a person.
- h A file in which data items must be accessed in order is known as a _____ file.
- i A floating point number is often represented by two parts: an _____ and a _____.
- j A _____ can be pictured as a one-dimensional array of _____s.
- k Each individual data item within a record is known as a _____.

- 2 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

A _____ data type contains one data item per identifier. The most common of these types are _____, _____, _____, _____ and _____. Two other possible simple data types are _____ and _____. When related data items are to be stored, we use a data type. The simplest of these data types are the _____ and the _____. _____ contain related data items of the same data type and _____ contain related data items which may be of a different data type.

- 3 The following two bytes are stored in a computer:

0100 1000 0110 1001

Determine their interpretation as Boolean, ASCII character and integer data types.

You may like to use a spreadsheet to help you with this activity. Save your file as EX404Q3.

- 4 What data type would be suitable for each of the following data items?
 - a the height of a person
 - b the birthday of a friend
 - c the address of a company
 - d the day's takings of a shop
 - e the number of litres of fuel used by a truck each day for a month
 - f the sex of a zoo animal
 - g the number of ice-creams sold in a day from an ice-cream van
 - h the name, address, telephone number and age of a friend
 - 5 Chris wishes to write a computer program to store her birthday book and has decided to use an array of records to store the data. Name suitable fields for her to use and the simple data type which would be used to represent each of them.
 - 6 Explain, using your own words, the differences between an array and a record. Use a word processor to prepare your answer and save the file as EX404Q6.
 - 7 Why would a programmer describe data in an abstract form in preference to describing it according to its physical characteristics?
-

Structured algorithms

The algorithm

An **algorithm** is a finite set of steps, with a single beginning and a single end, which solves a problem. This means that when we decide to design a solution to a problem, we can enter the algorithm at one place only and leave at one place only.

Designing an algorithm allows a problem to be solved by breaking it down into steps. It can be used to solve many types of problems and is an important part of the development of a computer-based system. In this section we examine algorithms from a familiar but non-computing context.

Algorithms can be used to describe simple daily actions. For example:

Algorithm to make a phone call:

- Pick up the phone.
- Dial number.
- Deliver the message.
- Hang up the phone.

Algorithm to catch a train:

- Travel to station.
- Buy a ticket.
- Go to the correct platform.
- If the correct train, catch the train; otherwise wait.
- Repeat the above step.

Algorithms are commonly used in recipes, do-it-yourself instructions, and knitting manuals. Here are some examples:

Algorithm to make lemonade:

- Fill a tall glass with cracked ice.
- Add lemon juice and sugar.
- Shake and pour unstrained into glass.
- Top with water.
- Add a slice of lemon.
- Serve with drinking straws.

Algorithm to make French onion soup:

- Peel onion and chop finely.
- Fry onion in butter, covered, on a very low heat for 20 minutes.
- Add water, crumbled beef stock cubes, and mustard.
- Simmer for 30 minutes.
- Spoon into heatproof dishes.
- Sprinkle grated cheese on top.
- Put under griller until cheese melts.

Algorithm to install a smoke alarm:

- Select a central location such as the hallway.
- Select a wall or ceiling on which to mount the alarm.
- Unlatch and remove the mounting plate from the alarm.
- Use the screws to secure mounting plate to wall or ceiling.
- Install batteries into alarm body.
- Hook the alarm body onto the mounting plate.

In all of these examples the algorithm presents a solution in a definite number of steps, with each step short enough for it to be easily carried out. For example, the algorithm to make lemonade takes six steps. Furthermore, for the algorithm to work in all situations the steps must be performed in a particular order. The French onion soup would not taste very nice if the water, stock cubes and mustard were added after the cheese.

Exercise 4.5

1 Copy and complete the following sentences:

- a An algorithm is a series of _____ which when performed correctly will solve a problem in a finite time.
- b Algorithms can be used to describe simple _____ that you do each day.
- c Algorithm description is commonly used in _____, do-it-yourself manuals, and knitting manuals.
- d Each step in the algorithm must be _____ enough so that it can be easily carried out.
- e For the algorithm to work in all situations, the steps must be performed in a particular _____.
- f Before an algorithm can be written, the _____ must be fully understood.
- g How you arrive at a solution depends very much on past _____.
- h After the algorithm has been written it needs to be _____.
- i If the results are _____ the algorithm must be modified or discarded.
- j There is usually more than one algorithm _____.

- 2 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

An _____ consists of a _____ number of steps which, when followed, will _____ a problem. An _____ must have a definite _____ and a single _____. The _____ of an algorithm must be performed in a particular order for it to work.

- 3 Why do we create algorithms?
- 4 What type of problems can an algorithm solve?
- 5 Where are algorithms commonly used?
- 6 What needs to be done before the algorithm is written?
- 7 What affects how you arrive at a solution to a problem?
- 8 What needs to be done after the algorithm has been written?
- 9 The following algorithms have errors in their sequence. Find these errors and write the correct solution to the problem.
- a Algorithm to read a book:
Open to first page.
Close book.
Get book.
Read book.
- b Algorithm to make buttered toast:
Get butter.
Remove toast from toaster.
Get bread.
Butter toast.
Put bread in toaster.
- 10 Design an algorithm to make a cheese sandwich. The algorithm should be written using a word processor and saved with a filename of SANDWICH. Obtain a printout of your algorithm and cut out each step. Rearrange these steps and give them to another student. Ask this student to put together your steps to make a cheese sandwich. Compare the student's result with your original algorithm.

EXTENSION

- 11 Design an algorithm to listen to a CD. The algorithm should be written using a word processor and saved with a filename of HEARCD.
- 12 Design an algorithm to pump up a bicycle tyre. The algorithm should be written using a word processor and saved with a filename of PUMPTYRE.
- 13 Find an example of an algorithm such as a recipe, a do-it-yourself instruction, or a knitting pattern. Use a word processor to write a description of your algorithm in ordinary English. Call your file MYALGOR.
- 14 At home, find as many examples of different algorithms as you can. For each algorithm write down what it describes, the way it is presented (i.e. as a recipe, an instruction book or some other way) and whether it is easy to understand. Create a database with the fields 'Description', 'Presentation' and 'Ease of use', use one record per algorithm, then save the database as ALGLIST.

Methods of algorithm description

Algorithms can be represented in a number of different ways. These are referred to as methods of algorithm description. There are many methods, such as pseudocode, flowcharts, English prose, structured English, Nassi–Schneiderman diagrams, HIPO, structure charts and decision tables. In this course, **pseudocode** and **flowcharts** are the approved methods of algorithm description.

Pseudocode

Pseudocode is one method of algorithm description involving the use of English. It relies on indenting lines and using keywords to highlight the structure of the algorithm. Pseudocode is very popular because it is in text form and can be easily modified using a word processor. Furthermore, many of its keywords are similar to those in programming languages such as Pascal.

Even though different standards of pseudocode have been established for different purposes, there are some generally accepted rules. The flow of control in pseudocode is always from the top to the bottom. The keywords are highlighted in capital letters or bold type to emphasise them and to indicate the type of action being performed. The most common keywords are shown in Table 4.4. These keywords are grouped in pairs. For example, for every BEGIN there is an END, and for every IF there is an ENDIF. Indentation is used to show the structure of the algorithm.

Keywords	Meaning
BEGIN END	Terminal: start and finish.
IF (condition) THEN process 1 ELSE process 2 ENDIF	Selection: different tasks are performed according to the condition. The ELSE statement is optional. (binary selection)
CASEWHEN (condition) (value 1: process 1 (value 2: process 2 OTHERWISE: process 3 ENDCASE	Selection: many tasks are performed according to the condition. The OTHERWISE statement is optional. (multiway selection)
WHILE (condition) process 1 ENDWHILE	Repetition: statements between the keywords are repeated while the condition is true. (pre-test repetition)
REPEAT process 1 UNTIL (condition is true. (post-test repetition)	Repetition: statements between the keywords are repeated while the condition is true. (post-test repetition)

Table 4.4 Some of the common keywords used in pseudocode.

Consider an example of an algorithm written in pseudocode.
Problem: Design an algorithm to describe making a cup of coffee.

Pseudocode

```
BEGIN make_coffee
  Fill a kettle with water.
  Boil the water in the kettle.
  Add coffee to the cup.
  Pour boiling water into the cup.
END make_coffee
```

Flowcharts

A flowchart is a pictorial or graphical method of describing an algorithm. Flowcharts are often favoured because it is easier to follow the structure in a picture than in words. On the other hand, it is very easy to draw a flowchart that is complex and difficult to change into a programming language.

The basic elements of a flowchart are a set of symbols that contain messages, and interconnecting lines with arrows. A set of standards for flowcharts has been established for a number of different applications. The four most commonly used symbols are shown in Table 4.5.

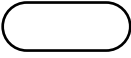



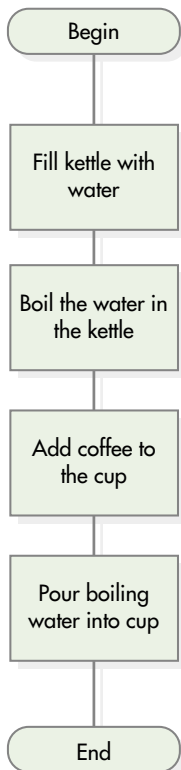
Symbol	Meaning
	Terminal: start and finish. There should be only one line in, or one line out.
	Process description of a process or action. It has only one line in and one line out.
	Subprogram process described by its own flowchart to perform a particular task.
	Decision: it has one line coming in at the top, and two lines leaving it.

Table 4.5 Some of the common symbols used in flowcharts.



The Australian standards for flowcharting indicate that the main direction of flow is accepted as being top to bottom or left to right. This flow of control is indicated by lines and arrows called **flowlines**. Flowlines do not need an arrow if the flow of control follows these main directions. For example, in Figure 4.15 the flow of control is from top to bottom, so there are no arrows on the flowlines.

Figure 4.15 Flowchart of algorithm for making a cup of coffee.

Exercise 4.6

- 1 For each statement, select a matching word or phrase from the following list:
algorithm, decision, flowchart, flowlines, indentation, keywords, pseudocode, terminal, word processor
 - a One method of algorithm description, involving the use of English, that relies on keywords and indentation.
 - b Software application used to modify algorithms written in pseudocode.
 - c Highlighted in capital letters or in bold type to indicate the type of action being performed.
 - d Used to show structure in pseudocode.
 - e A pictorial or graphical method of describing an algorithm.
 - f Lines and arrows used in flowcharts to indicate the flow of control.
 - g A series of steps which, when performed correctly, will solve a problem in a finite time.
 - h A symbol used in a flowchart to indicate the start and the finish.
 - i A symbol used in a flowchart to indicate selection.
- 2 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Some of the methods for algorithm description used today are _____, _____, _____ and _____. The two methods approved for this course are _____ and _____. The graphical method is the _____. The method which uses English-like language is called _____. When using flowcharts, we read them from _____ to _____ and _____ to _____, moving from step to step along _____. _____ is popular because its _____ are close to those in programming languages.
- 3 What is pseudocode?
- 4 Why is pseudocode very popular?
- 5 Where is the flow of control in pseudocode?
- 6 Why are keywords highlighted in pseudocode?
- 7 What is the purpose of indentation in pseudocode?
- 8 What is a flowchart?
- 9 Why are flowcharts favoured as a method of algorithm description?
- 10 What are the basic elements of a flowchart?
- 11 Use a drawing program to create a set of basic flowchart symbols. Save these with the filename FLOWSYMB. You can then copy the symbols from this file when you need to draw a flowchart.
- 12 What is the main direction of flow in flowcharts?

- 13 An algorithm to make a phone call is shown below:
 Pick up the phone.
 Dial number.
 Deliver the message.
 Hang up the phone.
- Convert the above algorithm into pseudocode. The algorithm should be designed using a word processor and saved with a filename of PHONEPSEU.
 - Convert the above algorithm into a flowchart. The algorithm should be designed using a drawing program and saved with a filename of PHONEFLOW.

EXTENSION

- Design an algorithm in pseudocode to fry an egg.
- Design an algorithm in pseudocode to wash a car.
- A set of standards for flowcharts has been established for a number of different applications. Find an example of a standard set of flowchart symbols.

Standard constructs

Algorithms are composed of **standard constructs** or **control structures**: a sequence of instructions, a selection between two or more alternative courses of action, and a repetition of a set of instructions a number of times (an iteration). Programming languages, irrespective of their generations, will implement these constructs. The languages may be represented in varying ways but the constructs will always be present. They are the building blocks of an algorithm.

Sequence

The simplest of the three standard constructs is the **sequence**, which is a set of instructions following one after the other, rather like a simple recipe. Table 4.6

Pseudocode	Flowchart
BEGIN maincode instruction 1 instruction 2 instruction 3 instruction 4 instruction 5 • • • END maincode	<pre> graph TD Begin([Begin]) --> I1[Instruction 1] I1 --> I2[Instruction 2] I2 --> I3[Instruction 3] I3 --> I4[Instruction 4] I4 -.-> End([End]) </pre>

Table 4.6 Representation of a sequence in pseudocode and as a flowchart.

shows how the sequence is represented in the algorithm description methods. A sequence can be easily identified since the flow will pass through every instruction of the sequence on all occasions.

In pseudocode, the steps are placed between BEGIN and END (see the problem below). The sequence of four steps is indented to show the structure and to improve the readability of the algorithm. The reason will become clear as the algorithms become more complex. Finally, note that the flow of control is from top to bottom, starting at the first step and finishing at the last step.

Problem: Design an algorithm to describe buying a pair of shoes.

Pseudocode

```

BEGIN buyingshoes
  Go to shop.
  Find shoes.
  Pay for shoes.
  Leave shop.
END buyingshoes
  
```

The flowchart always starts and finishes with a terminal symbol (an oval). The steps are placed between these symbols and joined by flowlines (see Figure 4.16). Each step in this problem is an action and is represented by a process symbol (a rectangle). Furthermore, the direction of flow is down the page between the terminal symbols.

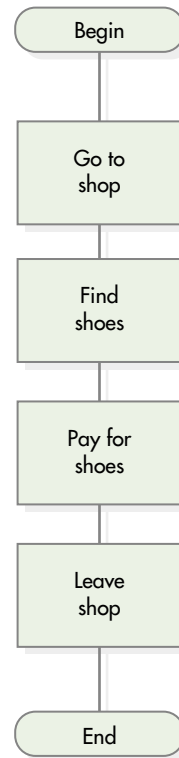


Figure 4.16 A flowchart for buying a pair of shoes.

Exercise 4.7

- Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.
The simplest _____ structure is the _____. The direction of flow passes through _____ steps in this structure. In pseudocode, a _____ will always be _____ the same amount. As a flowchart, the _____ is identified as a number of steps flowing the _____.
- What are the three control structures used in all algorithms?
- What is a sequence in an algorithm? Illustrate your answer with examples.
- How does an algorithm in pseudocode start and finish?

- 5 What is the advantage of indenting steps in pseudocode?
- 6 How does a flowchart start and finish?
- 7 What does the process symbol in a flowchart represent?
- 8 Why is the direction of flow for pseudocode and flowchart from top to bottom?
- 9 Rearrange into the correct sequence the following steps to cook a piece of steak:
 - Put steak into frying pan.
 - Remove steak when cooked.
 - Heat oil in frying pan.
 - Turn steak as required.
 - Get steak and oil.Use a word processor to prepare your answer and save the file as STEAKALG.
- 10 Rearrange into the correct sequence the following steps to sharpen a pencil:
 - Turn pencil.
 - Get pencil and sharpener.
 - Put pencil into sharpener.
 - Remove pencil from sharpener.Use a word processor to prepare your answer and save the file as SHARPEN.
- 11 Use the steps in question 10 to design an algorithm to sharpen a wooden pencil:
 - a in pseudocode
 - b as a flowchartYou can use a word processor for part a, saving your file as SHARPENP, and a drawing program for part b, saving your file as SHARPENF. (Don't forget to use the symbols you saved in Exercise 4.6 as the file named FLOWSYMB.)
- 12 Rearrange into their correct sequence the following steps to make cornflakes for breakfast:
 - Pour milk over cornflakes as required.
 - Get bowl, cornflakes, milk and sugar.
 - Put sugar over cornflakes as required.
 - Fill bowl with cornflakes.
- 13 Using the steps in question 12, design an algorithm to make cornflakes:
 - a in pseudocode
 - b as a flowchartYou can use a word processor for part a, saving your file as CORNFLP, and a drawing program for part b, saving your file as CORNFLF. (Don't forget to use the symbols you saved in Exercise 4.6 as the file named FLOWSYMB.)
- 14 Rearrange into their correct sequence the following steps to wash your hands:
 - Rinse off soap.
 - Turn on water.
 - Dry hands.
 - Clean hands with soap.
 - Wet hands.
 - Turn off water.
- 15 Using the steps in question 14, design an algorithm to wash your hands:
 - a in pseudocode
 - b as a flowchart

You can use a word processor for part **a**, saving your file as WASHP, and a drawing program for part **b**, saving your file as WASHF. (Don't forget to use the symbols you saved in Exercise 4.6 as the file named FLOWSYMB.)

EXTENSION

You may like to use a word processor and drawing program for the following. If you do, don't forget to save your work with appropriate filenames.

- 16 Design an algorithm in pseudocode to make a glass of cordial.
 - 17 Design an algorithm as a flowchart to run an application on the school computer.
 - 18 Find one application where an algorithm does not have a sequence construct.
-

Selection

A popular feature of computers is their ability to 'make a decision' and act accordingly. The process of decision making is known as **selection**; a particular course of action is followed according to the data at a particular time and a set of selection rules. Selection is a standard construct to determine which particular step (from a set of steps) is to be executed next. There are many situations where the normal sequence of one step followed by the next is not appropriate. Using selection, a condition such as a question can be given and, depending on the answer, different paths can be followed. There are two types of selection: binary selection and multiway selection.

Binary selection

The simplest selection structure will allow for two courses of action. This is known as a **binary selection**. The most common binary selection construction is of the form 'IF condition THEN action ELSE alternative action'.

Binary selection allows a choice between two options. If a condition (circumstance) is met, then one path is taken; otherwise the second path is followed.

In pseudocode the keywords used for binary selection are IF...THEN (see the problem below). The condition is put after the IF keyword. There are only two possible answers to the condition: true or false. If the condition is true, the process after the THEN keyword will be carried out or executed. If the condition is false, the flow of control moves to the next step and ignores the process(es) after the THEN keyword. The ENDIF indicates the finish of binary selection.

Pseudocode

```
IF condition THEN
  process
ENDIF
```

In the following problem, the condition is whether it is raining or not, and the process (if it is raining) is to take an umbrella.

Problem: Design an algorithm to describe deciding when to take an umbrella.

Pseudocode

```
BEGIN MAINPROGRAM
  Check weather
  IF it is raining THEN
    take an umbrella.
  ENDIF
END MAINPROGRAM
```

In a flowchart the selection is made using a decision symbol (a diamond). The condition is placed inside this symbol and the answer must be true or false (see Figure 4.17). These answers are placed on opposite sides of the decision symbol to indicate the two possible paths. In this problem, if the condition is true, the direction of flow is to the right, and a process (take an umbrella) is executed. If the condition is false, the direction of flow is to the left, and the process is ignored. It is very important for the two flowlines from the decision symbol to be labelled with true or false to determine which path to follow. The two flowlines join to complete the binary selection.

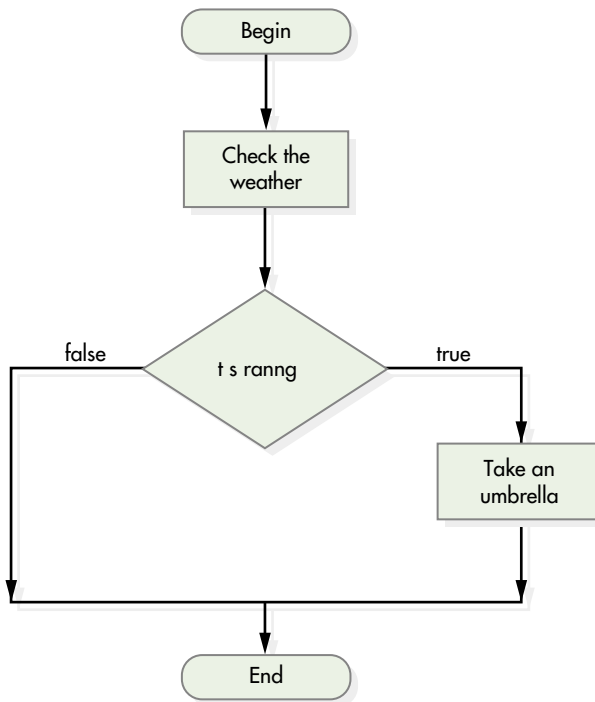


Figure 4.17 Flowchart for deciding when to take an umbrella.

The binary selection in pseudocode can be extended using the keywords of IF...THEN...ELSE (see the problem below). There are still two possible paths based on the condition after the IF keyword. Furthermore, the process after the THEN keyword still takes place if the condition is true. However, if the condition is false, the process after the ELSE keyword is executed.

Pseudocode

```
IF condition THEN
    process_1
ELSE
    process_2
ENDIF
```

In the following problem, if the signal is green, you pass through the traffic lights. If the signal is not green, you stop the vehicle.

Problem: Design an algorithm to follow when approaching a set of traffic lights.

Pseudocode

```
BEGIN MAINPROGRAM
  IF signal is green THEN
    pass through traffic lights.
  ELSE
    stop the vehicle.
  ENDIF
END MAINPROGRAM
```

The binary selection can also be extended using a flowchart (see Figure 4.18). The decision symbol is still used to determine the two possible paths. Furthermore, if the condition is true, the flow of control is still to the right, and a process is executed. However, if the condition is false, another process is executed. In this problem it is clear that if the signal is green you pass through the traffic lights, but if the signal is not green you stop the vehicle.

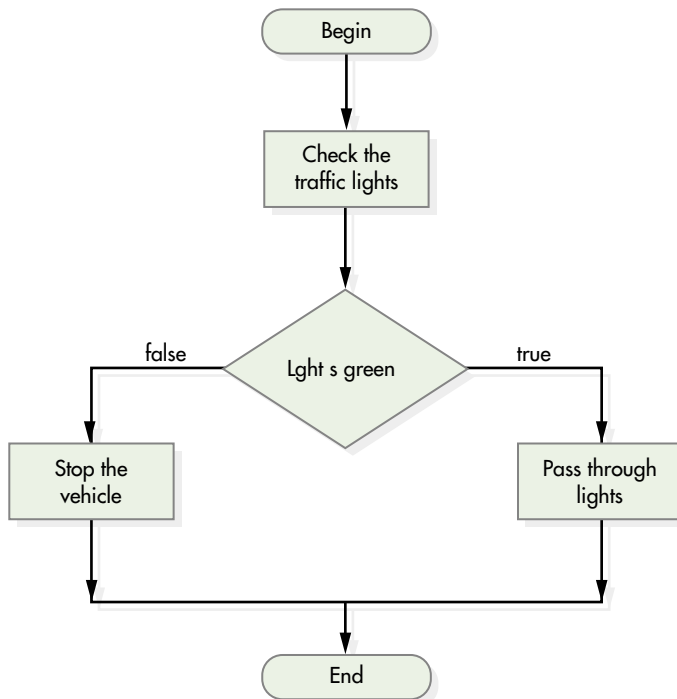


Figure 4.18 Flowchart to follow when approaching a set of traffic lights.

Multiway selection

A binary selection control structure is sufficient to enable us to design an algorithm that will function perfectly well. However, if a number of branchings need to be made at one point in the program, it becomes unwieldy to use a number of binary selections to perform that task. In order to classify a data element in one decision and take the appropriate action, a different selection structure is needed. These selections are known as **multiway selections**, as shown in the pseudocode and flowchart (Table 4.7).

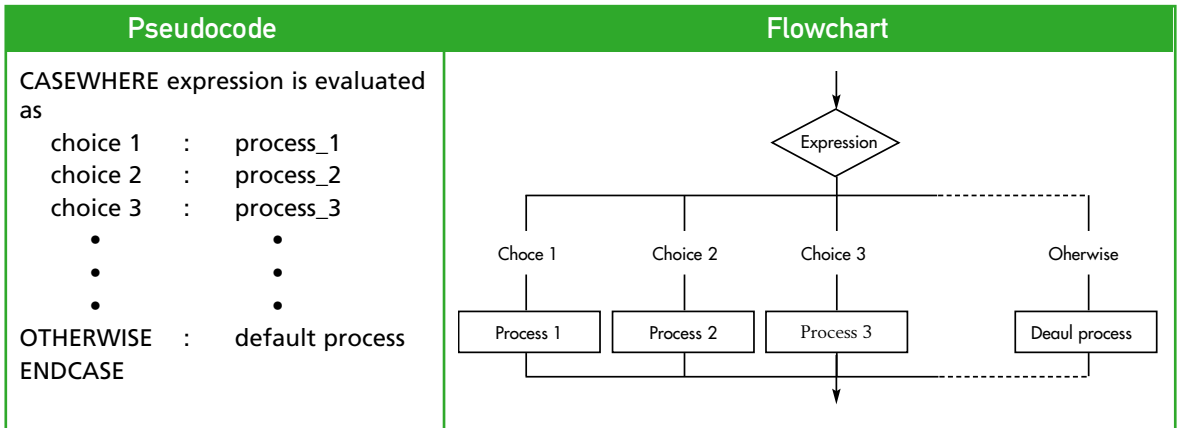


Table 4.7 Multiway selection shown in pseudocode and as a flowchart.

In pseudocode the keywords used for multiway selection (see problem below) are CASEWHERE...ENDCASE. The condition is put after the CASEWHERE keyword. Since there are many different answers or possibilities, you need to be aware of all the possibilities and provide an instruction to cater for each possibility. The possibilities are indented under the CASEWHERE keyword with a process to be executed only if the answer is true. Each data item, when compared, should make only one of the CASEWHERE choices true.

For example, if we were approaching a set of traffic lights, the signal could be red, amber, green or not working. There are four options. We could ask four questions and use binary selection to find the required action, but it is much easier to use a multiway selection (multiple selection).

In this problem there are three options: red, amber and green, with a colon specifying the process to be carried out if the option is true. The OTHERWISE keyword is used to account for any other possibility such as where the lights are not working. The OTHERWISE keyword is similar to the ELSE keyword in binary selection and is not always required.

Problem: Design an algorithm to describe a driver's response to all possible signals at a set of traffic lights.

Pseudocode

```

BEGIN MAINPROGRAM
  CASEWHERE signal is
    Red      : stop the vehicle.
    Amber    : stop the vehicle.
    Green    : pass through traffic lights.
    OTHERWISE : proceed with caution.
  ENDCASE
END MAINPROGRAM

```

In a flowchart, the multiway selection is made using a decision symbol (see Figure 4.19). The condition is placed inside this symbol and there are many different answers to this condition. These possibilities are written above the process to be executed if the possibility is true. In this problem there are four possibilities: red, amber and green, and otherwise.

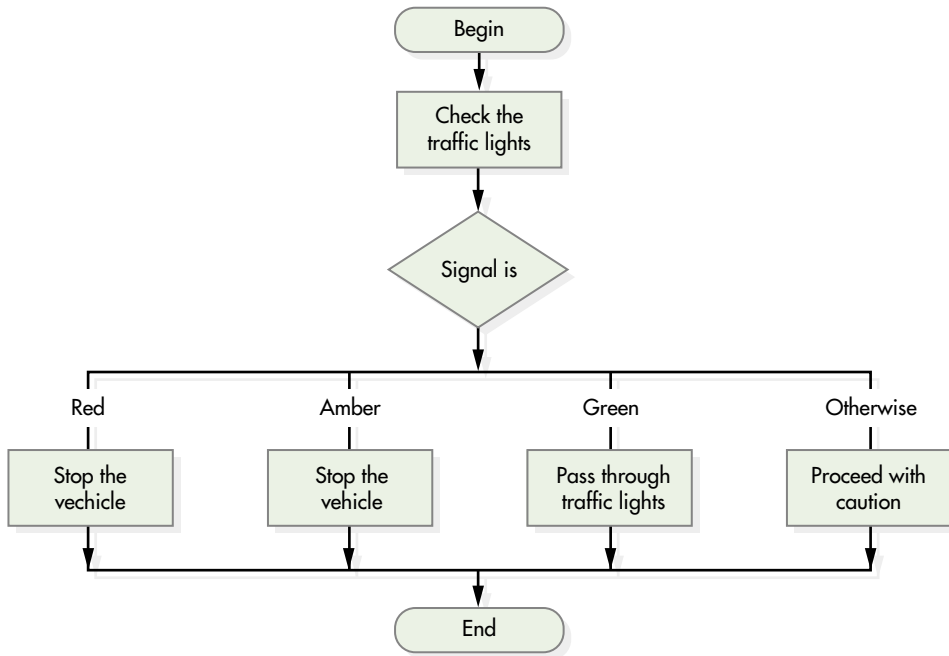


Figure 4.19 Flowchart to describe a driver's response to all possible signals at a set of traffic lights.

Exercise 4.8

- Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

When a decision is to be made in an algorithm, we have the choice of two types: the _____ and the _____. When there are only two options to choose from, we use _____. In pseudocode, these options are identified by the keywords _____ and _____. When there are more than two options we have to use a _____. It is identified in pseudocode by the keywords _____. In this type of selection we may also use the keyword _____ to allow for other data items to lead to an outcome. Each data item entering a _____ should make only one of the choices _____.

- What standard constructs are represented by the following algorithm segments?
 - IF (condition) THEN
 process
ENDIF
 - BEGIN MAINPROGRAM
 process 1
 process 2
 process 3
END MAINPROGRAM

- c CASEWHERE (condition)
 - process 1
 - process 2
 - OTHERWISE process 3
 - ENDCASE
- d IF (condition) THEN
 - process 1
 - ELSE
 - process 2
 - ENDIF
- e

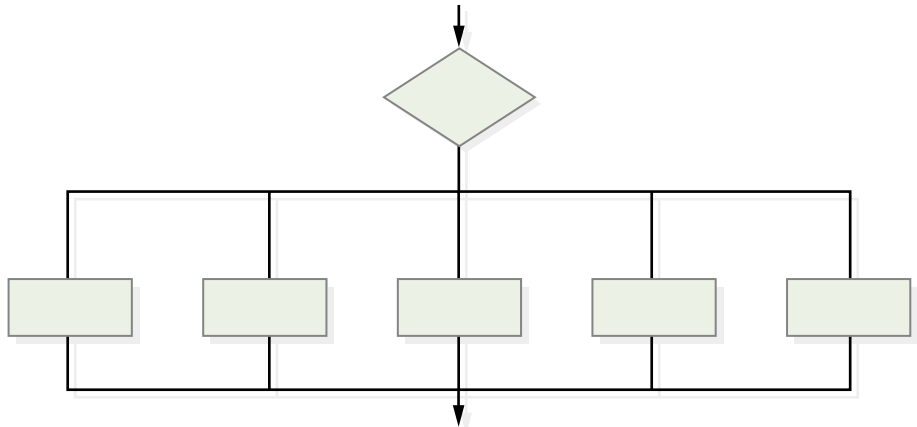


Figure 4.20

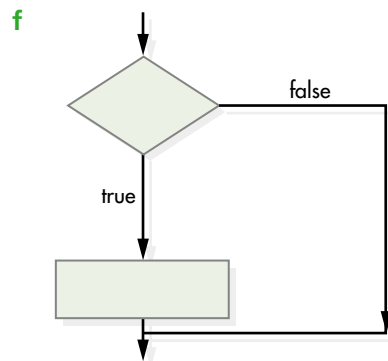


Figure 4.21

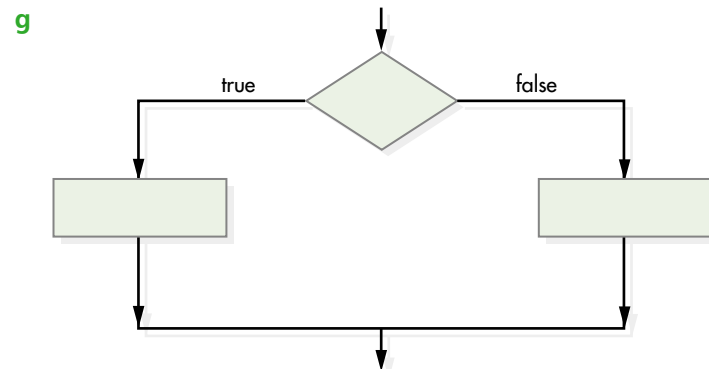


Figure 4.22

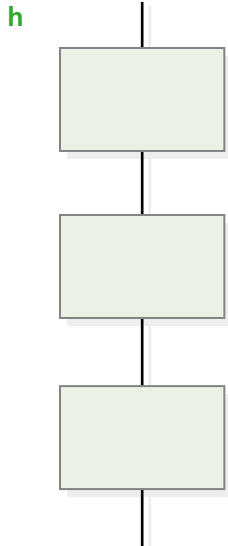


Figure 4.23

- 3 What is selection in an algorithm? Give an example to illustrate your answer.
- 4 Use examples to explain the difference between binary selection and multiway selection.
- 5 What keywords are used for binary selection in pseudocode?
- 6 Why are the answers placed on opposite sides of a decision symbol in a flowchart?
- 7 How can binary selection in pseudocode be extended so that two processes are executed?
- 8 What keyword is used for multiway selection in pseudocode?
- 9 Describe how multiway selection is made in a flowchart.
- 10 Explain what the following algorithms do?

a Pseudocode

```

BEGIN
    set number to user input
    IF number is negative THEN
        print 'Negative number—no square root'
    ELSE
        print the square root of the number
    ENDIF
END

```

b Pseudocode

```

BEGIN
    display 'What is the capital of Australia?'
    set answer to user input
    IF answer is Canberra THEN
        print 'Well done, correct.'
    ELSE
        print 'Sorry, the answer is Canberra.'
    ENDIF
END

```

c

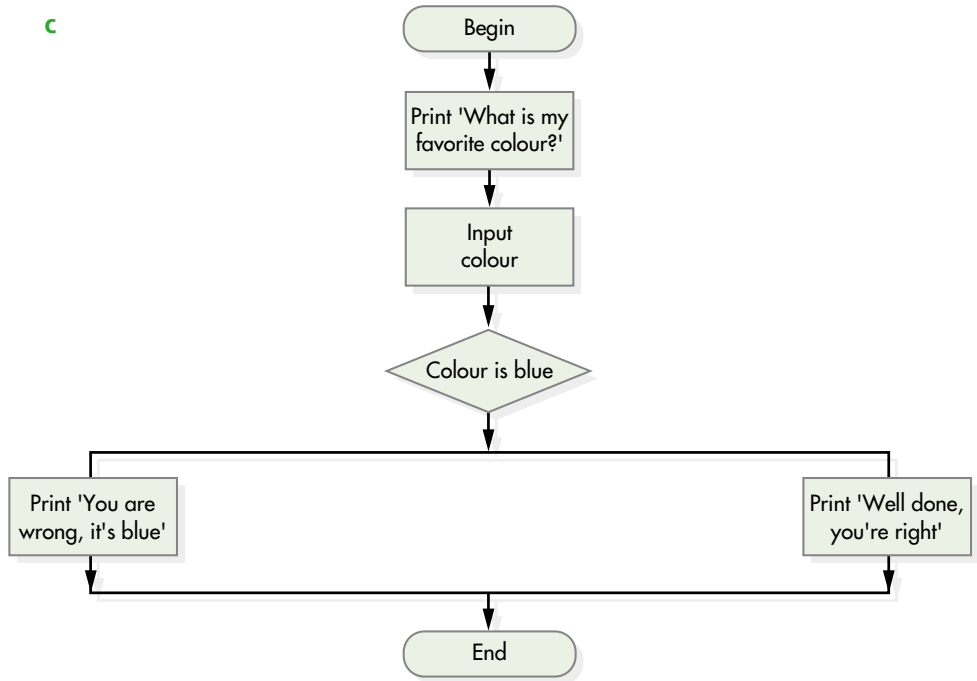


Figure 4.24

d

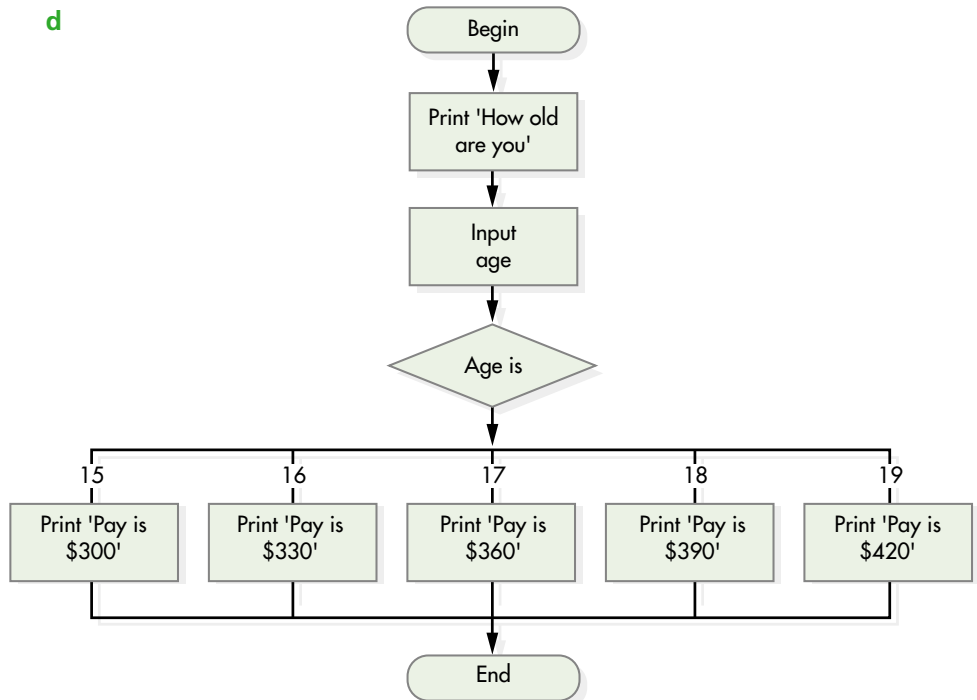


Figure 4.25

11 An algorithm to wash hands is as follows:

- Turn on the hot water tap.
- Turn on the cold water tap.
- If water is too hot, increase the amount of cold water.
- If water is too cold, increase the amount of hot water.
- Clean hands with soap.
- Rinse off soap.
- Turn off taps.

a Convert the above algorithm into pseudocode. Create your algorithm using a word processor and save it with the name WASHANDP.

b Convert the above algorithm into a flowchart. Create your algorithm using a drawing program (using your symbol file) and save it with the name WASHANDF.

12 An algorithm to make a phone call is as follows:

- Pick up the phone.
- Dial number.
- If answered, deliver message.
- Hang up the phone.

a Convert the above algorithm into pseudocode. Modify the file called PHONEPSEU (see Exercise 4.6) to represent this algorithm.

b Convert the above algorithm into a flowchart. Modify the file called PHONEFLOW (see Exercise 4.6) to represent this algorithm.

EXTENSION

13 Complete the following algorithms to determine if a particular examination score (expressed as a percentage) is a pass or a fail, and to print this result. If the score is above 50, it is a pass; if the score is below 50, it is a fail.

a

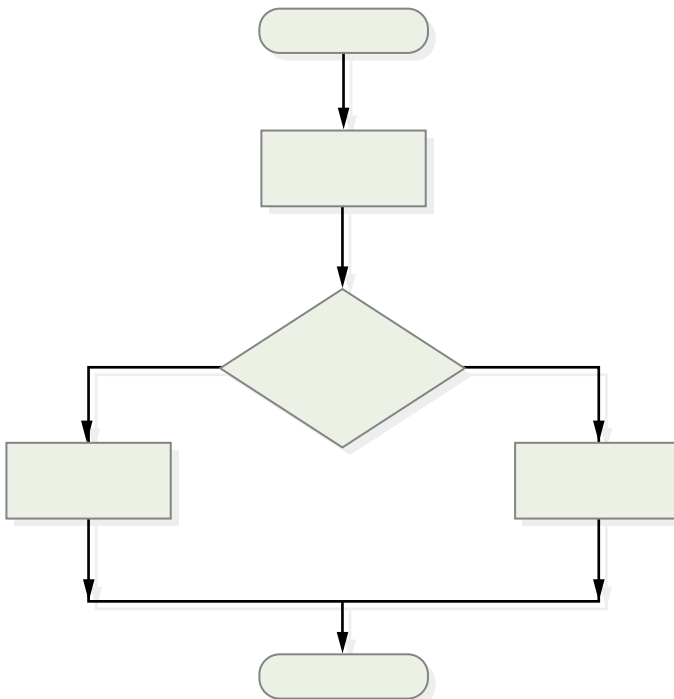


Figure 4.26

b Pseudocode

```
BEGIN
  INPUT
  IF ..... THEN
    print .....
  ELSE
    print.....
  ENDIF
END
```

- 14** Find a problem whose solution requires a selection construct. Design an algorithm to solve this problem. You may use a computer application to help you construct your algorithm description.
-

Iteration (repetition)

An **iteration** (also known as a **repetition** or **loop**) is a standard construct that allows a number of steps to be repeated until the same condition is satisfied. The steps to be repeated are referred to as the body of the loop. It is very important that each loop contain a condition to stop the loop going on forever. The condition can be checked, or tested, at the beginning or end of the loop, being respectively known as a **pre-test loop** or a **post-test loop**. A special construct known as a **counted loop** is often used when a loop has to be executed a pre-determined number of times.

Pre-test repetition

In a **pre-test repetition** or **guarded loop** the condition is checked at the very beginning of the loop before the steps to be repeated are executed.

In pseudocode the keywords used for a pre-tested repetition are WHILE...ENDWHILE (see problem below). The condition is put after the WHILE keyword. While the condition remains true, the body of the loop between the WHILE and the ENDDWHILE will be executed repeatedly. If the condition is false, control moves to the ENDDWHILE keyword and the loop is finished. To illustrate this, consider the problem below.

The guarded loop is important since the condition contained in the pre-test may prevent control from entering the loop on the first attempt. For example, a particular program is to calculate the sum of an unknown number of positive values, the end of the dataset being indicated by a negative number. If the first number in the dataset is negative, the loop should not be entered, as the entered value is not required for the sum. For this program to work properly, a pre-test has to be used in order to pass the program control around the loop.

In the following problem, if the condition that the car is travelling remains true, the 'keep seat belts on' message on will be repeatedly executed. When the car is not travelling, the repetition stops.

Problem: Design an algorithm to determine a safety procedure for travelling in a car.

Pseudocode

```
BEGIN
  WHILE car is travelling
    keep seat belt on
  ENDDWHILE
END
```


In a flowchart the pre-test repetition is made using a decision symbol and flowlines (see Figure 4.27). The condition is placed inside the decision symbol and there are two possible paths. If the condition is true, the direction of flow is down, and the body of the loop is executed. When these processes are executed, the flow of control is back to the condition in the decision symbol. If the condition is false, the repetition is complete.

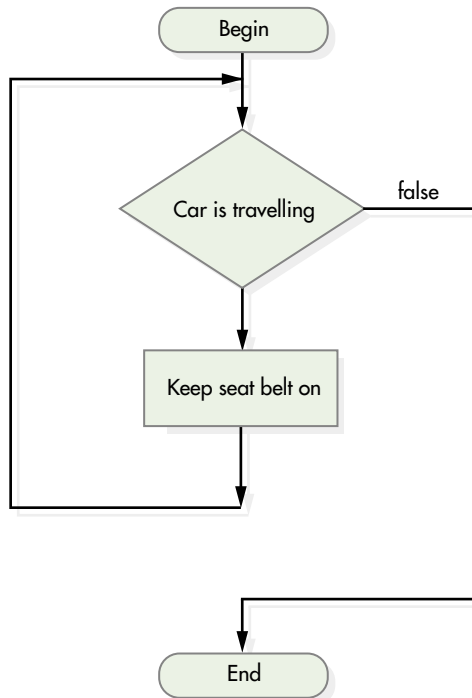


Figure 4.27 Flowchart with a pre-test repetition: to determine a safety procedure for travelling in a car.

Post-test repetition

In a **post-test repetition** or **unguarded loop** the condition is checked at the end of the loop after the steps to be repeated are executed. The important difference between a pre-test and a post-test repetition is that the body of the loop is executed at least once in a post-test repetition. A pre-test repetition if the condition was false would result in the body of the loop not being executed at all.

In pseudocode, the keywords used for a post-test repetition are REPEAT... UNTIL (see problem below). The body of the loop to be executed is placed underneath the REPEAT keyword. In the post-test repetition the condition is after the UNTIL keyword, and if it is true, the flow of control is back to the REPEAT keyword. If the condition is false, the repetition is finished.

In the following problem, the plants are watered before the condition is tested.

Problem: Design an algorithm to water plants until the ground is soaked.

Pseudocode

```

BEGIN
  REPEAT
    water plants
  UNTIL ground is soaked
END
  
```

In a flowchart the post-test repetition is made with a decision symbol and flowlines (see Figure 4.28). The body of the loop is executed before the condition is met in the decision symbol. If the condition is false, the direction of flow is up, and the body of the loop is executed again. If the condition is true, the repetition is complete.

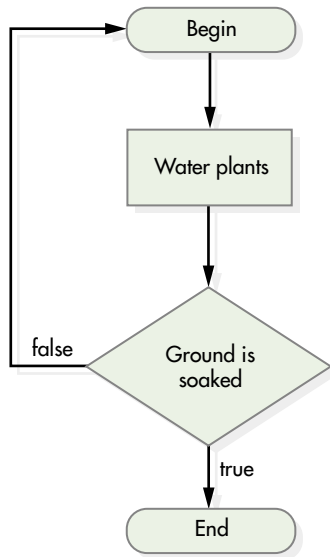


Figure 4.28 Flowchart with a post-test repetition: to water plants until the ground is soaked.

Counted loop

The two forms of iteration described above are sufficient for any algorithm. However, we often need to execute a set of steps a predetermined number of times. In this case, a counted loop using the construction: FOR identifier goes from value_1 to value_2 ... NEXT identifier. The implication here is that the value of the identifier is progressively increased from the first value mentioned up to the second value in steps of 1. For example, the following algorithm will progressively display the numbers 1 through to 10.

Pseudocode

```

BEGIN
  set end to 10
  FOR counter goes from 1 to end
    print counter
  NEXT counter
END
  
```

The main advantage of this type of construction is that it frees the programmer from having to worry about the starting and ending values of the counter and from having to put the increase of the counter in the right place to make the algorithm work correctly. This is especially important when there are loops inside loops. The following example shows how easily a set of multiplication tables can be printed by using two counted loops, one inside the other.

Pseudocode

```

BEGIN
  FOR firstnumber goes from 1 to 1
    print 'The ', firstnumber, ' tables are'
    FOR secondnumber goes from 1 to 12
      print firstnumber * secondnumber
    NEXT secondnumber
  NEXT firstnumber
END
  
```

Exercise 4.9

1 What standard constructs are represented by the following algorithm segments?

- a WHILE (condition)
 process
ENDWHILE
- b REPEAT
 process
UNTIL (condition)

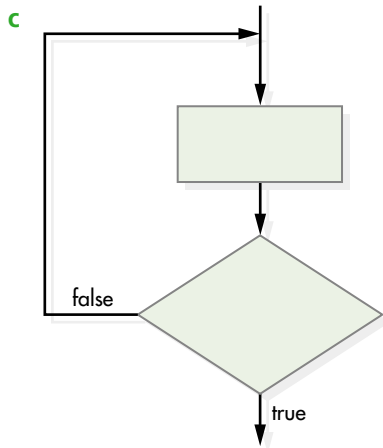


Figure 4.29

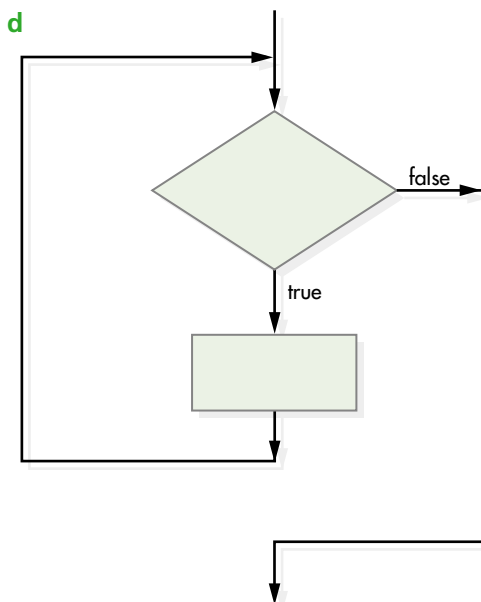


Figure 4.30

- 2 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

When a number of steps are to be executed more than once, a _____ should be put in the algorithm. There are two types of loop: the _____ oop and the _____. For convenience, we use a third type of loop called a loop when we want the steps executed a _____ number of times. Its other name is a _____ loop. The test in a _____ oop comes at the end of the loop. For a _____ loop, the test is placed at the beginning. The keywords used in pseudocode for the _____ loop are _____ and UNTIL.

- 3 What is another name for a repetition?
- 4 Why must a repetition contain a condition?
- 5 Explain the difference between a pre-test repetition and a post-test repetition. Give examples of each of the repetitions.
- 6 Describe how a counted loop works.
- 7 Draw flowcharts to represent the counted loops in the two examples on p. 132. Use the drawing program and your templates file to help you. Save your work as COUNTED1 and COUNTED2.
- 8 What keywords are used for a pre-test repetition in pseudocode?
- 9 Describe how a pre-test repetition is made in a flowchart. You may use the drawing program if you wish.
- 10 What keywords are used for a post-test repetition in pseudocode?
- 11 Describe how a post-test repetition is made in a flowchart. You may use the drawing program if you wish.
- 12 Explain what the following algorithms do.

a Pseudocode

```
BEGIN
    set count to 1
    WHILE count is less than 11
        print count
        increment count by 1
    ENDWHILE
END
```

b Pseudocode

```
BEGIN
    set count to 10
    REPEAT
        print count
        increment count by 1
    UNTIL count is 20
END
```

c Pseudocode

```
BEGIN
    set end to 52
    FOR index goes from 1 to end
        print 'Week' index
    NEXT index
END
```

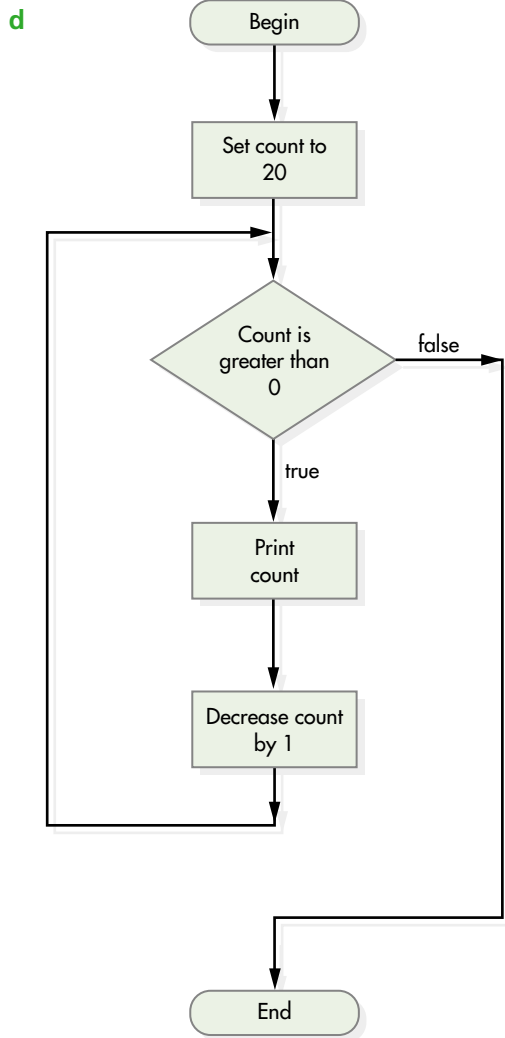


Figure 4.31

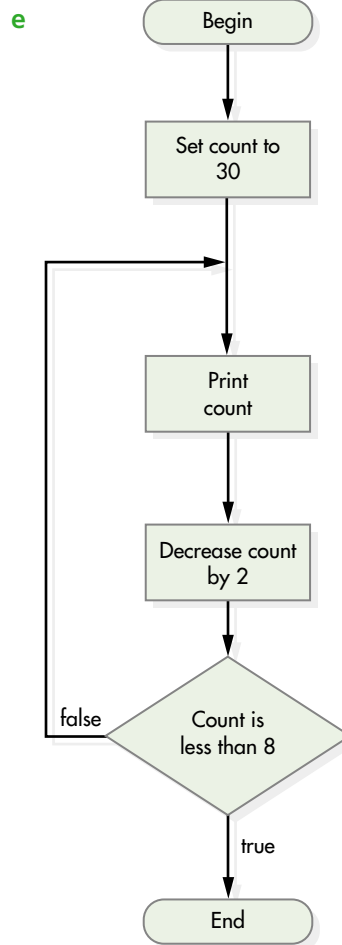


Figure 4.32

- 13 Describe the error that exists in the following algorithms as they attempt to satisfy the problem. Write the correct solution using the same standard constructs.

Problem: Design an algorithm to select a can of drink from a drink dispenser.

a Pseudocode

```

BEGIN
  remove drink
  REPEAT
    check all buttons
  UNTIL a button is pressed
END
  
```

b Pseudocode

```
BEGIN
  check all buttons
  WHILE no button has been pressed
  ENDWHILE
  check all buttons
  remove drink
END
```

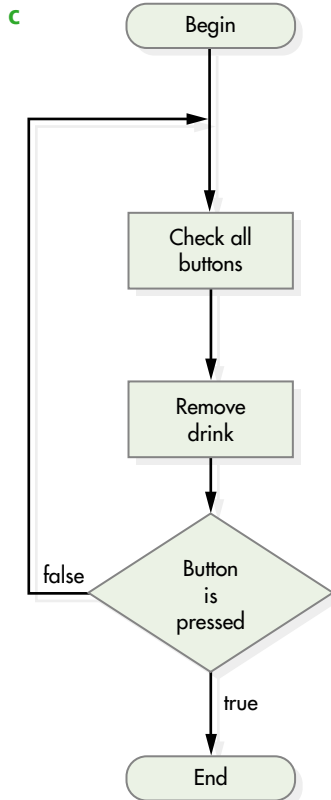


Figure 4.33

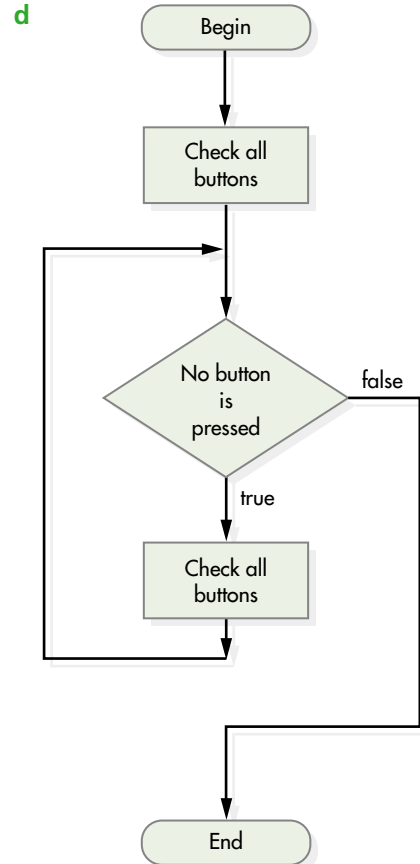


Figure 4.34

- 14** Describe the error that exists in the following algorithms as they attempt to satisfy the problem. Write the correct solution using the same standard constructs.

Problem: Design an algorithm to catch a bus.

a Pseudocode

```
BEGIN
  WHILE no bus caught
  walk to bus stop
  check all buses
  IF correct bus
  THEN catch bus
  ENDIF
  ENDWHILE
END
```

b Pseudocode

```
BEGIN
  walk to bus stop
  IF correct bus
    THEN catch bus
  ENDIF
  REPEAT
    check all buses
  UNTIL bus caught
END
```

c

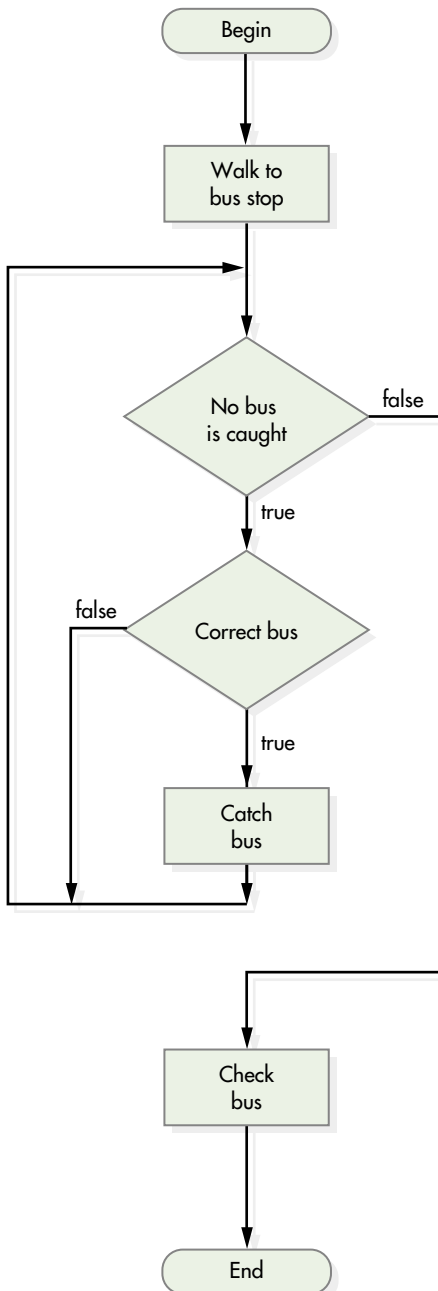


Figure 4.35

d

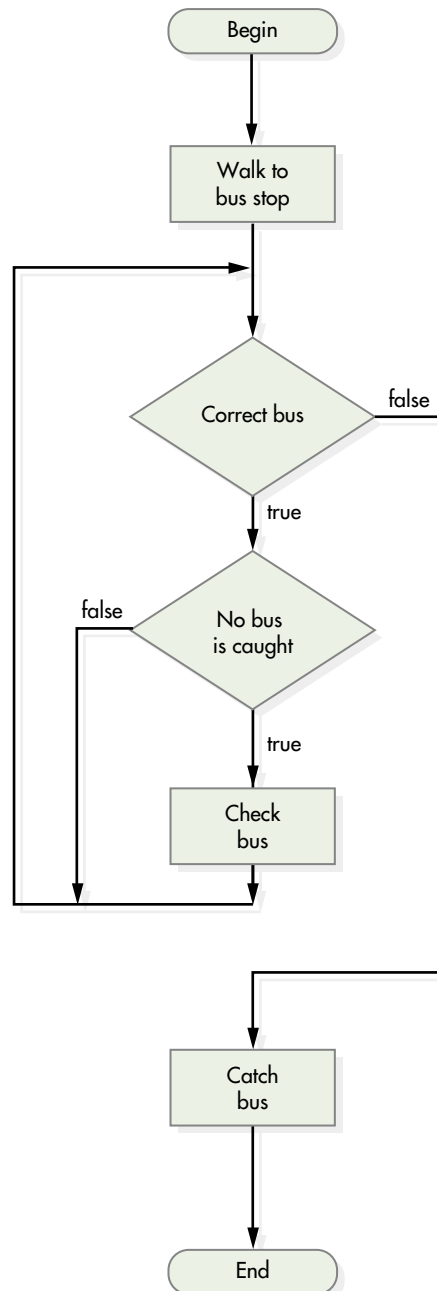


Figure 4.36

15 Design algorithms that will print out the multiples of 8 up to the number that is read.

a Pseudocode

```

BEGIN
  set multiple to 0
  set counter to . . .
  set number to user input
  REPEAT
    increase . . . by 1
    multiple equals 8 times counter
    print . . .
  . . .counter equals number
END
  
```

b

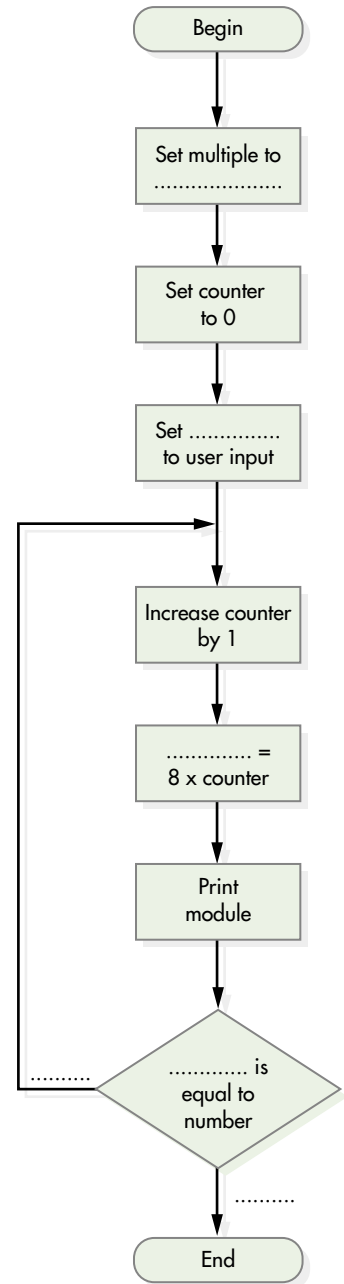


Figure 4.37

16 The following algorithm (Figure 4.38) attempts to satisfy the problem, but errors exist in the logic. Describe two errors and explain how each prevents the flow-chart from achieving its purpose.

Problem: Design an algorithm that will print a person's name, age, and whether or not they are eligible to vote (must be at least 18 years old).

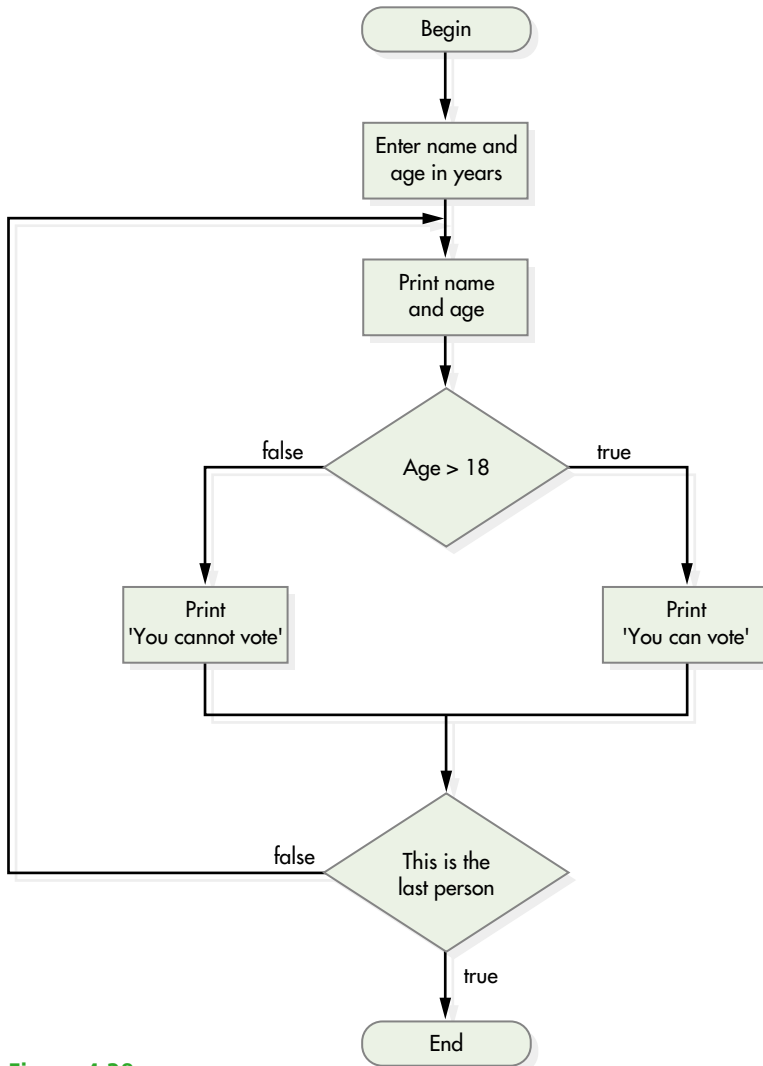


Figure 4.38

EXTENSION

17 Read the following algorithm:

```

BEGIN
  set counter to 1
  set sum to 1
  set number to user input
  REPEAT
    add counter to sum
    increase counter by 1
  UNTIL counter equals number
  print sum
END
  
```

- If the number is assigned the value 4 as a result of the 'read in' statement, what would be the result of the algorithm?
- The algorithm was meant to produce the sum of all the counting numbers up to and including the number which is 'read in'. Rewrite the algorithm so that it correctly carries out this task.

- 18 Find a problem whose solution requires a repetition construct. Design an algorithm to solve this problem. Answer this question using either a word-processing program or a drawing program. Save your file as MYLOOP.
- 19 The three basic control structures are not only applicable to computer programs. Give at least two non-computer examples of each of the different control structures. Prepare your answer using a word processor's 'table' functions.
- 20 Gina has included the following post-test in her program, but has found that a pre-test is needed as a negative mark may be entered as the first one. Construct an appropriate pre-test which achieves the same result as this post-test:

```
set mark to user input
REPEAT
    set total to total + mark
    set mark to user input
UNTIL mark < 0
```

Checking algorithms

One reason for designing an algorithm before writing a computer program is to make sure that the steps chosen to solve the problem actually do so. A very important step in the process of software design is that of checking an algorithm after it has been written. The process involved is known as **desk checking** the algorithm, so-called because it is performed without the help of a computer.

Desk checking involves working through the algorithm, keeping track of the values of the variables and the outputs of the algorithm as the steps are manually gone through with sample data items. These data items, known as **test data**, have been chosen so that they check all parts of the algorithm and have known outputs. We will look at how these items are chosen at a later stage in the course.

In performing a desk check, we need to record our progress. This is usually done by drawing up a table which lists the variables by name, the values of the variables as they change and the values of the outputs. One of the most common ways of drawing up the table is to have one line representing a different variable and its values.

As an example, we will desk check the following algorithm which is supposed to add up the counting numbers up to the number input by the user.

Pseudocode

```
BEGIN
    set end_number to user input
    set counter to 0
    set sum to 0
    WHILE counter < 5
        set counter to counter + 1
        set sum to sum + counter
    ENDWHILE
    display counter, sum
END
```

The first step is to read through the algorithm, listing the variables in the left column as we come to them for the first time. In this way no variables will be left out of the table (see Figure 4.39).

Variable	Vaues	Output
end_number		
counter		
sum		

Figure 4.39 The desk check table after listing the variables.

As the value of a variable is changed, it is neatly crossed out with a line so that it can still be read if necessary, and the new value is added. This process of keeping track of the values of the variables is commonly known as **tracing**. As outputs are reached in the algorithm, they are listed in the last column.

The table above will look like this after the algorithm has been tested:

Variable	Vaues	Output
end_number	5	5
counter	0, 1, 2, 3, 4, 5	15
sum	0, 1, 3, 6, 10, 15	

Figure 4.40 The desk check table after the check has been performed.

The final stage of a desk check is to compare the expected result, which has been provided with the test data, against the actual outputs in the table. If they match, the algorithm works; if they don't, the algorithm will need to be modified.

Modification of the algorithm will usually involve only a few steps, although in some cases it may be necessary to completely re-design the algorithm. This decision is made after the desk-checking process has been completed. Once an algorithm has been modified or re-designed, the whole process of desk checking should be undertaken again.

In more complicated algorithms, more than one set of test data may be required to check its workings. In this case a separate table will be required for each of the data groups. The test data itself will probably be displayed in the form of a table which lists the values and expected outputs. This table is known as an **input/output table**.

Example

A program is to be written to grade a typist's speed. If the speed in words per minute (wpm) is below 25, the typist is to be graded as 'Unsatisfactory—needs training'; if the speed is 25 wpm to 55 wpm inclusive, the typist is to be graded as 'Satisfactory'; if the speed is over 55 wpm, the typist is to be graded as 'Exceptional—receives a 10% bonus' (see Table 4.41 on the next page).

Data item Type speed	Expected output	Reason for inclusion
24	Unsatisfactory—needs training	Below the lower boundary value for <i>satisfactory</i>
25	Satisfactory	The lower boundary value for <i>satisfactory</i>
26	Satisfactory	Within the <i>satisfactory</i> range
55	Satisfactory	The upper boundary value for <i>satisfactory</i>
56	Exceptional—receives a 10% bonus	Above the upper boundary value for <i>satisfactory</i>
-1	Invalid input—out of range	To test data validation procedures

Figure 4.41 Input/output table for typing speed grading.

Exercise 4.10

- Copy and complete the following sentences with the most appropriate word or phrase from the list:

desk checking, input/output table, test data, tracing, variables

- A listing of test data items with the expected outputs is known as a(n) _____.
- _____ is the process of using a pen and paper to check an algorithm.
- A process like _____ is used to keep track of variable and their values.
- Data items chosen to check an algorithm are known as _____.
- A _____ must be listed before a desk check can take place.

- Draw up a table which would be used to desk check this algorithm:

Pseudocode

```

BEGIN
    set count to 1
    set total to 0
    set maximum to user input
    WHILE count is less than maximum
        set total to total + count
        print count and total
        increment count by 1
    ENDWHILE
END

```

- 3 The following algorithm is supposed to display all the numbers from 0 up to and including the input number. Draw up a table for the desk check, then use it to check the algorithm, using an input of 5. Does the algorithm work properly?

Pseudocode

```
BEGIN
  set count to 0
  set maximum to user input
  WHILE count is less than maximum
    print count
    increment count by 1
  ENDWHILE
END
```

- 4 Use an appropriate computer application to design a form that you could use for your desk checks. Save this form so that you can print and use it any time you have a desk check to perform.
- 5 This algorithm is designed to display the two input numbers in order, with the larger one first. Test data items for the desk check are shown in the input/output table in Figure 4.42. Perform a desk check on this algorithm and report whether it solves the problem.

Data item first number	Data item second number	Expected output	Reason for inclusion
10	11	Bigger number is 11 Smaller number is 10	Test with smaller number first
11	10	Bigger number is 11 Smaller number is 10	Test with bigger number first
10	11	Bigger number is 10 Smaller number is 10	Test with identical numbers

Figure 4.42 Input/output table for ordering two numbers.

Pseudocode

```
BEGIN
  set firstnumber to user input
  set secondnumber to user input
  IF firstnumber > secondnumber THEN
    set temporary to firstnumber
    set secondnumber to firstnumber
    set firstnumber to secondnumber
  ENDIF
  Print 'Bigger number is ', secondnumber
  Print 'Smaller number is ', firstnumber
END
```

Standard algorithms

There are some common processes which appear in a wide range of problems. Rather than rewriting the algorithm each time it is required, a standard one can be used. This saves both time and effort as the algorithm needs to be changed only slightly for each of the new applications. These changes will be to variable identifiers and the range of counters. Following are some standard algorithms which may be of use.

Swap two data items

This algorithm has an input of two data items called *item1* and *item2*. At the conclusion of the algorithm, the values of *item1* and *item2* will have been swapped. To use this algorithm, replace *item1* with one of the variables you wish to swap and *item2* with the other.

Pseudocode

```
BEGIN swap items
  set temporary to item1
  set item1 to item2
  set item2 to temporary
END swap items
```

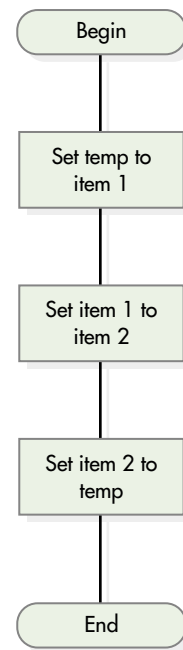


Figure 4.43 Flowchart to swap two data items

Access elements of an array

In many cases we wish to access the elements of an array in order. The most common of these cases would involve reading in the elements of the array, printing each of the elements of the array or processing the records of a file sequentially. The basic structure of these algorithms is the same, that is, to pass from the first array element to the next until the final one is reached. We will use the same basic structure to show this in two examples—working with an array and processing the elements of a file from the first until the last one is reached. Look at the similarities between the algorithms.

Array processing

Array name *arrayitem(count)*, starting index value *first* and ending index value *last* need to be replaced by the appropriate values for your problem (Figure 4.44).

Pseudocode

```
BEGIN process array
  FOR count goes from first to last
    process arrayitem(count)
  NEXT count
END process array
```

As the value of count changes, we are accessing different members of the array `arrayitem(count)`. For example, if count has a value of 6, the line `process arrayitem(count)` is exactly the same as using `process arrayitem(6)`. However, when counter changes to 7, our line `process arrayitem(count)` changes to mean `process arrayitem(7)`.

Processing records from a sequential file

The purpose of this algorithm is to pass through each of the data items in a sequential file until a **sentinel value** is found. If the data is stored as an array with a known number of elements, then the above algorithm may be used to pass through the array of records (see Figure 4.45).

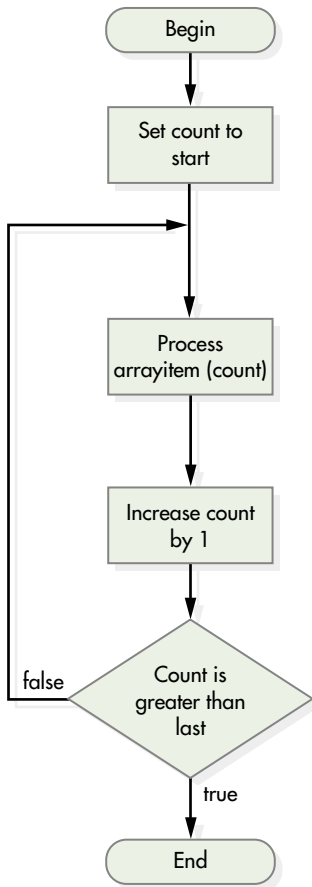


Figure 4.44 Flowchart for array processing.

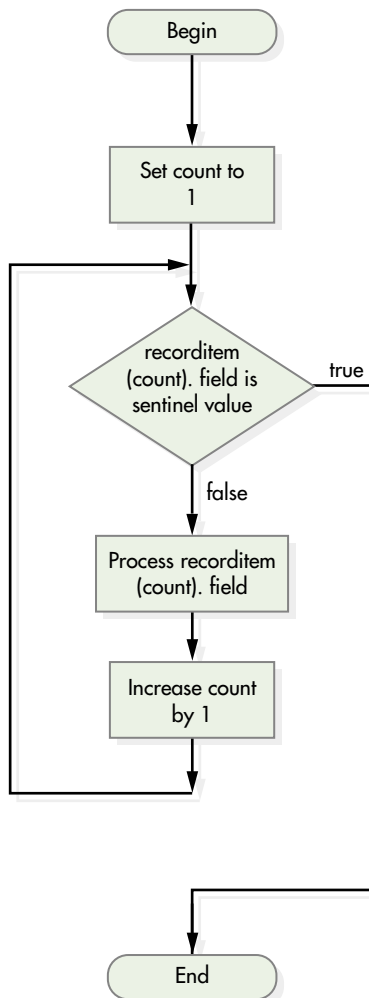


Figure 4.45 Flowchart for processing records from a sequential file.

The variable `recorditem(count).field` is the field that takes part in the processing in this template. It may need to be replaced by more than one field if the processing involves more than one field from each record. The value `sentinelvalue` should also be replaced by the value you are using as a sentinel.

```
BEGIN process sequential file
  set count to 1
  WHILE recorditem(count).field <> sentinelvalue
    process recorditem(count).field
    set count to count + 1
  ENDWHILE
END process sequential file
```

Exercise 4.11

- 1 Use a word processor to copy the three pseudocode algorithms from this section. Save the files as `SWAPVALP`, `ARRAYPRP` and `FILEWORP`. You can then use these files as templates for the following exercises.
- 2 Use a drawing program to copy the three flowcharts from this section. Save the files as `SWAPVALF`, `ARRAYPRF` and `FILEWORF`. You can then use these files as templates for the following exercises.
- 3 Using the swap algorithm in this section as a basis, create an algorithm to read in two numbers, swap their values and print out the resulting numbers.
- 4 Create an algorithm that will input two numbers and swap them only if the first number is bigger than the second. Print out the numbers in order, with the smaller one first.
- 5 Use the array algorithm as a template to help you design an algorithm that will input ten numbers and then print them out after they have been stored. (Hint: You will need to use the array algorithm twice.)
- 6 An array of names and ages has been saved as a sequential file. The sentinel value for the list is an age of `-1`. Modify the file algorithm so that all stored names and ages are printed. The records should be named as `person(count).name` and `person(count).age`.

EXTENSION

- 7 Write an algorithm that will print out the largest value in a list of ten numbers. Check your algorithm with the values 10, 4, 23, 16, 8, 19, 27, 13, 15 and 3.

Review exercises

- 1 Explain, in your own words, the three main steps in solving a problem. Use an example to illustrate your answer.
- 2 Create an IPO chart for a system that performs the following task:
When a sensor on an automatic door senses a person, the door is opened for 30 seconds. If a second person is detected within the 30 seconds, the door remains open for a further 30 seconds; otherwise the door closes. (You do not have to write an algorithm for this problem.)
- 3 An automatic petrol pump has three nozzles which dispense three different types of fuel. A program is required to operate the pump. Decompose this problem into a number of sub-modules.
- 4 Represent the number 212_{10} in:
 - a binary form
 - b octal form
 - c hexadecimal form
- 5 Represent the binary number $1001\ 1101_2$ as:
 - a an octal number
 - b a hexadecimal number
 - c a decimal number
- 6 Name the different simple data types and give an example of the appropriate use of each of them.
- 7 Explain the difference between an array and a record, giving an example of each.
- 8 Describe, using examples from the flowchart in Figure 4.46, the basic structures to be found in this algorithm.

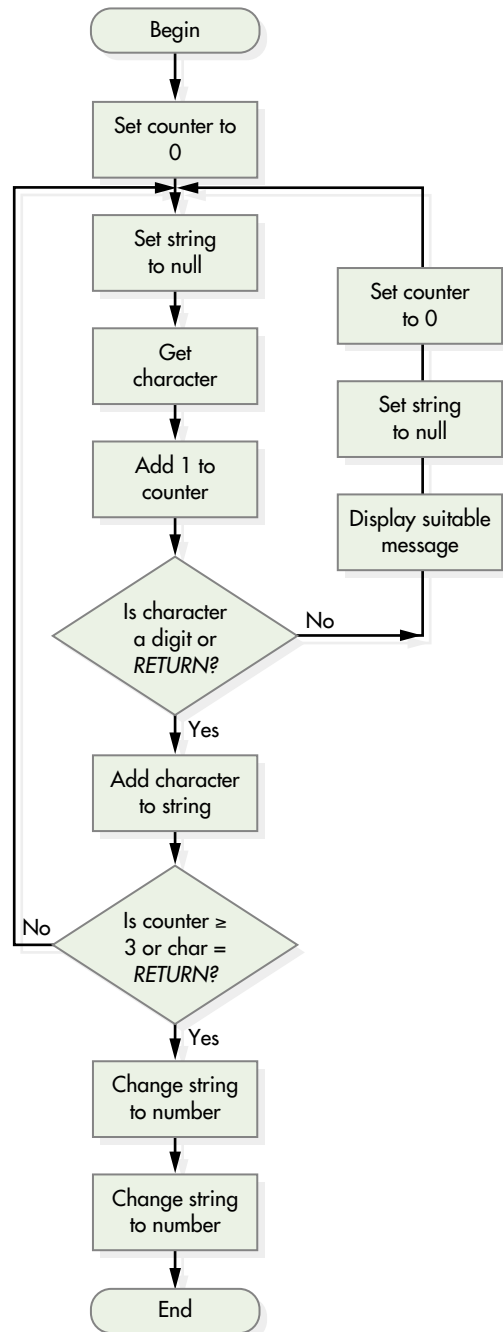


Figure 4.46

Review exercises

- 9 Create an algorithm that will input ten numbers and print out their sum and their average.
- 10 Explain the similarities and differences between the three types of iteration. Show how each of these types of iteration could be used to create an algorithm, in pseudocode, that displays the first ten square numbers.
- 11 An algorithm is required to grade oranges as 'small', 'medium' and 'large'. 'Small' oranges have a diameter of less than 8 cm, 'large' oranges have a diameter over 12 cm, and all others are graded as 'medium'. Design a set of test data to test this algorithm. Display your test data as an input/output table.
- 12 The following algorithm in pseudocode is designed to add up the numbers from 1 to 10 and display the result at the end. Perform a desk check on the algorithm. Find the error in the algorithm and correct it, showing that it works correctly by performing a desk check on your corrected solution.
- ```
BEGIN
 set counter to 0
 set total to 0
 REPEAT
 set total to total +
 counter
 set counter to counter
 + 1
 print total
 UNTIL counter > 10
END
```
- 13 A class markbook program calls for 25 marks from a test to be entered and stored as an array. These marks are then to be averaged and the average printed. Using the array template as a guide, write an algorithm that will perform this task.

## Team Activity

Design a program that will keep rainfall records for the past 30 days. When a new day's rainfall is entered, the previous rainfalls are 'shuffled back' in the array. (For example, day 29's rainfall goes into day 30, day 28's rainfall goes into day 29 and so on, so that the most recent rainfall is then placed into day 1's place.) Create an IPO chart to go with the

solution, as well as an algorithm. The final project should be presented as a computer-generated document, both on disk and on paper. You can use either a word processor or a drawing program to construct the algorithm, depending on whether you wish to use pseudocode or a flowchart.

# Chapter summary

- There are three steps to solving a problem:
  - Understand the problem.
  - Work out a way to solve the problem.
  - Check the solution to the problem.
- An IPO (input, processing, output) chart can be used to help understand the problem.
- An IPO chart is created by looking first at the outputs required, then the inputs needed, and finally the processes required to turn the inputs into the required outputs.
- One way of solving a problem is to break it down into smaller and smaller parts. This method is known as the top-down approach.
- An advantage of the top-down approach is that the logic of the program can be tested at each stage of development.
- Computers use binary digits to represent data and instructions.
- The binary system uses the digits 0 and 1 to represent all numbers.
- We may need to convert numbers from decimal into binary in order to understand what happens inside a computer.
- The binary number is not a convenient way to write numbers, so octal (base eight) or hexadecimal (base sixteen) numbers may be used.
- Data often needs to be represented on paper so that we can understand what needs to be done to process it. These structures are known as abstract data types.
- The simple data types are:
  - Boolean (true/false)
  - character (such as letters and digits)
  - string (a number of characters treated as one item)
  - integer (whole numbers)
  - floating point (decimal numbers)
  - date and currency (special purpose types).
- Structured data types are:
  - array (a number of related data items of the same type stored together)
  - record (a number of related data items of different types stored together)
  - file (a number of records stored together).
- A sequential file is stored in such a way that the records are processed in order.
- An algorithm is a finite number of actions with a single beginning and a single end that will solve a problem.

# Chapter summary

- Algorithms can be described in a number of different ways. The approved methods in this course are flowcharts and pseudocode.
- Pseudocode is an algorithm description method that uses the English language to describe the steps.
- A flowchart is a graphical method of describing an algorithm.
- All algorithms are made from three standard constructs:
  - sequence (steps follow one after the other)
  - selection (a choice is made between different paths)
  - iteration, or repetition (one or more steps are repeated a number of times).
- There are two forms of selection:
  - binary selection, in which there is a choice of two different paths
  - multiway selection, in which there is a choice of three or more different actions.
- There are two different types of iteration:
  - the pre-test loop (the test to exit takes place at the start of the loop)
  - the post-test loop (the test to exit the loop takes place at the end of the loop)
- In a counted loop iteration the loop will be executed a counted number of times.
- An algorithm needs to be tested to ensure that it works properly. With a set of test data that has known outputs, a manual testing method known as desk checking is used.

# chapter 5

## → *Building software solutions*

### Outcomes

- describes and uses appropriate data types (P 1.2)
- describes the interactions between the elements of a computer system (P 1.3)
- identifies the issues relating to the use of software solutions (P 3.1)
- investigates a structured approach in the design and implementation of a software solution (P 4.2)
- uses a variety of development approaches to generate software solutions and distinguishes between these approaches (P 4.3)
- uses and justifies the need for appropriate project management techniques (P 5.1)
- uses and develops documentation to communicate software solutions to others (P 5.2)
- describes the role of personnel involved in software development (P 6.1)
- communicates with appropriate personnel throughout the software development process (P 6.2)
- designs and constructs software solutions with appropriate interfaces (P 6.3)

### Students learn about:

#### Coding in an approved language

- metalanguages
- the syntax of control structures
- the syntax of defining data structures

#### Error correction techniques

- types of coding errors
- stubs

- flags
- debugging output statements

#### Libraries of code

- reusable code
- combining code and modules from different sources

#### User interface development

- consultation with users
- user and developer perspectives
- effective user interface

#### Documentation

- types of documentation
- internal documentation
- online help

## Students learn to:

- read and construct productions using the metalanguages BNF, EBNF and syntax structure diagrams
- identify the syntax of control structures and correctly code a control structure in a chosen language
- define data types in a chosen language
- identify types of coding error including errors in syntax, run-time errors and logical errors
- use stubs, flags and debugging output statements to help identify errors.
- create standard modules for reuse
- use code from different sources to create a solution to a problem with minimal change
- develop an effective user interface after consultation with users
- create appropriate documentation for a software solution
- interpret code and documentation from other developers

## Personal Profile—John Backus (1924– )

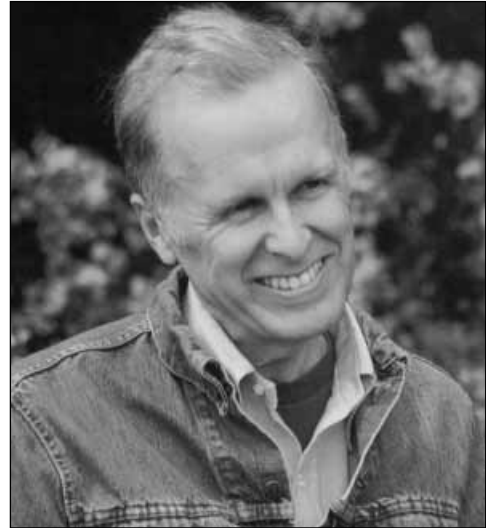
Born in 1924 in Philadelphia, John Backus spent his early years in Wilmington, Delaware. His family was quite wealthy as his father was a chemical engineer, so he was able to attend the prestigious Hill School in Pottstown, Pennsylvania. At this stage he exhibited little interest in academic pursuits, earning poor grades but still managing to graduate.

John studied chemistry briefly at the University of Virginia, where his lack of interest in the practical aspects of the course saw his attendance rates fall and he was expelled. In 1942 he joined the army. As a corporal in charge of an anti-aircraft battery, he was given an aptitude test which led to him studying engineering. A further aptitude test while he was in this course prompted another career change, this time to medicine. During the pre-medicine course at an Atlantic City hospital he was diagnosed with a brain tumour. A subsequent operation left him with a plate in his head. John's medical career lasted only a further nine months at which stage he felt that medicine too was not to be his chosen career. He left the army in 1946.

While enrolled in a radio technician's course, John found that he had an aptitude for mathematics, which led to his enrolment in a science degree majoring in mathematics. In 1949, when almost at the end of his degree course, John visited the IBM Computer Centre on Madison Avenue and mentioned that he was looking for a job. He was hired to work on the IBM Selective Sequence Electronic Calculator (SSEC), a primitive mechanical computer which had no software memory, all programs being entered using paper tape. During his three years working on SSEC, John devised a program called Speedcoding which was the first to include floating point numbers.

At the end of 1953 John proposed a programming language for the new 704 model computer. The proposal was accepted by IBM as the language would be quicker and simpler for programmers to use. The language was FORTRAN (FORMula TRANslation), which is still used by many mathematicians and scientists today. However, the team assembled by Backus was overtly optimistic about the time it would take to design both the language and its translator, the whole of the design taking two years rather than the anticipated six months.

John Backus's next important contribution to computer science was the development of the notation BNF, used to describe the rules of a language, in 1959. The first use of BNF was in the design, concluding in 1960, of the language ALGOL. Following his involvement in the ALGOL project he continued to look for better methods of programming, and his third contribution was the functional-level language FP. During his career John Backus was given many awards for his contributions to computer programming. He retired from the computer industry in 1991.



# Coding in an approved programming language

## Metalanguages

One of the most important features of any programming language is that it must not contain any ambiguous statements. In other words, any statement written using the language must have only one, precise meaning.

First- and second-generation languages, since they are processor-specific, have individual instructions which are directly translated into an action by the processor. The form of these instructions is created when the processor is designed. When programming with first- and second-generation languages, the algorithm is broken into these single instruction steps. Since the program is written as a series of immediately recognisable simple instructions, there can be no thought of ambiguity.

Languages of later generations, or 'high-level languages', are more complex, since they more closely approach natural languages. A single statement in one of these languages will be translated into a number of processor instructions. The need to translate from one language to another made it important to have every statement in the high-level language unambiguous. If this were not the case, a program might produce one result when run one day and a different result the next. It is obviously unacceptable to create computer programs that produce unpredictable results.

In the mid to late 1950s, an American, John Backus, devised a notation which could be used to describe the manner in which a programming language's rules could be described. The notation is known as a **metalanguage**, that is, a language used to describe the rules of another language. In 1959 this notation, known as **Backus Normal Form (BNF)**, was used to describe the syntax of the language ALGOL 58. In 1960 Peter Naur modified the metalanguage slightly for the description of the language ALGOL 60. This modified form, now known as **Backus-Naur Form (BNF)**, has become one of the more common methods of describing programming languages. Later a further change was made to BNF which gave the metalanguage the ability to describe the repetition of a language element. This extended form is known as **Extended Backus-Naur Form (EBNF)**.

Because it is a text-based description of a language, the structure of each of the language elements may not be as clear as it would be with a graphical description. In order to overcome this problem, it is possible to show graphically the structures described by BNF or EBNF. These diagrams are commonly known as **syntax structure diagrams** or **syntax graphs**.

## Representation of syntax

The **syntax** of a language refers to the rules which govern the way that the elements of that language can be joined to form valid statements. The syntax of a language will determine how simple elements can be joined to create **statements**, as well as the ways in which these statements can be combined to form **compound statements**.

A computer language needs to be precise in its definition, since every statement has to be unambiguous. Ambiguity refers to the situation where a particular statement can be given two or more different meanings. For example, the statement 'Jane beat Graeme' might mean that Jane scored better than Graeme in some form of competition, or it might mean that she hit Graeme.



## Syntax structure diagrams (railroad diagrams)

Syntax structure diagrams show the allowable structure of a language element by tracing a path from left to right; the allowable features correspond to the 'stations' along the line. Branches are shown as railway 'points', and the path can only move onto a branch by travelling onto the points as if it were a train (see Figure 5.1).

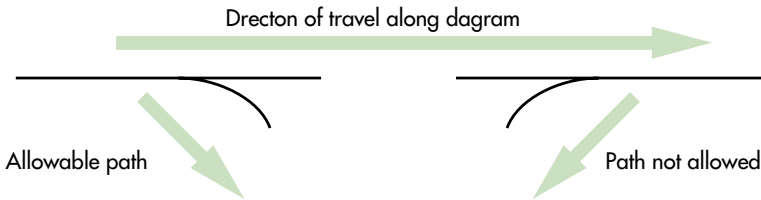


Figure 5.1 Branchings of a syntax structure diagram.

The syntax structure diagram employs three symbol types to represent two allowable elements: the rectangle to represent a previously defined element, and a rounded rectangle (bubble) or circle to represent a fixed element (such as an individual symbol or reserved word) (see Figure 5.2).

### Interpretation of syntax structure diagrams

As described above, syntax structure diagrams are read as if following the path of a railway train from left to right. When a branching point (like the set of railway points) is reached, there is the option of following one of these branches.

Once on a fork, that line must be followed until it reaches the main line again. There will be only one beginning to a diagram and only one ending.

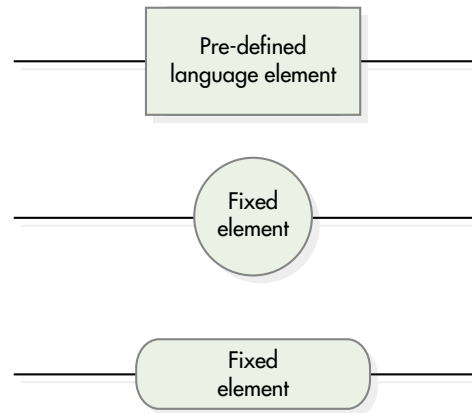


Figure 5.2 Basic symbols of a syntax structure diagram.

### Example 1

Following the diagram shown in Figure 5.3 from the left, a Boolean constant is defined as being either True (following the main line from left to right) or False (following the branch line). Note that the flow does not allow us to retrace the line back through 'True' once the branch joins the main line.

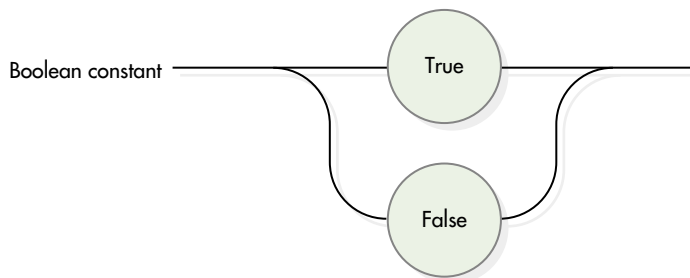


Figure 5.3 Alternative elements.

## Example 2

The diagram shown in Figure 5.4 presumes that the element 'letter' has been previously defined. The definition of a 'word', using the diagram, is that it must consist of at least one letter (since the main line runs through the letter element) but may consist of more, as the branch may be followed around to another letter.

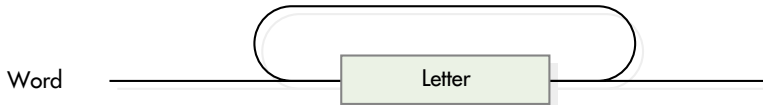


Figure 5.4 Repeated elements.

## Example 3

In the structure diagram shown in Figure 5.5, the element 'digit' has already been defined. This diagram defines an integer as having an optional 'minus' sign (since there is a path past the minus sign) followed by at least one digit. One or more optional elements can be included in a diagram by leaving a free path in addition to the branch or branches.

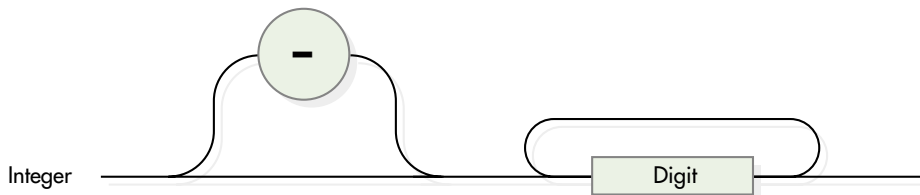


Figure 5.5 Optional elements.

### Construction of syntax structure diagrams

The construction of a syntax structure diagram to represent a particular language element is not very difficult. The two main structures illustrated in Figure 5.2 (previously defined elements and fixed elements) can be joined to create a diagram which represents a legal language element.

For example, a name might be defined as 'an uppercase letter followed by one or more lowercase letters'. (Assume that the elements 'uppercase letter' and 'lowercase letter' have been previously defined.) The structure diagram must therefore contain an uppercase letter as the first 'station', followed by a lowercase letter, giving the diagram in Figure 5.6.



Figure 5.6 Stage 1 of the construction.

In order to repeat the lowercase letter, a branch needs to be inserted after the lowercase letter leading back behind that letter, so the completed diagram is as shown in Figure 5.7.

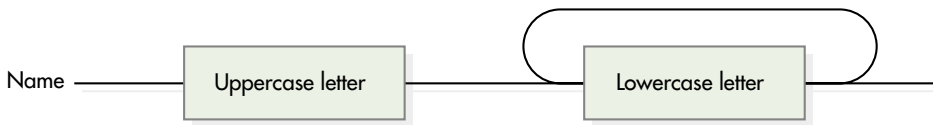


Figure 5.7 The completed syntax structure diagram.

### Checking the syntax of a written language element

The main purpose of syntax description is to enable programmers to check coded statements and ensure that they have been coded within the rules set down for the particular language. In the examples shown in Table 5.1, the syntax structures in Figure 5.8 will apply.

Some of the code samples in Table 5.1 are legal, and some of them are illegal. The right-hand column of the table indicates whether the sample is legal in terms of the syntax rules; if it is not, an explanation is given as to why it is illegal.

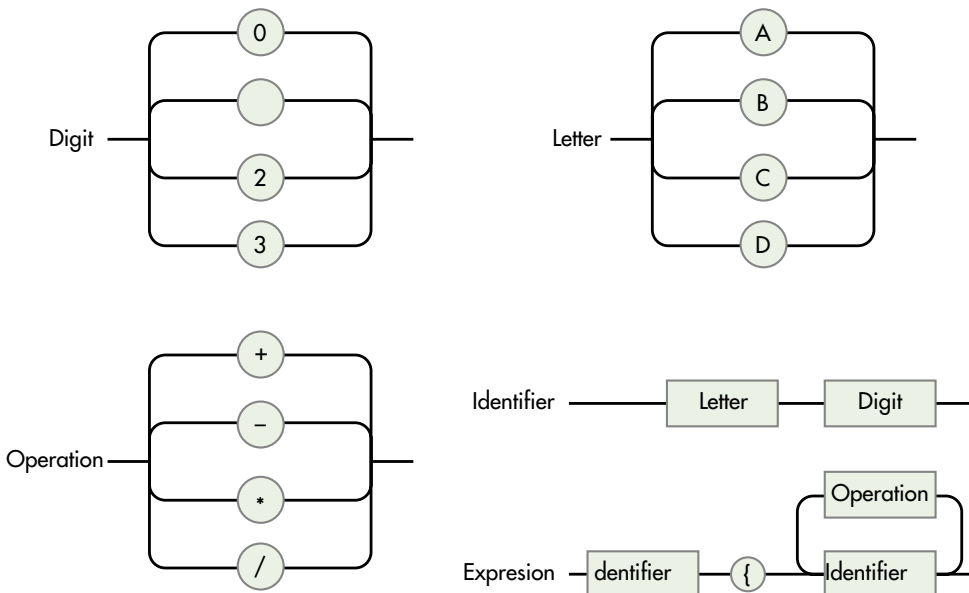


Figure 5.8 Syntax structure diagrams showing the rules used in Table 5.1.

| Code sample                     | Legality                                                                         |
|---------------------------------|----------------------------------------------------------------------------------|
| 1. B3                           | Legal identifier—it contains a defined letter followed by a defined digit        |
| 2BAllea idnifier—th             | first letter has tbe followed by a defined iit                                   |
| 3C3{A4/B1*Clleal exprssion      |                                                                                  |
| 4b3{A1*Clleal—th idnifier b3    | stats wianillegal charcter (only uppercase letters have been defined s a letter) |
| 5A2*C4{Blleal—the syntaxon      | allows onedentifier before the brace ('{' character)                             |
| 6B4{2*Clleal—nl idnifiers a     | allowed to be operated upon                                                      |
| 7A1{(B2+C4)/Alleal—the brackets | are t allowable charates wthin this syntax                                       |
| 8. A5{C4—B3                     | Illegal—the digit in the first identifier is not defined                         |

**Table 5.1** Code samples written using the syntax structures.

## Backus-Naur Form and Extended Backus-Naur Form

### Backus-Naur Form (BNF)

The Backus-Naur Form (BNF) of describing syntax structure has found favour with many language developers as it can be employed by a computer system to assist with language design and implementation. BNF was designed by the American computer scientist John Backus (the ‘father’ of FORTRAN) to assist in the formal description of the ALGOL language which, at the time, was proposed as a universal language. BNF was originally called the Backus Normal Form, but it was refined by the Danish astronomer Peter Naur, and so his name was incorporated into the description name (the acronym BNF conveniently staying the same).

BNF is a text-based system and as such cannot draw upon shapes to define or refer to elements. Several symbols are used in BNF to indicate the structures seen above (see Example 4).

Language elements are referred to either by their names (if they have been pre-defined) or as individual characters. Those elements which are defined elsewhere (known technically as **non-terminal symbols**) are enclosed by ‘less than’ and ‘greater than’ symbols—for example <language element>. Individual characters or strings of characters (known as **terminal symbols**) are not enclosed. For example, in BNF ...print... means that the word ‘print’ in a definition is a string of characters. (Note that each of the characters in the word is a terminal symbol.) If it appears in a definition as ...<print>..., the word ‘print’ represents something defined elsewhere. (The word ‘print’ is, therefore, a non-terminal symbol.)

Alternative elements are indicated by placing a vertical bar in each space between alternatives: |. The symbol set ::= is used to represent the statement ‘is defined as’.

Note that, for clarity, in this text extra spaces have been placed between language elements in both BNF and EBNF; this may not always be the case.

### Extended Backus-Naur Form (EBNF)

The main shortcomings of BNF are its inability to simply show optional parts of a definition (such as the negative sign in the integer example above), possible

## Example 4

- 1 The Boolean constant defined in Figure 5.3 would appear thus in BNF:

```
Boolean_constant ::= true | false
```

- 2 The word defined in Figure 5.4 would appear thus in BNF:

```
word ::= < letter > < word > | < letter >
```

In this example a word is defined differently from the structure diagram, since BNF has no way to show repetition. Repetition is indicated by using a method known as **recursion**, in which the definition of the element is used to define itself. A word here is defined as being a letter followed by either a word or a letter. This definition eventually breaks down a word into two consecutive letters, which is a legal construction. A single letter, under this definition, is not a word. For example, TUSIL is a legal word since IL is a letter followed by another. Hence SIL is a legal word as it is the letter S followed by the word IL. Therefore USIL is a word, and finally TUSIL is a word under this definition.

- 3 To define integer in the same way as it is defined in Figure 5.5 using BNF we need to define two elements, an unsigned integer and then the integer:

```
unsigned_integer ::= < digit > | < digit > < unsigned_integer >
```

```
integer ::= < unsigned_integer > | - < unsigned_integer >
```

repetition (such as possible extra letters in the definition of ‘word’ above) and the grouping of elements. The extended form of BNF introduces these capabilities to the structure (see Example 5).

Alternative elements are again indicated by placing a vertical bar in each space between alternatives: |. The symbol ‘=’ is used to represent the statement ‘is defined as’. Terminal and non-terminal elements are represented in the same manner as in BNF.

Repetitions are indicated by enclosing the repeated element(s) in braces: { }. Repetition is taken as being anything from 0 times upwards, meaning that the *repetition is also optional*.

Optional elements are enclosed in square brackets: [ ].

Grouped elements are enclosed in parentheses: ( ).

## Example 5

- 1 The Boolean constant defined in Figure 5.3 appears in this form in EBNF:

```
Boolean_constant = true | false
```

- 2 The word defined in Figure 5.6 would appear thus in EBNF:

```
word = < letter > { < letter > }
```

This definition is easier to follow as it defines a word as at least one letter followed by an optional repetition of other letters.

- 3 The integer defined in Figure 5.5 is again much simpler to define in EBNF than in BNF as it can be accomplished in one statement:

```
integer = [-] < digit > { < digit > }
```

In this definition an integer is defined as an optional minus sign followed by at least one digit.

## Checking the syntax of a written language element

Like the railroad diagram, a BNF or EBNF statement is read from left to right. The structure of the element being examined can be checked against the defined syntax.

In the examples shown in Table 5.2, the following EBNF syntax structures will apply:

```

letter = w | x | y | z
digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
operation = + | - | x | ÷ | √
identifier = <letter> { <letter> | <digit> }
number = [-] <digit> • <digit> { <digit> } [e <digit> <digit>]
expression = <identifier> ← <identifier> | <number> { <operation>
<identifier> | <number> }

```

Some of the code samples that are shown in Table 5.2 are legal, and some are not. The right-hand column of the table indicates whether the sample is legal in terms of the syntax rules; if it is not, an explanation is given as to why it is illegal.

| Code sample      | Legality                                                                                                                 |
|------------------|--------------------------------------------------------------------------------------------------------------------------|
| 1. z3            | Legal identifier—it contains a defined letter followed by a digit                                                        |
| 2. xw34yz        | Legal identifier—the first character is a letter, and each of the other is either a letter or a digit                    |
| 3. we3           | Legal identifier—it contains an undefined character (the letter 'e')                                                     |
| 4. w70←3.14xx    | Legal                                                                                                                    |
| 5. x2xy4←z2      | Legal—the syntax allows only one identifier before the arrow                                                             |
| 6. w41←2√z143    | Legal                                                                                                                    |
| 7. y1←(x2+y4)/w2 | Legal—the brackets and the slash (/) are not allowable characters in this syntax                                         |
| 8. Z5←x4-yq3     | Illegal—the letter in the first identifier is not defined (it is uppercase and only lowercase letters have been defined) |

**Table 5.2** Code samples written using the syntax structures.

## Syntax description examples

In the Examples 6 and 7, we will examine the manner in which syntax descriptions can be used to assist in the coding of a language. These examples illustrate the simple assignment of summing two values stored in main memory, and placing the result (weekly wage) in a third storage area, and calculating the area of a trapezium. They illustrate different ways in which languages implement this simple structure. Each of the examples gives the structure of the statement in the syntax description methods of BNF and structure (railroad) diagrams.

## Example 6 FORTRAN IV

A BNF description of assignment statements and associated elements in the FORTRAN language is presented below:

Arithmetic operation ::= <Number> | <Identifier> | <Bracketed Operation> <Arithmetic Operator> <Number> | <Identifier> | <Arithmetic Operation>

Arithmetic operator ::= + | - | \* | / | ^

Assignment statement ::= <Identifier> = <Identifier> | <Arithmetic Operation> | <Bracketed Operation>

Bracketed operation ::= ( <Arithmetic Operation> )

Identifier ::= <Uppercase Letter> <Letter or Number String>

Letter or number string ::= <Uppercase Letter> | <Digit> | <Letter or Number String>

Number ::= <Real Number> | <Integer>

Note: In order to save on space and assist with the understanding of the structure, definitions of symbols such as Uppercase Letter and the Number types have been omitted, as have the distinctions between Integer and Real Identifiers. For simplicity, inbuilt functions such as the trigonometric functions have also been omitted. Thus in FORTRAN, as described above, the following translations may be made.

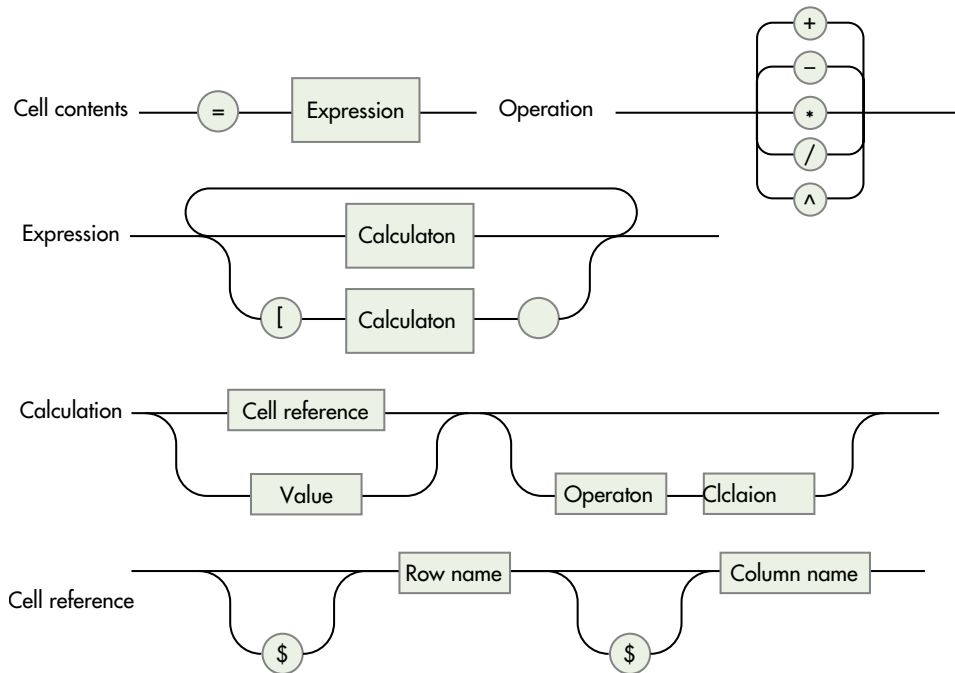
- a The pseudocode instruction  
set weekly\_wage to normal\_pay + overtime  
becomes, in FORTRAN:  
WEEKWAGE = NORMALPAY + OVERTIME
- b The pseudocode instruction  
set area to (top\_side + bottom\_side) \* height / 2  
becomes, in FORTRAN:  
AREA = (TOP + BOTTOM) \* HEIGHT / 2

## Example 7 A typical spreadsheet application

A description of the syntax of a simplified spreadsheet application is presented in Figure 5.9 on the next page as a structure diagram.

In this simplified spreadsheet language, the assignment of a value to a cell would involve the entry of the appropriate language elements into the cell. Thus:

- a The pseudocode instruction  
set weekly\_wage to normal\_pay + overtime  
becomes, using cell B5 for the contents of weekly\_wage, and cells B2 and B3 for the normal\_pay and overtime values respectively, and typing the following formula into the cell to achieve the required result:  
= B2 + B3
- b The pseudocode instruction  
set area to (top\_side + bottom\_side) \* height / 2  
becomes, using a similar structure to the above and placing the formula into an empty cell:  
= (J12 + K12) \* B3 / 2



**Figure 5.9** Representation of a simple spreadsheet syntax using structure charts. Values have not been defined in this structure, but are numerical.

## Exercise 5.1

- Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Computer languages must be \_\_\_\_\_ so that each statement has only one \_\_\_\_\_. In the design of languages a \_\_\_\_\_ is used to show the rules or \_\_\_\_\_ of the language. The graphical form of such is called \_\_\_\_\_ a \_\_\_\_\_ diagram or \_\_\_\_\_ diagram. We can also show the rules by using one of two text-based forms known as \_\_\_\_\_ and \_\_\_\_\_.

- The syntax structure diagrams in Figure 5.10 define the syntax for a simple theoretical language we will call TUSIL (To Understand Syntax Instructional Language). Use the structure diagrams to answer the questions which follow. Note that the language element RB is used in the assignment statement to represent the term 'is replaced by'.
  - Identify whether each of these TUSIL elements is legal or illegal, and if it is illegal, state your reasons. For example:
    - Identifier MN3\_12 is a legal identifier.
    - Identifier 32\_VB is not a legal identifier as the first character is not a letter.
    - Assignment TOTAL RB SUM1 + SUM2 + SUM3 is a legal assignment.
      - identifier EVERY\_1
      - identifier ALL\_OF\_THEM
      - identifier Weeks\_Wages
      - integer -32176
      - real number -3214.676
      - real number 23



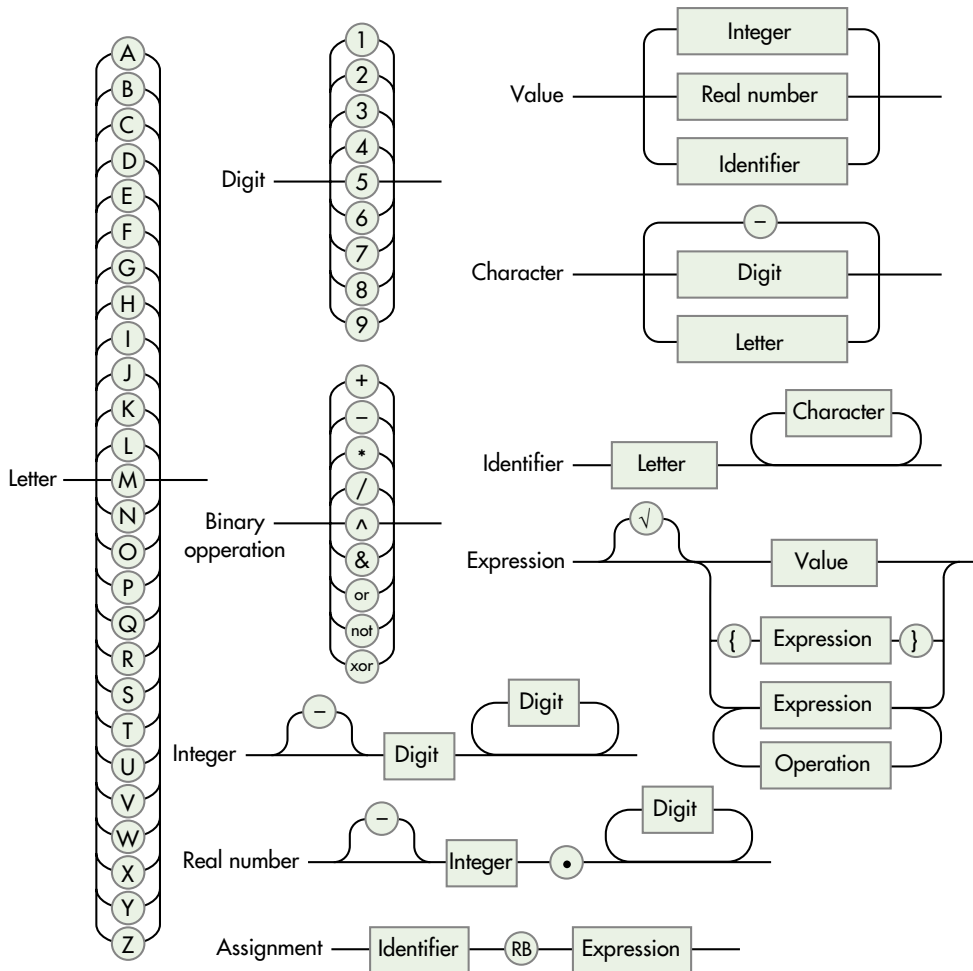


Figure 5.10 TUSIL structure diagrams.

- vii assignment WEEKS\_WAGES RB HOURLY\_RATE \* HOURS\_WORKED \* 1.12
  - viii assignment IS\_IT RB IS\_IT\_FOUND or IS\_LIST\_FINISHED
- b** Using the structure diagrams as a guide, write a legal TUSIL expression for each of the following. Create language elements which are legal in the TUSIL language.
- i an identifier to represent a student's test mark
  - ii a real number representing  $\pi$  (use your calculator to find the value)
  - iii an expression for calculating the number of days in  $W$  weeks
  - iv an expression which calculates the hypotenuse of a right-angled triangle given the other two sides
  - v An expression which finds the value from using the logical operator AND on an identifier and the integer value 8
- c** Draw a structure diagram for each of these TUSIL elements as described below.
- i A relational operator is one of the following operations:  
greater than, less than, equal to, greater than or equal to, and less than or equal to.
  - ii An IF statement consists of the word IF followed by an identifier, a relational operator (you may only use the term 'relational operator' to refer to

the definition above), and a value, followed by the word THEN, after which comes an expression. The word ELSE is optional, but if used it must be followed by an expression. The IF statement must be terminated (ended) by the word ENDIF.

- iii A variable declaration begins with the word 'declaration !'. Following that, each variable declaration consists of an identifier followed by the words 'is an integer' or 'is a real number'. The declaration of each variable is separated by an exclamation mark symbol (!) and the declaration is ended by an exclamation mark and the statement 'end declaration'. At least one variable must be declared. For example, a legal variable declaration would be written thus:

- to declare a single variable such as COUNTER  
 declaration ! COUNTER is an integer ! end declaration
- to declare more than one variable  
 declaration !  
     PAY\_RATE is a real number !  
     NUMBER\_OF\_DAYS is an integer !  
     HOURS\_WORKED is a real number !  
 end declaration

- d Using the diagrams in Figure 5.10 and those created in part c above, check the following piece of TUSIL code, reporting and correcting any syntax errors:

```

declaration OLD_MARK is a real number !
 NEW_MARK is a real number !
 TOTAL_PROCESSED is an integer !
end declaration
IF OLD_MARK < 50 THEN
 NEW_MARK = OLD_MARK * 2
ELSE
 NEW_MARK = OLD_MARK * 3.
ENDIF
TOTAL_PROCESSED RP TOTAL_PROCESSED + 1

```

- 3 Using the following EBNF syntax descriptions, identify and correct the fragments of code which follow:

```

upper_case_letter = A | B | C | D | E | F | G | H | I | J | K |
L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
lower_case_letter = a | b | c | d | e | f | g | h | i | j | k |
l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
letter = <upper_case_letter> | <lower_case_letter>
digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
letter_or_digit = <letter> | <digit>
space = ASCII character 32
special_characters = " | # | % | & | ' | (|) | * | + | , |
- | . | / | : | ; | < | = | > | _ | |
character = <letter_or_digit> | <special_character> | <space>
integer = <digit> { <digit> }
signed integer = [-] <integer>
decimal_number = <signed integer> [. <integer>]
exponent = E [+] <integer> | E [-] <integer>
scientific_number = <decimal_number> <exponent>
character_string = " { <character> } "
identifier = <letter> { [_] <letter_or_digit> }

```

- a Determine the type of each of the following legal elements in the language described by the above EBNF.
- i E
  - ii E3
  - iii E\_3
  - iv 3E+43
  - v +45.81E-12
  - vi 4123645
  - vii "The answer to the question of the meaning of life is 42"
  - viii E\_4123645
- b Each of the following identifiers is illegal according to the syntax described above. Find the error and name and correct it.
- i Item 1
  - ii 1\_item
  - iii Item\_number\_1\_
  - iv Item#1
  - v Item\_bought\_on\_12.3.97
- 4 The letters of the alphabet are divided into vowels (A, E, I, O and U) and consonants (all other letters). A two-letter word must consist of either a vowel followed by a consonant or a consonant followed by a vowel. Write these syntax rules both in BNF and as a syntax structure diagram.
- 5 Convert the TUSIL syntax structure diagrams in Figure 5.10 above into EBNF.
- 6 Convert the EBNF statements in question 3 above into syntax structure diagrams.
- 7 Describe the syntax of a simple arithmetic number sentence (for example  $3 + 4 - 5 = 2$ ) using BNF, EBNF or a syntax structure diagram.

## The syntax of control structures

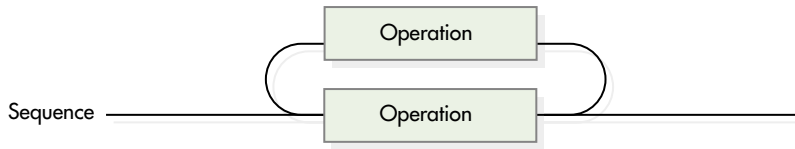
Although different programming languages have differing syntaxes, the basic control structures are very similar.

### Sequence

The coding of a sequence is basically the same in all languages. Each instruction in the sequence will be in the position that corresponds to its place in the algorithm. Some languages, such as FORTRAN IV, require that each instruction is placed on a separate line. Others may allow multiple instructions on a line, each individual instruction being separated from the others by some kind of separator symbol (for example, Pascal uses the semicolon). Other languages may require the use of brackets to separate instructions. No matter how the language has been designed, the basic structure of a sequence will be apparent, with one operation following another (see Figure 5.11).

*Syntax of a sequence in EBNF*

```
sequence = <operation> { <operation> }
```



**Figure 5.11** Structure diagram showing the syntax of a sequence.

In each of the following code examples, the area of a circle of radius 7 is calculated and printed. In the FORTRAN sample, the line numbered 200 is used to declare the format of the printed numbers.

## Example 8

### FORTRAN code

```
RADIUS = 7
AREA = 3.1415927 * RADIUS *RADIUS
WRITE(1s, 200) RADIUS, AREA
200FORMAT(1X, F5.1 , F10.5)
```

### Pascal code

```
radius := 7;
area := 3.1415927 * radius * radius;
writeln(radius, area);
```

### Visual Basic code

```
radius! = 7
area! = 3.1415927 * radius * radius
message = Format(radius, "Fixed", area , "Fixed")
MsgBox(message)
```

### Java code

```
{radius = 7;
area = 3.1415927 * radius * radius;
system.out.println(radius + "" + area);}
```

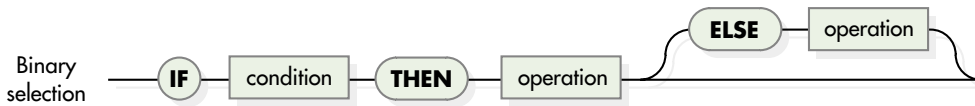
## Selection

When implemented in a language selections will have the same basic structure. The binary selection usually starts with the keyword IF and a multiway selection uses the keyword CASEWHERE, just as they do in pseudocode.

The second part of a **binary selection** will be some condition which will have a result which is either true or a false (the condition is often said to have a **Boolean result**). The keyword THEN is required, followed by an action. If the result of the condition is true, the action following the keyword THEN will be performed. The alternative action (what happens if the condition is false) will follow the keyword ELSE. In cases where there is to be no alternative action, a majority of languages will allow the ELSE part of the statement to be omitted.

*Syntax of a binary selection in EBNF*

binary selection = IF <condition> THEN <operation> [ ELSE <operation> ]



**Figure 5.12** Structure diagram showing the syntax of a binary selection.

The binary selection in each of the following code examples gives a tax rate of 0.2 to those whose pay is \$500 or less and a rate of 0.35 for all other values of pay. In the spreadsheet example, the value of pay is held in cell A5, the tax rate being held in the cell containing the formula.

### Example 9

```
if pay > 500 then
 tax_rate := 0.35
else
 tax_rate := 0.2;
```

#### Spreadsheet code

```
=IF(A5 > 500, 0.35, 0.2)
```

#### Visual Basic code

```
If pay > 500 Then
 tax_rate = 0.35
Else
 tax_rate = 0.2
End If
```

#### Java code

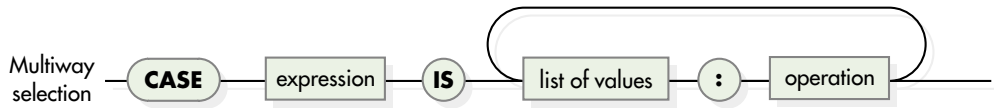
```
if (pay > 500) {
 tax_rate = 0.35;
}
else {
 tax_rate = 0.2;
}
```

The **multiway selection** is implemented in a large number of recent languages. It is a more efficient way of performing a choice than a number of nested IF statements. However, the disadvantage is that only a list of values can be used. This makes it difficult to use for cases where there are a large number of values in a group or where the value being tested has a range rather than specific values.

The multiway selection will be identified by the use of the keyword CASE, followed by an expression that can be evaluated. The appropriate course of action is followed by finding the value of the expression in the list given as part of the CASE statement.

*Syntax of a multiway selection in EBNF*

```
multiway selection = CASE <expression> IS <list of values> :
 <operation> { <list of values> : <operation> }
```



**Figure 5.13** Structure diagram showing the syntax of a multiway selection.

The code examples following illustrate the use of a multiway selection by assigning a grade A to marks of 5 and 4, a grade B to marks of 3 and 2 and a grade C to marks of 1 and 0.

## Example 10

### Pascal code

```

case mark of
 5 , 4 : grade := 'A';
 3 , 2 : grade := 'B';
 1 , 0 : grade := 'C';
end

```

### Visual Basic code

```

Select Case mark
 Case 5 , 4
 grade = "A"
 Case 3 , 2
 grade = "B"
 Case 1 , 0
 grade = "C"
End Select

```

### Java code

```

switch(mark) {
 case 5 :
 case 4 : grade = 'A';
 break;
 case 3 :
 case 2 : grade = 'B';
 break;
 case 1 :
 case 0 : grade = 'C';
 break;
}

```

## Iteration

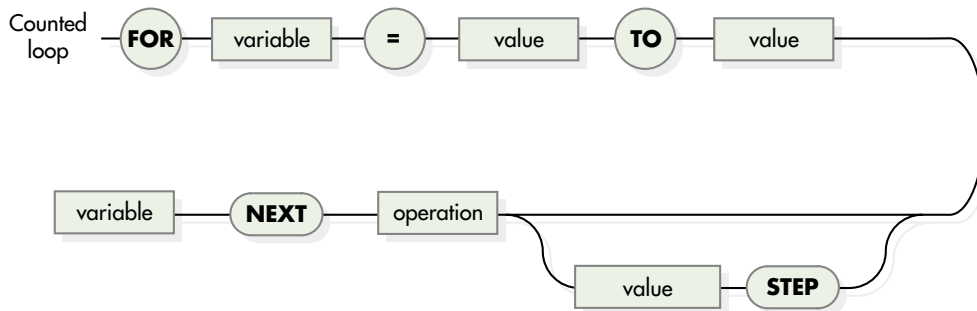
The three types of repetition appear in most languages in a form that is not too different from its representation in the approved form of pseudocode.

When implemented in a language, the **counted loop** has probably the most consistent form of the three iterations. Almost all languages use the keyword FOR, with most using the keyword NEXT to terminate the loop. The counter used within the loop is commonly **incremented** (increased by 1) as a default;

some languages allow the programmer to increase, or decrease, the counter by using the keyword STEP.

*Syntax of a counted loop in EBNF*

counted loop = FOR < variable > = < value > TO < value > [ STEP < value > ] < operation > NEXT < variable >



**Figure 5.14** Structure diagram showing the syntax of a counted loop.

The following code examples illustrate the use of a counted loop by finding the sum of the numbers from 1 to 10.

## Example 11

### Pascal code

```
sum :=0;
for counter := 1 to 10 do begin
 sum := sum + counter
end;
```

### Visual Basic code

```
sum = 0
For counter = 1 to 10
 sum = sum + counter
Next counter
```

### Java code

```
sum = 0;
for (counter = 1; counter <= 10; ++counter)
{
 sum = sum + counter;
}
```

The **pre-test loop** can be identified by the use of the keyword WHILE at the beginning of the loop. Various languages implement this structure in different ways, but if you look for the keyword at the beginning you should have no problem. The end of a pre-test loop is not so easy to pick. Look at the following examples to see how the pre-test loop is implemented. Remember that the operation(s) inside the pre-test loop are executed while the condition is true, the loop finishing once the condition is not met.

*Syntax of a pre-test loop in EBNF*

pre test loop = WHILE <condition > <operation > ENDWHILE



**Figure 5.15** Structure diagram showing the syntax of a pre-test loop.

The following examples illustrate how a pre-test loop can accomplish the same task as given for the counted loop.

## Example 12

### Pascal code

```
sum := 0;
counter :=1;
while counter <= 10 do begin
 sum:= sum + counter;
 counter := counter + 1
end;
```

### Visual Basic code

```
sum = 0
counter = 1
Do While counter <= 10
 sum = sum + counter
 counter = counter + 1
Loop
```

### Java code

```
sum = 0;
counter = 1;
while(counter <= 10)
{
 sum = sum + counter;
 ++counter;
}
```

The **post-test loop** is usually implemented in a similar way to its pseudocode representation, that is, using the keyword REPEAT at the beginning of the loop and the keyword UNTIL at the end together with the condition needed to end the loop. Remember that the post-test loop continues while the condition is false, looping ends once the condition becomes true.

*Syntax of a post-test loop in EBNF*

post-test loop = REPEAT < operation > UNTIL < condition >



**Figure 5.16** Structure diagram showing the syntax of a post-test loop.



The following example of adding the numbers from 1 to 10 illustrate the similarities and differences between languages.

### Example 13

#### Pascal code

```
sum := 0;
counter :=1;
repeat
 sum:= sum + counter;
 counter := counter + 1
until counter > 10;
```

#### Visual Basic code

```
sum = 0
counter = 1
Do
 sum = sum + counter
 counter = counter + 1
Loop Until counter > 10
```

#### Java code

```
sum = 0;
counter = 1;
do
{
 sum = sum + counter;
 ++counter;
} while(counter <= 10)
```

### Syntax of data type definitions

Languages are structured in such a way as to ensure that the storage required for variables is made available when needed. This is done in a variety of ways.

One way is to require that variables are defined (often known as **variable declaration**). For example, the following declaration of variables representing simple data types in Pascal illustrates the methods employed in this type of language:

```
var
 counter, item_count : integer;
 item_description : packed array [1..20] of char;
 markup , cost_price , sell_price : real;
 order_now : boolean;
 row_id : char;
```

A second method is to use a **symbol** after the variable name to identify its simple data type. The variable type symbol is placed at the end of the variable identifier the first time it is used in the program. This example shows a portion of code in Visual Basic which uses this idea:

```
markup! = 0.25
cost_price@ = 8.8
sell_price@ = cost_price * markup
order_now = false
```

Visual Basic also allows for variables to be declared in a similar way to Pascal:

```
Dim counter As integer, item_count As Integer
Dim item_description As String
Dim markup As Single
Dim cost_price As Currency , Dim sell_price As Currency
Dim order_now As Boolean
Dim row_id As Boolean
```

A method used by older languages, such as FORTRAN, was to allocate a data type certain **initial letters** of variable names. In the case of FORTRAN, the letters I, J, K, L, M and N were taken to mean an integer data type, all others implying a floating point data type. Thus in FORTRAN the variable name COUNT would be taken to be representing a floating point number; if an integer was required, the variable name would be changed to ICOUNT.

Structured data types need to be declared in all languages before they are used. The reason for this is that often large amounts of storage have to be allocated for the data items.

**Arrays** are declared in such a way that the size of the array is given together with the simple data type which is to be stored in that array. For example, in Pascal an array to contain the twelve monthly average temperatures (see Chapter 4) may be declared as follows:

```
var
 average_temperature : array [1..12] of real;
```

In Visual Basic the same declaration would be made using the following code:

```
Dim temperature(1 To 12) As Single
```

In declaring a **record** data structure, the name and simple data type of each of the fields needs to be declared. This is done in the following way in Pascal:

```
type
 details = record
 initial : char;
 surname : packed array [1..20] of char;
 age : integer;
 height : real
 end;
```

The variable used in the program is then given the data type 'details' in the normal way in the variable declaration. This is how that is done:

```
var
 person : details;

```

To access one of the fields in the record, for example if we wished to increase the age by 1, then the following statement would be used in Pascal:

```
person.age := person.age + 1;
```

This statement has the effect of retrieving the value of the field 'age' from the 'person' record, adding 1 to the value and then storing that value in the field 'age' of the 'person' record.

A **sequential file** is processed from the first record until a sentinel value (usually an end-of-file or EOF character) is found.

## Exercise 5.2

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

The three basic control structures of computer languages are \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_. Keywords to look for in selection are \_\_\_\_\_ and \_\_\_\_\_. The binary selection usually uses the keyword \_\_\_\_\_ and the mutway selection use \_\_\_\_\_ or smethnsimilar Repetitions are also known as \_\_\_\_\_ and are generally identified by keywords such as \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_. The counted loop will contain the keywords \_\_\_\_\_ and \_\_\_\_\_. Pre-test repetitions have the \_\_\_\_\_ at the \_\_\_\_\_ of the loop, whereas \_\_\_\_\_ have the \_\_\_\_\_ at the end of the loop.

- 2 Using the programming language of your choice, write a statement which will:
- Define the variable `counter` as an integer.
  - Define the variable `name` as a string of length 15 characters.
  - Define the variable `financial_member` as Boolean.
  - Define an array `months_sales` as being an array with 31 elements.
  - Define a record type called `stock` which contains the fields `item_id` which is a six-character string, `quantity` which is an integer, and `cost_price` and `markup` which are both floating point numbers. Define a variable of type `stock` which has a variable name `item`.
- 3 Code each of the following actions in your chosen language.
- Add 1 to the variable `counter` and store it back in `counter`.
  - Set the value of `pi` to 3.14159, then calculate the circumference of a circle that has a radius given by the value of `radius`. (Remember  $C = 2\pi r$ .)
  - Calculate `profit` from `cost_price` and `sell_price`.
  - Award a grade of 'GOOD' for a `mark` of 80% or better, and a grade of 'FAIR' for a `mark` of less than 80%.
  - Use a counted loop to count to 100 and print the even numbers up to 200.
  - Use a pre-test repetition to display the even numbers from 0 to 200.
  - Use a post-test repetition to display the numbers from 0 to 200.
- 4 Check your answers to question 3 parts **b** to **g** by completing a program to perform each of the tasks. You will have to add some input and output statements and possibly some variable declarations depending on your chosen language. Test your programs by running them on a computer
- 5 Write a small program to set up an array of ten items called `evens`. Use a counted loop to put the even numbers from 2 to 20 into the array elements 1 to 10 respectively. Print out the array elements in order. Test your program by running it on a computer.
- Extension:* Modify the program so that the array elements are printed out in reverse order (that is, from 20 down to 2).
- 6 Construct a program that will input the rainfall for each of twelve months and then print out the total average rainfall together with the average of the monthly rainfalls. Test your program by running it on a computer
- Extension:* Create an array of strings and enter the month names into the array. Print out the month's name together with the average rainfall for that month.

- 7 Write a portion of code in your chosen language that will set up a record type, called `results`, that contains the following fields:
    - `student_name` a string (of length 25 if needed)
    - `test_mark1` a floating point number
    - `test_mark2` a floating point number
    - `average` a floating point number
    - `grade` a character
  - 8 Use your portion of code from the answer to question 7 to declare a variable called `student` that has a data type of `results`.
  - 9 Use your answers to questions 7 and 8 to declare an array of records that will contain 30 student records, each of data type `results`.

*Extension:* Create a program that will input up to 30 names together with two marks for each student and print out the student's name and his or her total mark. Test your program by running it on a computer.
- 

## Error-correction techniques

### Types of coding errors

Most programs when first coded will contain errors of one kind or another. The errors will fall into one of two categories: errors in syntax or execution errors. Frequently, logic errors will have been removed by this stage by following the sequence of the program development cycle. However, some logic errors may only surface when the program is running and is able to be tested using the appropriate test data.

### Syntax errors

**Syntax errors**, or **compile-time errors**, are identified by the programmer as written statements that do not conform to the rules of the language. This type of error is, therefore, dependent on the language used. Variations in the implementation of languages may render a legal statement on one computer platform illegal on another. Most syntax errors, however, occur through typographical mistakes when the code is being prepared on the text editor. Care with typing can assist in reducing the number of errors. A further aid in error detection, with both syntax and run-time errors, is to have one or more people desk check the program steps as if they were the computer compiling and running the program. (For a more complete description of desk checking, refer to Chapter 4.) As far as possible, the author of the program should not participate in this check.

Early compilers were very curt in providing indication as to the problem in the source code. Statements such as `ERROR 002 IN LINE 23` or `ERROR 35` gave little indication of the form of error; this had to be determined by the programmer from the compiler documentation. Early personal computers, such as the Apple II, often came with a BASIC interpreter in ROM. These interpreters, again, were very curt in their error indication, issuing the message `SYNTAX ERROR IN LINE ...` but giving no indication as to the form of the error. The main reason for the brevity of the message was the need to create the compiler or interpreter so that it would occupy minimal memory. As computing technology has advanced and memory is available in larger quantities, the assistance that can be provided by the interpreter has increased. Current-generation compilers and interpreters

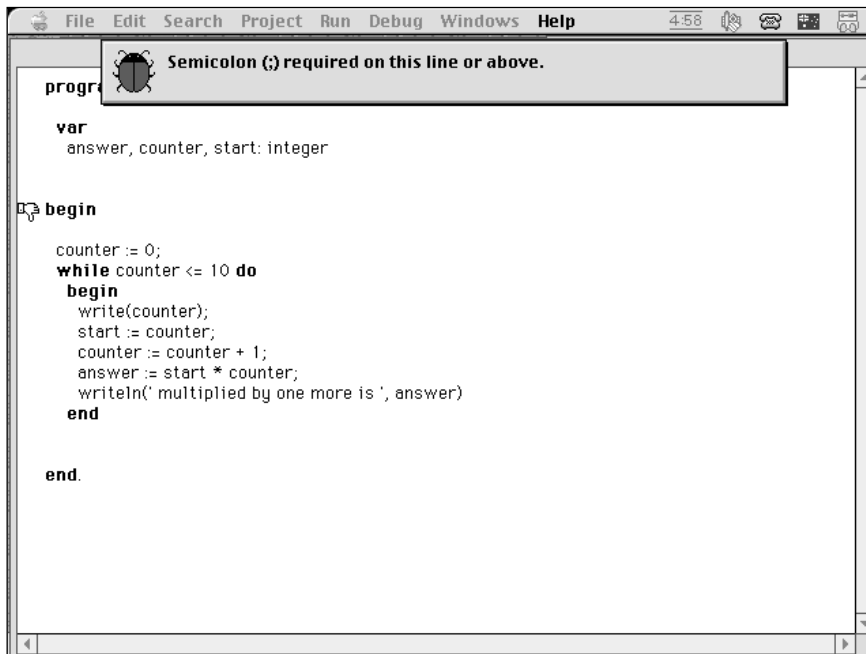


Figure 5.17 A syntax error message in THINK Pascal for the Macintosh.

assist the programmer by providing not only a message stating the error but also an indication as to where the error might occur and how it might be corrected.

Common causes of syntax errors include:

- missing or wrong statement punctuation, for example failure to close parentheses, missing commas, missing semicolons (see Figure 5.17) or the wrong type of parentheses for the statement type
- typographical errors in reserved words, for example the word REPEAT being entered as REPET
- failure to complete groupings, for example following a REPEAT with UNTIL at the end of a loop
- incomplete program statements, for example starting an IF statement but omitting a condition statement such as WAGE > 40000.

Compile-time errors may be caused by other factors besides a badly constructed program statement. In languages that require identifiers to be declared before use, a syntax error will be generated by the use of an undeclared identifier (see Figure 5.18). The cause of an undeclared identifier may be either a typing mistake or the inadvertent omission of the identifier from the declaration section of the program.

Other syntax errors include:

- a mismatch in the types of variable within an assignment statement, for example trying to assign a numerical value to a variable of type character.
- the use of a reserved word as a variable identifier or, in some languages, within an identifier. (For example, the use in Pascal of the variable `end_value` may be rejected by the compiler as `end` is a reserved word.)
- compiler only being able to handle strings of characters up to a certain length (for example in displayed messages) or of a certain complexity in calculation. In both cases the errors can be eliminated by splitting the text or calculation into two or more parts, each occupying its own statement.

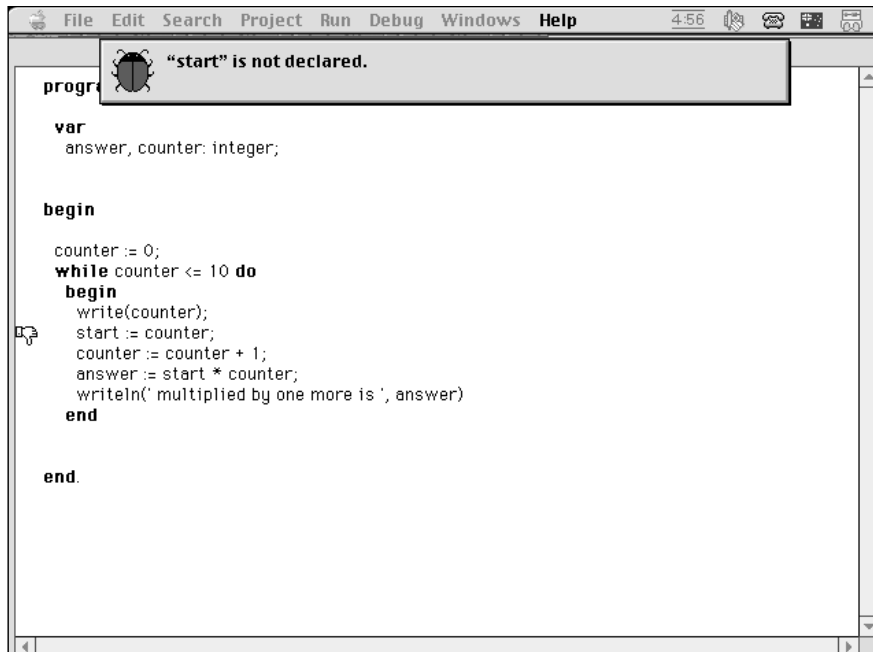


Figure 5.18 An undeclared identifier error.

### Run-time errors

Syntax errors are relatively easy to locate and correct, but **run-time errors** (or **execution errors**) can be difficult to locate. A run-time error in a program may not be evident at the point at which it occurs, but it may surface later, causing all kinds of trouble in the program.

The first kind of error involves one or more data values that cause the computer to attempt a calculation for which there is no result or for which the result is not as anticipated. For example, a division by zero is unable to be evaluated and so the operating system will cause an error to be generated. This type of error will be signalled at the point in the program where it occurs, usually halting execution, so it is easy to locate and correct. Other errors which fall into this category are those in which the tangent ratio of  $90^\circ$  is to be used and some results of integer arithmetic.

### Example 14

Integer arithmetic can cause problems if the result of an operation is outside the range allowed for that machine (typically between  $-32\,768$  and  $32\,767$ ). If, for example,  $32\,766$  and  $32\,760$  are added using integer arithmetic, a result of  $-10$  occurs, which is clearly not the correct result. This problem may be overcome by reordering the operations in the statement or choosing a different data type for the variable(s).

Calculations performed by a computer using real numbers are subject to errors caused by the inability of the computer to exactly store decimal values. A small part of the value may be 'lost'. This process is called **truncation**. In many cases the errors are insignificant and will not cause problems for the programmer.

However, if there are a large number of calculations to perform, the order of operations may make a difference. In this case a reordering of the calculations may improve the accuracy.

A further problem caused by the manner in which real numbers are stored is that of **equality**. If a decision is to be made on whether a value is equal to another or not, a wrong result may be obtained if the tested value is close. The reason for this is again the manner in which calculation results are stored in main memory. The following BASIC program illustrates this problem.

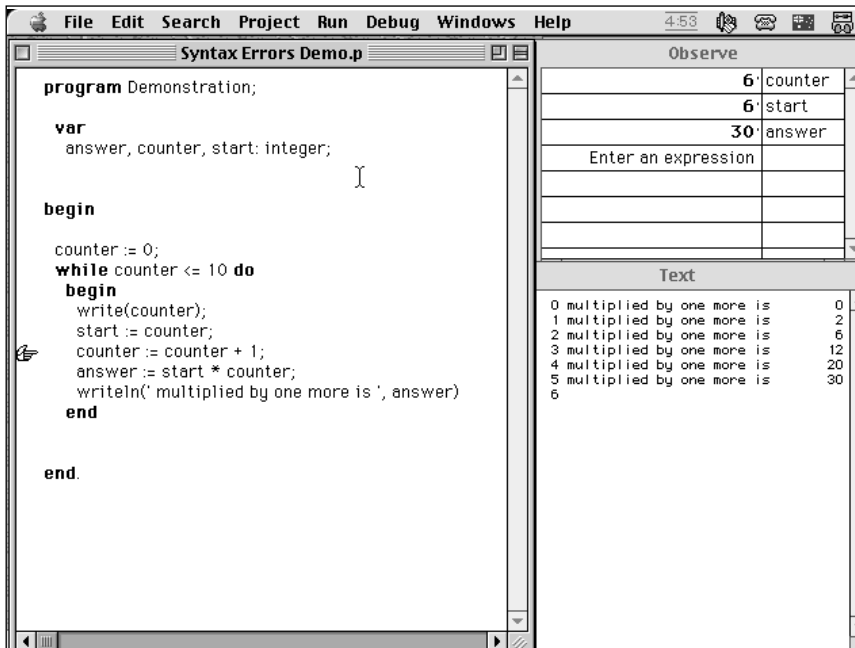
### Example 15

```
10 INPUT NUMBER
20 ANSWER = NUMBER / 9999999999
30 ANSWER = ANSWER * 9999999999
40 IF ANSWER = NUMBER THEN PRINT "equal" ELSE PRINT "not equal"
50 END
```

The above program does not perform the expected match as the stored value of ANSWER is truncated in line 20. When the multiplication in line 30 takes place, a slightly different value is obtained, so the message 'not equal' is displayed.

By rewriting the program in the following manner, a correct result is obtained each time the program is run:

```
10 INPUT NUMBER
20 ANSWER = NUMBER / 9999999999 * 9999999999
40 IF ANSWER = NUMBER THEN PRINT "equal" ELSE PRINT "not equal"
50 END
```



**Figure 5.19** THINK Pascal for the Macintosh allows the programmer to step through a program and view the values of the variables at each stage of execution.

The ability to step through a program one instruction at a time and the ability to display the values of each of the variables at each stage of execution will often help in locating and correcting these run-time errors. Once the run-time errors have been located, they need to be corrected. The process of correction may be as simple as changing a line of code or adding a new line, or it may require a rewriting of an algorithm and coding a completely new subprogram.

### Logic errors

Most logical errors should be identified during the algorithm description stage of the design process. However, the differences between an algorithm description method and the language used for the program may require that an action is not coded in exactly the same way as it has been shown in the algorithm. This type of logical error may involve control not following the paths as designed in the algorithm description. This kind of error is much harder to rectify and involves using techniques such as setting breakpoints and tracing the flow of control.

**Breakpoints** are places in the program where execution is temporarily suspended so the programmer can determine whether the program flow reaches that point. A breakpoint is often placed after a printout of values of variables. This allows the programmer to examine the values before resuming the program.

**Tracing** refers to a display on the screen of the path taken during execution of a program. Some languages support a trace function (for example many versions of the BASIC language), but statements can be placed in the code of a subprogram to achieve the same result. These statements may be simple messages such as `This is the barcode search module.`

## Exercise 5.3

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Three types of error can creep into a computer program. The three types are \_\_\_\_\_ errors, \_\_\_\_\_ errors and \_\_\_\_\_ errors. Those errors coming from the algorithm are known as \_\_\_\_\_ errors. Those errors picked up when the program is being translated are called \_\_\_\_\_ errors. A \_\_\_\_\_ error occurs during program execution. Errors in the algorithm are usually found during \_\_\_\_\_, although some may not be found until the program is \_\_\_\_\_.

- 2 Explain the term 'syntax error' in your own words. Give examples of syntax errors to illustrate your answer.
- 3 There is an error in each of the following segments of code. Identify the most likely error and correct it.
  - a `COST PRICE-DISCOUNT`
  - b `while counter < 10 do begin  
time := 0`  
(Use the Pascal syntax diagrams in the appendix to help with this question.)
  - c `write('This message will appear on the screen to help you');`  
(Use the Pascal syntax diagrams in the appendix to help with this question.)
  - d `if then index := index + 1`  
(Use the Pascal syntax diagrams in the appendix to help with this question.)



- 4 Examine the following program in Pascal and identify as many syntax errors as you can. Correct each error and compile the program to discover the errors you may have missed. (Use the Pascal syntax diagrams in the appendix to help with this question.)

```
program question_3
 var
 answer, counter : integer;
begin
 counter := 0;
 while counter <= 10 do
 begin
 write(counter);
 start := counter;
 counter := counter + 1
 answer := start * counter;
 writeln('multiplied by one more is', answer)
 end
end
```

- 5 Examine the following program using LOGO turtle graphics and identify as many syntax errors as you can. Correct each error and run the program to discover the errors you may have missed.

```
TO SQUARE
 REPEAT 4[FORARD 100 RIGHT 90]
END
TO CIRCLE
 REPEAT 360[FORWARD 1 RIGHT 1
TO HOUSE
 SQUARE
 TRIANGLE
END
TO TREE
 FORWARD 100
 RIGHT 90
 CRCLE
 LEFT 90
 BACKWARD 100
END
```

- 6 Examine the following GW-BASIC code and identify as many syntax errors as you can. Correct each error and run the code to discover the errors you may have missed. (You may have to modify some of the code to run on your system.)

```
10 COUNT = 0
20 WHILE COUNT < =
30 PRINT COUNT;
40 START COUNT
50 COUNT = COUNT + 1
60 ANSWER = COUNT START
70 PRINT " multiplied by one more is ;ANSWER
80 WEND
90 END
```

- 7 Examine the following code in Hypertalk and identify as many syntax errors as you can. Correct each error and execute the code to discover the errors you may have missed. You will need to create two card fields called Fahrenheit and Celsius.

```
on mouseUp
 set numberFormat to 00.00
 get card "Fahrenheit"
 subtract 32 from it
 multiply it by 5
 divide it by 9
 put it into card field "Celsius"
end
```

- 8 Describe in your own words the meaning of the term 'run-time error'. Give examples to illustrate your answer.
- 9 Describe the steps that can be taken to identify where a run-time error has been created.
- 10 Create a small program which asks for the entry of two numbers and outputs their product. Write the code in two different ways: the first way treating both numbers as integers, the second way treating the numbers as floating point values. What effects does each of these methods have on the stored result?
- 11 Explain how you could trace the path through a program while it is being executed. What purpose would a trace serve in the debugging process?
- 

## Stubs

Top-down programming often leads to the use of subprograms or modules. The ways in which the modules work together have to be tested to determine whether the desired result will be achieved. A programmer cannot sensibly wait for a whole program to be coded and translated before performing these tests. Many different causes can prevent the proper action of the program and, with programs becoming more complex, debugging may become almost impossible. One widely used method of testing involves creating small modules that represent parts of the program which have yet to be written. These modules are called **stubs**.

A stub may consist of a message which indicates to the programmer that a particular portion of the program has been reached or it may set a particular variable's value. We will examine the use of each type of stub in the following simple examples.

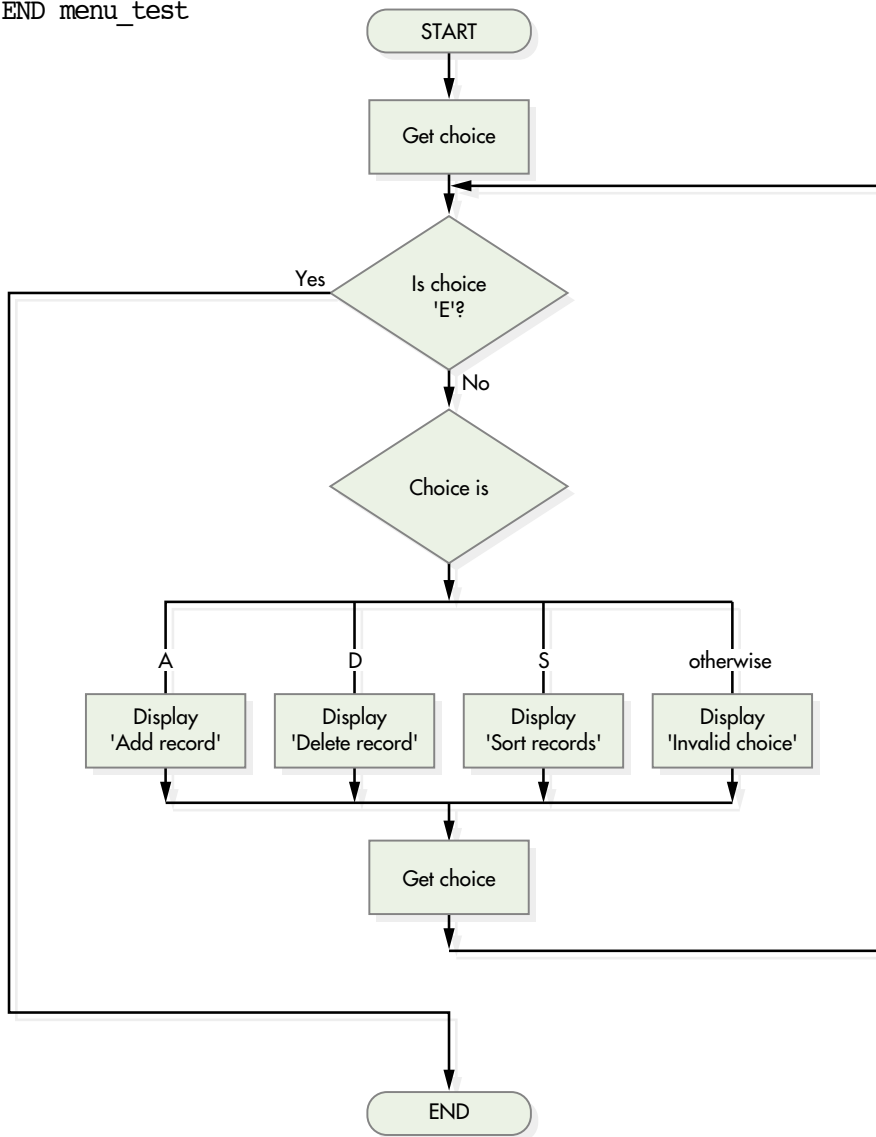
### Example 16

Stubs in this example are used to test the menu module of a program. The menu consists of four choices: add a record, delete a record, sort the records and exit the program. The procedures for adding, deleting and sorting records have not been written in full. Instead they have been replaced by stubs so that the main logic can be tested.

*continued next page*

*Pseudocode*

```
BEGIN menu_test
 get choice from user
 WHILE choice is not 'E'
 CASEWHERE choice is
 'A' : display 'Add record'
 'D' : display 'Delete record'
 'S' : display 'Sort records'
 OTHERWISE display 'Invalid choice'
 ENDCASE
 get choice from user
 ENDWHILE
END menu_test
```



**Figure 5.20** Flowchart showing the use of stubs to test a menu.

*continued next page*

A Pascal program corresponding to the above algorithm would be:

```
PROGRAM menu_test (input , output);
VAR
 choice : char;
PROCEDURE Add_record;
 BEGIN
 writeln('Add record stub')
 END;
PROCEDURE Delete_record;
 BEGIN
 writeln('Delete record stub')
 END;
PROCEDURE Sort_records;
 BEGIN
 writeln('Sort records stub')
 END;
PROCEDURE Invalid_input;
 BEGIN
 writeln('Invalid choice stub')
 END;
BEGIN
 writeln('Please press the letter key corresponding to your choice');
 writeln('A to ADD a record');
 writeln('D to DELETE a record');
 writeln('S to SORT the records');
 writeln('E to END this session');
 readln(choice);
 WHILE (choice <> 'E') AND (choice <> 'e') DO
 BEGIN
 IF choice IN ['A','a','D','d','S','s'] {this statement ensures
 that other characters are excluded; otherwise...}
 THEN CASE choice OF
 'A','a' : add_record;
 'D','d' : delete_record;
 'S','s' : sort_records
 END
 {end of case statement}
 ELSE Invalid_input;
 writeln('Please press the letter key corresponding to your choice');
 writeln('A to ADD a record');
 writeln('D to DELETE a record');
 writeln('S to SORT the records');
 writeln('E to END this session');
 readln(choice);
 END
 {end of while statement}
END.
 {end of program}
```

## Example 17

Stubs in this example are used to set values so that part of the test data can be used.

A program has been designed to input data from a file stored on disk and to search through that data. The programmer wishes to test the search algorithm but does not want to input each of the data elements each time and has not yet written the disk input/output modules. A series of assignment statements is used to provide the necessary data elements. In this case the algorithm is one which has a number of friends' names and addresses. The user is required to enter an address, and the name of the person living at that address is displayed.

A search algorithm (known as a **linear search**) for a particular address within this array follows. (This type of search looks through each element of the array in turn for the wanted item. You may like to modify the algorithm and use it for your programs.)

| Algorithm                                                      | Comments                       |
|----------------------------------------------------------------|--------------------------------|
| BEGIN                                                          |                                |
| set flag to 0                                                  | Initialise variables           |
| set index to 0                                                 |                                |
| set friend[1].name to 'Jane Soo'                               | Begin stubs to set             |
| set friend[1].address to                                       | the array element to test      |
| '11 Marion St Goolga 2990'                                     | data values                    |
| set friend[2].name to 'Felix Kato'                             |                                |
| set friend[2].address to                                       |                                |
| '2A Beacon Dr Aulswell 2828'                                   |                                |
| set friend[3].name to 'Kim Kong'                               |                                |
| set friend[3].address to                                       | End stubs with test data       |
| 'Box 8751 GPO Hapion 3876'                                     | Input search item              |
| get wanted_address from user                                   | Search the array one item at a |
| REPEAT                                                         | time for a match               |
| set index to index + 1                                         |                                |
| IF friends [index].address                                     |                                |
| = wanted_address THEN                                          |                                |
| set flag to index                                              |                                |
| ENDIF                                                          |                                |
| UNTIL (index = last) OR (flag <> 0)                            |                                |
| IF flag <> 0 THEN                                              |                                |
| print 'The person living at that address is' friend[flag].name |                                |
| ELSE                                                           |                                |
| print 'The address you want is not in the list'                |                                |
| ENDIF                                                          |                                |
| END                                                            |                                |

*continued next page*

A Pascal coded program for this search is as follows:

```
PROGRAM address_book (input, output);
CONST
 last = 3;
TYPE
 person =
 RECORD
 name : string[20];
 address : string[50]
 END;
VAR
 friend : ARRAY [1..last] OF person;
 wanted_address : string[50];
 index , flag : integer;
BEGIN
 flag := 0;
 index := 0;
 friend[1].name := 'Jane Soo'; {Begin data stubs}
 friend[1].address :=
 '11 Marion St Goolga 2990';
 friend[2].name := 'Felix Kato';
 friend[2].address := '2A Beacon Dr Aulswell 2828';
 friend[3].name := 'Kim Kong';
 friend[3].address :=
 'Box 8751 GPO Hapion 3876'; {End data stubs}
 writeln('Please type in your friend''s address');
 readln(wanted_address);
 REPEAT
 index := index + 1;
 IF friend[index].address = wanted_address
 THEN
 flag := index
 UNTIL (index = last) OR (flag <> 0);
 IF flag <> 0
 A THEN
 writeln('The person living at', wanted_address,
 'is', friend[flag].name)
 ELSE
 B writeln('The person you want is not in the list')
END
```

If, for example, the user input 11 Marion St Goolga 2990 to test the program, what would the expected output be?

## Exercise 5.4

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.  
A ~~is a short~~ piece of code used to represent an unwritten part of code.  
A \_\_\_\_\_ may be used to test the workings of the main or \_\_\_\_\_ module of a program. Using stubs allows the programmer to test the workings of a \_\_\_\_\_ without having to \_\_\_\_\_ code it. Stubs may also be used in \_\_\_\_\_ to provide \_\_\_\_\_ values rather than having to enter them each time the program is \_\_\_\_\_.
- 2 What is the meaning of the term 'stub' when applied to the debugging process? Explain, using examples, how stubs may be used during the development of a computer program.
- 3 Describe the different types of stub, illustrating your answer with examples.
- 4 Write an algorithm for a bulletin board logon module which requests a user's identity and password. If the user is a new user (indicated by the identity 'NEW'), control passes to a new user module and asks for various details, following which access to the board is given. For a registered user, control passes to the security module which determines the user's access rights and allows access to the board. Code and test your algorithm using stubs to represent the various modules.
- 5 Write an algorithm which finds the largest and smallest members of a ten-element array of integers. Code and test this algorithm in an approved language, using stubs to provide the array members instead of the normal input statements. Use the following ten values as test data: 52, 69, -4, 45, 74, 89, 0, -81, 91, 5.

### Debugging output statements

The process of error detection does not end with the correct working of the algorithm and error-free coding. It is often necessary to trace the values of variables as they change during the execution of a program in order to detect where the problem comes from. One of the most widely used tracing methods is to use output statements to display or print the values of various variables at each stage of processing. By using this technique, the progress of data items can be monitored during the testing stage. (Note that some programming environments, such as Visual Basic and THINK Pascal, have the ability to trace the value of a variable without the programmer having to add debugging output statements to the code.) On the completion of the testing stage, these debugging statements may be removed from the program, leaving only those program statements that are necessary for the program to function correctly. Following are some general problems in which debugging output statements can help the debugging process.

#### Example 18

A program which reads data from disk and processes it cannot be tested for correct input until the coding process is completed. A set of output statements can be written which immediately 'echoes' the data read to the screen. In this way the programmer can examine the data before processing begins, ensuring that the dataset used in the program is being read correctly.

## Example 19

Programs incorporating a number of loops may contain errors in one or more of the loops. For example, a program may be required to print out all the times from midnight (0:00) to 23:45 in quarter-hour increments. If the looping is wrongly constructed, we will end up with the incorrect output. When coded, the program can have debugging output statements inserted where the hours and quarters are changed in order to follow through the order of the changes. The following algorithms using counted loops illustrate how the placement of one loop inside the other is important. Perform a desk check on each of the algorithms to determine which one works properly. Code both of the programs in your chosen language to find the error. In the algorithm with errors, place two output statements after each of the FOR statements to display the change that has been made to the variable.

### Algorithm 1

```
BEGIN
 FOR quarter_count goes from 1 to 3
 FOR hour_count goes from 0 to 23
 set minutes to 15*quarter_count
 print hour_count, ":" , minutes
 NEXT hour_count
 NEXT quarter_count
END
```

### Algorithm 2

```
BEGIN
 FOR hour_count goes from 0 to 23
 print hour_count, ": 00" ,
 FOR quarter_count goes from 1 to 3
 set minutes to 15*quarter_count
 print hour_count, ":" , minutes
 NEXT quarter_count
 NEXT hour_count
END
```

## Exercise 5.5

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.  
A \_\_\_\_\_ is used to display the value of a \_\_\_\_\_ at a point in the program being tested. This \_\_\_\_\_ is then removed after testing. Another use of this type of statement is to help \_\_\_\_\_ the flow of control through the \_\_\_\_\_. Some language systems have a function which allows the \_\_\_\_\_ through the program to be followed.
- 2 Explain, with examples, the purpose of debugging output statements.
- 3 Choose a program you have already written in your chosen programming language. Place some debugging output statements in appropriate places within the program (for example at the beginning, inside and after a loop) and follow the values of the variables as they change during the execution of the program.



## Flags

As programs become more complex, it is easy to become lost in the maze of coding. It is sometimes necessary to record whether a certain condition has been met and to act on that condition later in the program. A **flag** is a variable which is used to indicate or 'flag' whether an event has occurred or not. Such events may be the reading of a negative value, or whether a matching value has been found in a list.

A flag is usually created as a Boolean variable (able to be set to either 'true' or 'false'). It is set to false at the beginning of the program, then has its value changed as control enters the part of the program which is to be 'flagged'. By using an output statement at some appropriate place after this part of the program, the flag can be displayed, its value indicating whether the desired actions have taken place or not. Flags can be particularly useful in the program testing by indicating whether a particular piece of code (for example a sub-program) has been called.

### Example 20

A program has been written which contains a number of calls to various sub-programs, but it does not work properly. The programmer decides to use flags to indicate whether a subprogram has been called. This is accomplished in a manner similar to the following.

A flag is created as a global variable of type Boolean. (This is needed as we wish to mark whether a subprogram has been entered. A variable which is local to the subprogram will not be valid outside the subprogram for which it has been declared.) The flag is set to 'false' at the beginning of the program; the same variable is assigned 'true' within the subprogram. At an appropriate point in the program (for example when the output from that subprogram is used) an IF statement is used to read the flag variable and print, or display, an appropriate message.

In Pascal, for example, the output statement would appear thus:

```
If subprogram_flag then
 writeln('The subprogram has been used')
else
 writeln('The subprogram has not been used');
```

(Since `subprogram_flag` is of type Boolean, it can only have the value 'true' or 'false'; therefore there is no need to write `If subprogram_flag = true then ...`)

## Exercise 5.6

- 1 Describe, using examples, the purpose of a flag.
- 2 Why is a flag usually declared as a Boolean variable?
- 3 Australia Post has created a computer program for its letter-sorting machine in Sydney. The program reads the postcode of a letter and determines which State it is to be delivered to, by using the first digit of the postcode. Unfortunately, the program does not work for letters to the ACT (since ACT postcodes begin with the same digit as those in NSW). Explain how flags may be used to discover the error in the program. Write an algorithm and a program which inputs a postcode and determines which State or Territory the letter is to be delivered to.

## Libraries of code

As we have already seen, modules from other sources can be used to reduce the amount of development needed for an application. When stored for later use, pre-written modules form what is known as a **library**. We may also use code from other sources, modifying them to suit our own needs. Sources of code will include user-written modules, modules from the Internet (or other widely available sources) and modules or templates included as a part of a development system. When using modules other than those developed by yourself you should always be mindful of the legal and ethical considerations such as copyright.

### Reusable code

When writing modules we should look at keeping them as independent from the main program as possible. In this way we can ensure that the module can be reused with the minimal possible effort. This can be achieved by using local variables where possible and defining identifiers for constant values within the module rather than using the actual value in the module. By choosing to write modules in this way, we can use them in other applications with very little modification.

A very common process used in computer applications is the **validation** of data items. The process of validation checks that data items are entered into the system in an appropriate form or within a particular range. The following algorithm can form the basis of any data validation module, as we will see.

#### *Pseudocode*

```
BEGIN data_validation module
 set data_item to user input
 WHILE data_item does not meet the requirements of the program
 display message 'This value is invalid, please check and
 re-enter the item'
 set data_item to user input
 ENDWHILE
END data_validation module
```

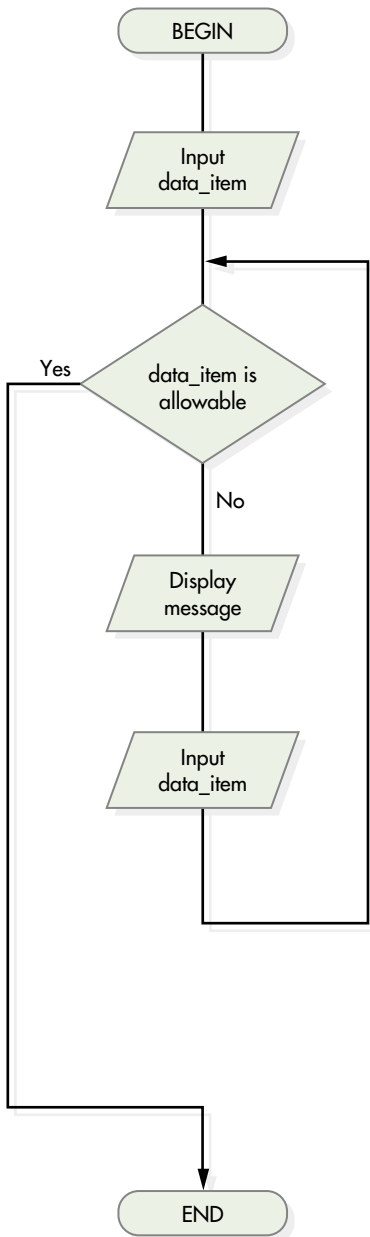
The flowchart is shown in Figure 5.21.

This module can now be used within a program. For example, if test marks required for a markbook program must be in the range 0 to 100 inclusive, the validation algorithms can be modified in the following manner:

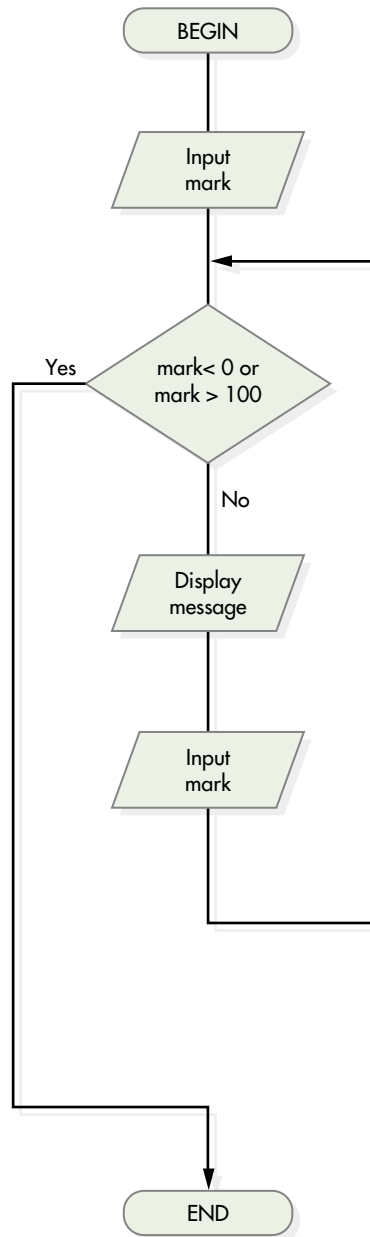
#### *Pseudocode*

```
BEGIN data_validation module
 set mark to user input
 WHILE (mark < 0) or (mark > 100)
 display message 'This mark is out of range, please check
 and re-enter the mark'
 set mark to user input
 ENDWHILE
END data_validation module
```

The flowchart is shown in Figure 5.22.



**Figure 5.21** Flowchart of data validation module.



**Figure 5.22** Flowchart of mark validation module.

## Exercise 5.7

- Using the general data validation algorithm described in the above section as a template, write an algorithm that will accept values of the variable size within the range 100 to 1000 inclusive. Use the appropriate computer application to present your answer.

- 2 Write a data validation module that will accept the inputs of height and weight, ensuring that both are positive with the height less than 3 and the weight less than 500. Present your algorithm as a computer file called SIZE.
- 3 This algorithm accepts a date in the form dd-mm-yyyy and prints it in the form dd month year. (For example, an input of 25-05-2004 will output 25 May 2004.)

*Pseudocode*

```

BEGIN datechange(day, month, year)
 CASEWHERE month is
 01 : set month_name to 'January'
 02 : set month_name to 'February'
 03 : set month_name to 'March'
 04 : set month_name to 'April'
 05 : set month_name to 'May'
 06 : set month_name to 'June'
 07 : set month_name to 'July'
 08 : set month_name to 'August'
 09 : set month_name to 'September'
 10 : set month_name to 'October'
 11 : set month_name to 'November'
 OTHERWISE : set month_name to 'December'
 END CASE
 output dd, month_name, year
END datechange

```

Modify this algorithm to change the date format from the form dd month year and output the numerical form dd-mm-yy. (For example, an input of 25 May 2004 will output 25-05-2004.) Present your algorithm in an appropriate form.

- 4 Use the CASEWHERE statement in an algorithm that will change the written form of a number into its numerical form. The algorithm should be able to input the words 'one' to 'ten' and output the values 1 to 10. (For example, if the input to the algorithm is the string 'six', the output will be 6.)

## EXTENSION

- 5 Use your answer to question 4 as a template to input the word numbers 'twenty one' to 'ninety nine' and output the appropriate numerical form. (For example, an input of 'sixty seven' will output the number 67.)
- 6 Modify your algorithm so that it will accept all word forms of the numbers 'zero' to 'ninety nine' and output the appropriate numerical values.
- 7 Using an appropriate language, code and run one or more of the modules developed within this section.

---

## Combining code and modules from different sources

As we have seen, it is economical, both in effort and development time, to reuse modules and code whenever possible. The sources of code used in this way are as varied as using one's own code from a previous development through to standard modules or processes. The amount of effort needed to modify a particular coded module can often be reduced by the use of a few simple rules when coding modules.

These rules are the basic guidelines that should be used when writing modules whether or not you see a need for them in the future:

- Write each module as an independent unit, using local variables for processing. The external data items can be entered as parameters. For example, a sorting module may be written to sort the array called `name`. Rather than using this array name in the module, a local array called `data` could be used. The items from the input array are transferred to the local array as the first part of the algorithm.
- Any constants that appear in the module should be defined with constant names at the beginning of the module (if the programming language being used allows this), or local variables should be used, with the variable being assigned the value at the beginning of the module. For example, in the sorting algorithm the number of elements in the array, and hence the largest index number to be used, may be given the identifier name `last`. Thus when the module is used with an array of size 20, `last` would be set to 20. If the module were used later with an array of size 1000, then setting `last` to 1000 would allow the module to read all the data from the new set.
- Design modules so that, even though a process may not be exactly the same as that designed, similarities in function can be used as a basis for the new module. For example, if a module has been written to input the values into an array, there is a great similarity to one in which the elements of the array are printed out. (The only difference is the use of an output statement in place of an input statement inside the loop.)

If the code modules are saved as text files, the process of reuse within a particular application becomes a simple matter of replacing old values with those required by the new application.

Most programming languages support the use of modules as subprograms and as functions. **Subprograms** are completely self-contained parts of a program that have only one starting point and only one end. They may be **called** at any time during program execution and will always return control to the instruction immediately after the calling instruction. Subprograms may or may not pass values to or from the **calling module**. **Functions** are sets of instructions which always return a single value from any input data.

Regardless of whether the module is designed as a subprogram (sometimes called a **procedure**) or as a function, the designer should always ensure that the code is independent of the module calling it. As we have seen, the use of local variables and the use of constant identifiers will help in this regard. The link between modules and the main program is created by the use of global variables known as **parameters**. When writing the module algorithm in pseudocode, it is common practice to put brackets in after the module name and enclose the parameters being passed within those brackets. For example, the following pseudocode algorithm draws a rectangle:

```
BEGIN rectangle(length, width)
 set right_angle to 90°
 FOR half goes from 1 to 2
 draw line length units
 turn right_angle
 draw line width units long
 turn right_angle
 NEXT half
END rectangle
```

When called from a program, the subprogram will have the parameters passed in the following manner: `rectangle(45,50)` would create a rectangle 45 units long and 50 wide and the statement `rectangle(side, up)` would produce a rectangle `side` units long and `up` units wide.

Notice that, when the module is called, the first value in the brackets replaces the first variable in the module definition and the second value replaces the second variable. This means that `rectangle(20,50)` will produce a different rectangle from `rectangle(50,20)`. You must keep this in mind when modules are reused.

## Exercise 5.8

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

When developing solutions, we may use \_\_\_\_\_ modules to \_\_\_\_\_ the time and effort needed to produce the solution. Three sources of such modules are \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_. Ethical and \_\_\_\_\_ aspects of using \_\_\_\_\_ from another source should always be kept in mind. Modules should be kept \_\_\_\_\_ from the main program as this helps when we decide to \_\_\_\_\_ them later. It is best to use \_\_\_\_\_ variables in a module as this makes for faster \_\_\_\_\_ when they are to be reused.

- 2 The following algorithm passes through five members of an array and adds them up to form a total.

*Pseudocode*

```
BEGIN sum(array)
 set size to 5
 set total to 0
 set count to 1
 WHILE count <= size
 set total to total + array(count)
 set count to count + 1
 ENDWHILE
END sum(array)
```

- a Modify the algorithm so that it:
    - i asks for 5 items to be input
    - ii displays the 5 items
    - iii allows 20 items to be input, then added
    - iv asks for 5 names and addresses to be input into an array of records stored as a sequential file
  - b Use your answer to part a above to write an algorithm that will read in five values, find their sum and print out the five values followed by their sum.
- 3 Use an appropriate database system to create a personal database of algorithms you have written or come across in your study. Fields in the database should describe the purpose of the algorithm, list the parameters used by the algorithm, describe the source of the algorithm (for example self, textbook) and whether the algorithm is of a function or subprogram.

# User interface development

The **user interface** is the only means by which the user communicates with the application. The system designer needs to keep this in mind when deciding the manner in which the user will work with the system.

There are several important factors that must be taken into consideration when the user interface is designed. One factor is the users' expectations of the system and these can best be found by consultation with the users. A second factor is the capabilities of the output system. Also ergonomic considerations need to be kept in mind as well as those of efficiency of operation. Consistency in design throughout the interface is very important so that users quickly gain confidence in the system's operation and the amount of user training and help needed are reduced.

## Consultation with users

When designing the user interface, one of the most valuable sources of information is the user. The value comes from two main areas: their user's experience with the old system and their expectations of the new one. The developer can use the experience of the user with the old system to benefit the new system by making a determined effort to gain as much information and input from the user as possible.

Many different tools are available to the developer, including interviews, surveys, questionnaires, observation and examination of documentation.

The *interview* process allows the developer to interact directly with the user and the user to feel part of the development process. Interview questions need to be carefully planned in order to gain the maximum amount of information from the user. This interaction with the user gives the developer the opportunity to follow up any unexpected responses. A disadvantage of the interview process is that the users may not give completely honest answers to the questions since the process is not anonymous.

An advantage of **surveys** is that the users are able to answer questions without the fear of their identity becoming known. Survey questions are usually framed in such a way that responses can be given from a number of choices or as short answers. This makes the data collected from the survey easy to collate. Questions in a survey have to be very clear as they cannot be clarified by the user asking questions.

A *questionnaire* has the advantages of a survey, but the responses to questions are usually much more detailed, and can take whatever form the user chooses. Information gained from a questionnaire will take more time to analyse than that from a survey, but the information may bring up aspects not anticipated by the developer.

The *observation* of a user interacting with a system can give a developer a great deal of insight into the way in which the interaction works. There are two ways in which observation may be used. The first way involves observing the user



**Figure 5.23** Many automatic teller machines still have a text-only display.

interacting with the current system. This is important as the developer can use the types of interaction that take place as the basis for the new interface design. The second way involves the use of a prototype of the new system. Various aspects of the new interface can be tested using this technique, and the user's reaction to each part of the interface is evaluated to provide a guide for further modification.

Aspects of the interface investigated during this activity include:

- The *type of interface* required (that is, whether a graphical or text-based interface is suitable). Although we tend to think of the GUI as the most effective interface, there are applications where hardware devices still rely on a text-based display. (For example, many automatic teller machines use a text-based LED screen for their interface.)
- The *system response times* needed to properly process requests. Speed of processing will determine the manner in which the interface works. For example, the interface used by a word-processing program will have different requirements from that for a real-time video editing application. (The processing time for a text-based document will be smaller as less data needs to be processed than for a full-motion video where there is a large amount of data.)
- The *manner in which the interface will work*. This aspect includes navigation between program segments and the methods employed to give the user a choice (for example whether choices are placed on a menu bar, displayed as a palette or displayed as a dialogue box with choices).
- *Security* considerations. Levels of security within a system will have to be carried through to the computerised solution. The effect of this is that the interface will have to cope with those levels and display various prompts and messages in an appropriate form.
- *Hardware* requirements (for example specialised input devices such as barcode readers). Different hardware elements have different interfaces. The programmer needs to take into account the physical and technical limitations of these interfaces when designing the way in which the user will interact with them. For example, a barcode reader may be directly interfaced to a workstation, giving a different interface from one that is portable and interfaced to the computer only when the data has to be uploaded to the terminal.

## User's and developer's perspectives

The developer and the user have two different perspectives of the system. The developer views the system in terms of the processes and data needed to produce a solution. Users, on the other hand, view the system as a means of achieving a desired result. Both perspectives are equally important in ensuring that the final system achieves the goals set by the requirements definition.

Developers are interested in a system from a problem-solving perspective, which means that they concentrate on the factors relevant to this task. These factors include:

- The forms of the *input data*. For example, a designer of an application to process speeding infringements will look at the data needed to produce the infringement notice. (These would include the speed limit, location, registration number of the vehicle, registered driver of the vehicle and the driver's address.) Some of the data items would come from an existing system (for example the driver and vehicle details), whereas other items would come from other sources (for example a scanned photograph).



- The *processing* required to turn the input data into output data. For the speed camera application, the process involves scanning the photograph for details of the location and speed limit (these may be set up by the camera operator to be recorded on each photograph) as well as locating and 'reading' the registration number of the offending vehicle. Calculation of the appropriate penalty would also be part of the programmer's job here.
- The format of the *output*. For example, the programmer would have to examine the form of the speeding notice sent to the offending driver and the way it is to be done.

The user's view of the speed camera is entirely different from that of the programmer in that the user needs to know what is to be entered, the sequence of steps to be followed and the appropriate tasks to perform with the output. For example, input may be a roll of film being placed into a scanner. The tasks performed by the user would be to start the program running and monitor the results of the processing. Output may be a number of infringement notices and addressed envelopes. Further processing of this output would be the user placing each notice in its appropriate envelope for posting.



**Figure 5.24** The processing of data from a speed camera illustrates the difference between a programmer's view of the system and a user's view.

## Screen design

### Design principles

The current generation of highly interactive computers relies heavily on a display screen to communicate with the user. It is through the user interface that the user of a program communicates with it and it also allows the program to communicate with the user. Through this interface, the user has to make choices and enter data. A poorly designed screen may prevent the user from effectively using the program, by contributing to fatigue, hindering navigation through the program or impeding data entry.

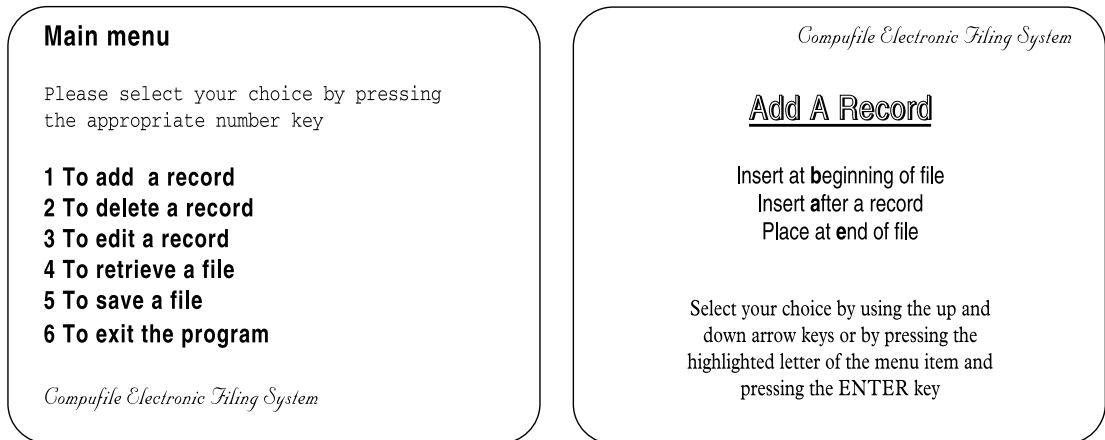
Many features contribute to a screen design, and all of them are equally important. The major concern in screen design is to provide an ergonomically sound interface involving the accurate transmission of messages to and from the user. The next sections examine the following design features: consistency, message structure and text features.

### Consistency in design

Design consistency involves the creation of screens which appear similar in design and have uniform commands and message placement. A user will rapidly gain confidence in using an application if the screen design is consistent; their actions become intuitive, with little or no thought being necessary to find a menu item or issue a command. Consistency involves the following:

- placement of like items in similar places
- choice of the same font and format to display the same type of information

- similar commands to make choices or to navigate to new screens
- similar highlighting methods and the use of the same screen divisions to separate differing screen elements
- appropriate use of colour to emphasise or de-emphasise screen elements (for example display options unavailable at that time).



**Figure 5.25** These two screens from the same program show a lack of consistency.

Compare the two screens from the same program in Figure 5.26. They illustrate some of the inconsistencies which should be avoided in screen design:

- inconsistent placement of items such as the screen heading and instructions
- different methods of choosing menu items
- different formatting of the screen items
- different fonts used to display the same type of information.

The two screens need to be redesigned to follow the same format. Inconsistency in design is not the only fault with these screens; they exhibit other problems which will be examined later.

## Message structure

Messages are a communication between the programmer and the user and are employed to assist the user with some aspect of the program. It is important that these messages are clear, concise and non-threatening so that they encourage confidence in the use of the program and give the required help.

Some early amateur programmers used error messages such as 'You pressed the wrong key, dummy!' or 'Wrong!'. These kinds of messages are threatening to inexperienced users and can undermine their confidence in using the program and even the equipment. Such messages are better expressed in terms of advice or help. For example, a wrong key press might be corrected by a statement such as 'Please press the number key corresponding to your choice'.

Long or imprecise instructions are also unsuitable, as they can be misinterpreted by the user or may contribute to a novice user's feeling of insecurity.

Other inappropriate messages include those that rely on humour, those that give the impression that the computer has human traits or a personality, and, of course, those that use inappropriate language such as vulgar terms and jargon.

## Text features

### Legibility

Legibility refers to the overall ability of the screen elements to be distinguished and interpreted by the user. Factors affecting legibility include the choice of fonts and the use of colour.

- A screen has a lower resolution than hard copy, so a font which works well as a printing font may not be as legible as a screen font. Also, the use of an excessive number of fonts can have a detrimental effect on the legibility of the displayed text.
- Some colours should not be used together as they produce a poor contrast, which reduces the legibility of the display items. For example, purple text displayed on a red background will be difficult to read.

In addition, all of the following text features affect legibility: spacing, use of uppercase and lowercase, use of borders, and justification and alignment.

### Text spacing

Text is still the most common form of communication on a screen. The spacing of text will affect both its readability and its perceived importance. Text areas that are isolated by space attract the immediate attention of the user and can therefore be used to highlight important information. The use of space, both horizontally and vertically, will also provide the user with information about the grouping, or non-grouping, of the various displayed messages. For example, a set of menu choices grouped together will be separated by smaller spaces than the menu items and instructions.

Space is also employed to avoid overloading a screen with text. Excessive amounts of textual information cannot be fully understood without the operator making a conscious effort, which thus reduces concentration on the computer task. This distraction will diminish the efficiency of the program, especially if the screen is used a large number of times. The problem can usually be overcome by splitting the screen into two or more related sub-screens (for example by using two or more screen displays in preference to a single crowded display, or using scrolling screens to present part of the information at a time).

### Use of uppercase and lowercase text

The excessive use of uppercase text is one of the most common screen design faults. Some people overuse this form of highlighting under the misconception that uppercase text is easier to read. Uppercase can be utilised for features such as major headings, but it should be used sparingly as it detracts from the readability of a screen. Compare the legibility of the two screens in Figure 5.27. In the screen using only uppercase, the text appears to be all at the same level. This reduces readability, as an effort has to be made to distinguish the menu elements from the menu header and instructions, as well as to distinguish individual words.

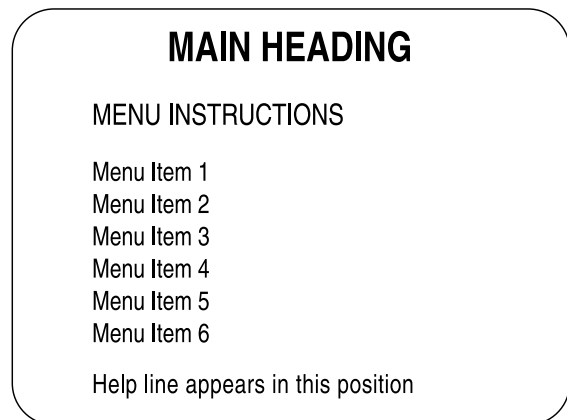
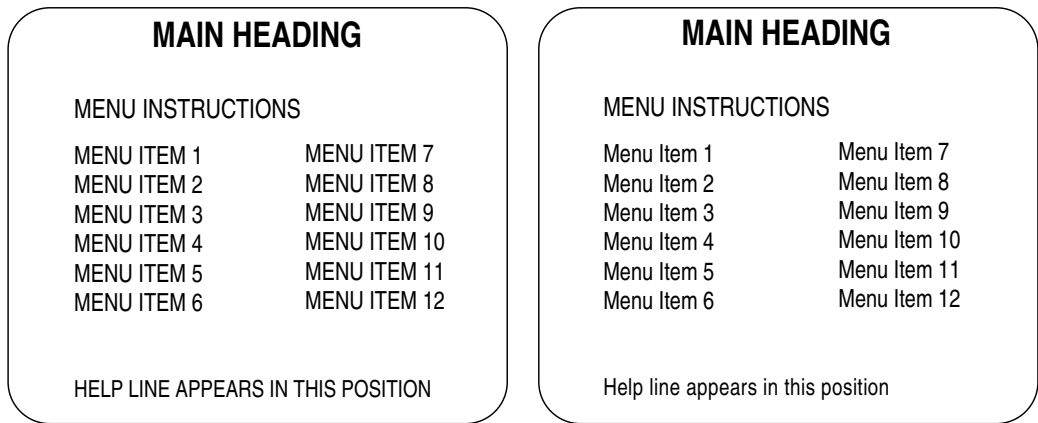


Figure 5.26 Spacing text helps group similar items.



**Figure 5.27** The use of both uppercase and lowercase text to display information is clearer than using uppercase text only.

### Text colour

As mentioned previously, the colour of the displayed text can have an adverse effect on the legibility of a message. However, text colour can also be used to assist in the conveyance of a message. Red, for example, is a colour associated with danger and so is a natural choice for the display of a warning or caution message. It is not a suitable choice for the items of a menu or for help topics. Colour should be used sparingly, since a large number of colours can lead to 'visual overload'. When this occurs, the brain spends more of its effort in decoding the colours, leaving a smaller proportion available to process the information presented. This is especially important if the user is colour blind.

### Use of borders

Borders are an effective method of separating different screen elements. A border indicates a relationship between the various elements within the border, whether they are items stored in the same volume, words in a displayed document or items in a menu. Borders should not be so prominent that they attract the user's attention, but should be used purely for the purpose of separation. Compare the screens in Figure 5.28.

### Justification and alignment

The terms 'justification' and 'alignment' refer to the lining up of the characters at the beginning and end of a line of text. There are four methods used:

- **left** alignment (sometimes called 'left justification'), in which the first character of each line of type aligns vertically, and the characters on the far right create a 'ragged' look
- **right** alignment (sometimes called 'right justification'), in which the last character of each line forms a vertical line, the characters on the left creating a 'jagged' look
- **centre** alignment (sometimes called 'centred text'), in which the lines of text are placed equally on both sides of an imaginary line running vertically down the centre of the screen
- **justified** (also called 'fully justified'), in which the first and last characters of each line of text are lined up with those above and below to form a neat rectangle of text.

## MAIN HEADING

Use the up or down arrow keys to make your choice

|               |              |
|---------------|--------------|
| Menu item 1   | Menu item 7  |
| Menu item 2   | Menu item 8  |
| Menu item 3 ← | Menu item 9  |
| Menu item 4   | Menu item 10 |
| Menu item 5   | Menu item 11 |
| Menu item 6   | Menu item 12 |

For help, press the F1 key

Screen has no borders and related items do not stand out clearly

## MAIN HEADING

Use the up or down arrow keys to make your choice

|               |              |
|---------------|--------------|
| Menu item 1   | Menu item 7  |
| Menu item 2   | Menu item 8  |
| Menu item 3 ← | Menu item 9  |
| Menu item 4   | Menu item 10 |
| Menu item 5   | Menu item 11 |
| Menu item 6   | Menu item 12 |

For help press the F1 key

Screen has light borders and related items are clearly separated from other items

## MAIN HEADING

Use the up or down arrow keys to make your choice

|               |              |
|---------------|--------------|
| Menu item 1   | Menu item 7  |
| Menu item 2   | Menu item 8  |
| Menu item 3 ← | Menu item 9  |
| Menu item 4   | Menu item 10 |
| Menu item 5   | Menu item 11 |
| Menu item 6   | Menu item 12 |

For help press the F1 key

Screen has excessively thick borders which make the items hard to distinguish and the borders are overpowering

**Figure 5.28** Borders are used to separate screen elements. The choice of border can affect the legibility of the elements.

In the presentation of screen information, care must be taken to ensure that the text is clear and legible. For example, a major screen heading is more prominent if centred across the screen rather than to the left or the right. However, centred menu items do not work satisfactorily as the eye has to move horizontally and vertically in order to read the next or previous item. Left alignment is the most appropriate format. Right alignment is rarely used, for the same reason. Justification may cause problems depending on whether the type is proportional or mono-spaced. Using a mono-spaced typestyle gives each character, including spaces, the same horizontal room. The only way that mono-spaced fonts can be justified is if full spaces are added between words, which can lead to distracting 'rivers' of background appearing down the screen, as shown in Figure 5.29.

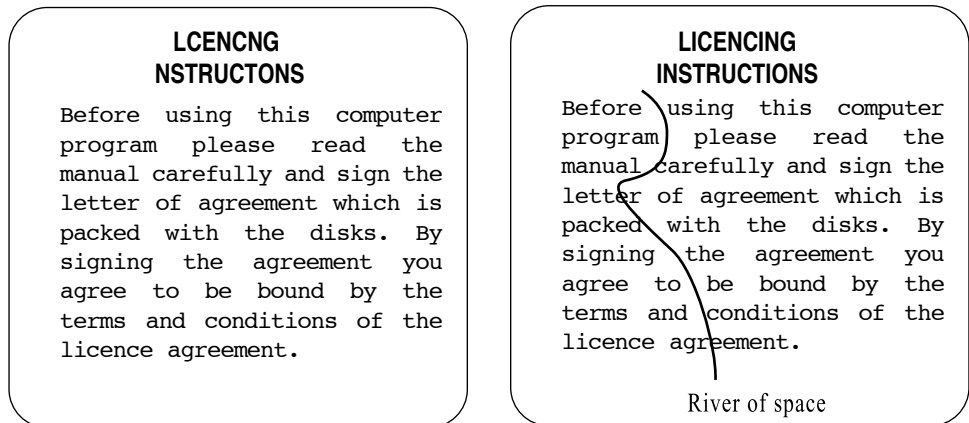


Figure 5.29 Justified mono-spaced text showing 'rivers' of white space.

## Exercise 5.9

- 1 Complete each of these statements with the most appropriate word from the list: alignment, clear, colour, concise, consistency, ergonomically, interface, justification, readability, relationship
  - a Uniform commands throughout a program is an example of design \_\_\_\_\_.
  - b A computer communicates with the user through the user \_\_\_\_\_.
  - c Excessive use of \_\_\_\_\_ in text can overload the user's visual system.
  - d Screen messages should be \_\_\_\_\_ and \_\_\_\_\_.
  - e The spacing of text will affect the \_\_\_\_\_ of a set of instructions.
  - f A border indicates a \_\_\_\_\_ between the elements it encloses.
  - g \_\_\_\_\_ and \_\_\_\_\_ refer to the lining up of text at the beginning and end of a line of text.
  - h One of the concerns of screen design is to provide a(n) \_\_\_\_\_ sound interface.
- 2 Briefly describe the features that constitute good screen design.
- 3 Describe the screen design features which assist with the legibility of a screen. Illustrate your answer with examples.
- 4 Explain why consistency in screen design plays an important part in the construction of a good user interface.

- 5 'Colour can play an important part in providing information to a user, but its overuse can be detrimental to the effective use of a program.' Discuss this statement in terms of your knowledge of screen design principles.
- 6 Compare the two screens in Figure 5.25 in terms of screen design principles. State the good and bad features of each. Redesign the two screens so that they exhibit good design principles including that of consistency.
- 7 Choose a public domain program and comment on the good and bad points with regard to its screen design.
- 8 Design a text-based screen which gives the user a choice of seven different instructions at the bottom of the screen.
- 9 Use the form view of a database application such as Claris Works or Microsoft Works to create an input screen for an address book.
- 10 Visit an ATM at a bank or building society and take note of the screen design. Describe the features that exhibit good design and those you think can be improved. Design a screen which welcomes a customer to an ATM.

## Screen elements

Screen design is one component of the process of communication between a program and its user; the other involves the choice of appropriate elements to convey the information. Early programmers had little choice in the method of presentation as most screens were text-based, but we are more fortunate. With the advent of cheaper and faster computers, it became possible for graphical methods to be used for display, and so the **graphical user interface (GUI)** was born. The GUI provided two opportunities for the screen designer.

- to make communication between the computer and user more *intuitive*
- to provide information in a *graphical* form.

The remainder of this section describes the major types of screen elements available to the programmer. The choice of programming language to implement a problem may dictate whether or not some of these elements can be used.

## Choice elements

### Radio buttons

Radio buttons provide a method of choosing one of several options, at the same time deselecting the previous choice (see Figure 5.30). They take their name from the imagined action, which is similar to selecting a station on a push-button

|                                                                                                                                                                           |                                                                                                                                                                           |                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <input checked="" type="radio"/> <b>Print to Parallel Printer</b><br><input type="radio"/> <b>Print to Serial Printer</b><br><input type="radio"/> <b>Print to Screen</b> | <input type="radio"/> <b>Print to Parallel Printer</b><br><input checked="" type="radio"/> <b>Print to Serial Printer</b><br><input type="radio"/> <b>Print to Screen</b> | <input type="radio"/> <b>Print to Parallel Printer</b><br><input type="radio"/> <b>Print to Serial Printer</b><br><input checked="" type="radio"/> <b>Print to Screen</b> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

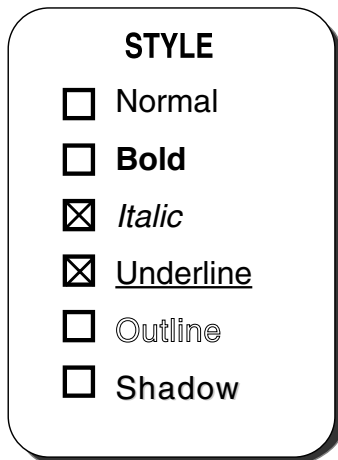
First choice—parallel printer

Second choice—serial printer

Third choice—print screen

**Figure 5.30** A sequence of three radio buttons—only one of them is 'active' at a particular instant. Pressing the 'Print to Screen' button deselects the 'Print to Parallel Printer' button.

radio. They are suitable for those menus which have several options, with only one of them being active at any particular time. Radio buttons are usually used in conjunction with a mouse or other pointing device, and the click of the mouse button further adds to the perception that a button has been pressed.



**Figure 5.31** Two or more check boxes may be marked at a particular time.

### Choice boxes

Radio buttons are used where only one choice is able to be used within a selection. However, there are cases in which two or more options can be chosen from a selection. Choice boxes or check boxes allow the user to mark a number of selections before proceeding. **Check boxes** are used to turn on or off a particular selection; this process is known as **toggleing**. Check boxes usually indicate that an action has been chosen by the user placing a cross inside a small square beside the menu item. When a checked box is again selected (usually by clicking the mouse button), the cross disappears, indicating that the choice has been deselected. A common use of check boxes is within the formatting commands for text, where characters may be given different characteristics such as boldness, italics, underlining or shadow.

### Dialogue boxes

A dialogue box, sometimes called a **choice box**, will usually occupy less than a full screen. Its purpose is to provide the user with a choice of options before some action is to take place. A dialogue box may consist of a simple message and two or more choices, as shown in Figure 5.32, or it may provide access to a number of different choices by means of choice boxes or radio buttons. Dialogue boxes will usually appear on the screen with the most often used choice(s) as default value(s), so that acceptance of the most commonly used items can be effected



**Figure 5.32** A dialogue box offering a default choice (to cancel).

simply. A default choice can also be used in a dialogue box as a 'safety net' to allow the user to exit from a potentially dangerous situation. The example in Figure 5.32 illustrates the use of a 'safety net' type of default choice, the Cancel option being a way out if the dialogue box has been accidentally activated.

A dialogue box may also be used to provide the user with further information or help. While on the screen, a dialogue box will usually be the only active area of the screen, thus drawing the user's attention to the messages that it carries.

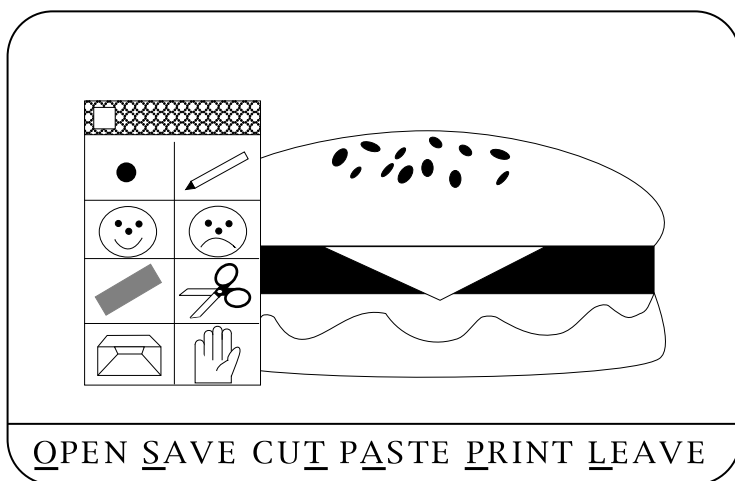
### Navigational elements

An important aspect of program design is the manner in which the user moves from choice to choice. This process is similar to choosing a path or route and is therefore called **navigation**.



Several differing navigation methods are available to the programmer. Those that are chosen will depend on the type of application, the type of user being targeted, the type of user interface and the options available within the particular programming environment. The methods that are most commonly used include the use of screen hot spots, icons, menus, special key combinations, palettes and toolbars.

- Screen **hot spots** such as buttons have various actions associated with them.
- **Icons** (small pictures) represent the various program modules.
- **Menus** offer a number of choices, the choice being selected by the use of a keyboard, a pointing device or another input mechanism. A menu may be presented as a line of options on the screen, in which case it is called a **menu bar**. Menu items can also be presented as 'pop-up' or 'scroll-down' menus.
- **Special key combinations** are very popular in the 'command line interface' in which all communication with the computer is accomplished by keystrokes. The use of key combinations is often accompanied by the provision of a keyboard template which shows the user all of the appropriate key combinations.
- **Palettes** present the navigational options in the form of a grid of choices, which are often shown as icons. Palettes may be **floating**, which enables the user to put them in places on the screen that do not interfere with the task being performed. This type of navigational aid is often used in graphic-processing applications, where a majority of the screen area is needed for the manipulation of data.
- **Toolbars** present their options as a line of icons. Toolbars are usually placed along one of the edges of the screen.



**Figure 5.33** A screen showing a 'floating palette' and a menu bar.

### Menus and menu bars

In all but the simplest programs, the user is required to make choices. One of the most popular ways of displaying those choices is to use either a menu or a menu bar. Creating a usable menu or menu bar is not just a matter of listing the choices and thinking of the way in which they are to be presented. Foremost in the mind of the screen designer is the need to quickly and clearly present the available options and provide the user with an easy method of making a choice.

Items presented in either a menu or a menu bar should be clearly defined and consistent in placement, method of choice and meaning. As much as possible, the same, or similar, menu items should appear in the same places on the screen and require the same actions for choice. For example, it is common practice among a number of programmers to use the ESCAPE key to either cancel an action or return to the previous screen. If this method of cancellation is carried through the entire program, it builds confidence in the user. However if, for some reason, a programmer decides to use the ESCAPE key to cancel on one menu, and uses the key combination CONTROL-X on the next, a user will have to make a conscious effort to match the screen with the method of cancellation. The second example will slow down the operator and may cause anxiety and loss of confidence in the use of both the program and the computer.

### Icons

An icon is a pictorial representation of an object or action. A properly designed icon can provide an easily recognised portrayal of that item. The design of a suitable icon is not easy. Several points have to be taken into consideration:

- The icon should be easily recognisable as *representing the action or object*. For example, an icon in the shape of a pair of scissors is almost certainly going to represent the process of 'cut'. The scissors icon could not reasonably be interpreted as 'delete from disk'.
- The icon should be *simple*. Fancy colours and graphics provide the user's brain with more information than is necessary, all of which has to be processed before a decision on the icon's meaning can be made.
- The icons in a screen group should, as far as possible, be easily *distinguishable* from the others in the same group.
- Icons should be *not too small* as the user needs to be able to place the cursor on them.

Icons can be used on the screen by placing individual icons in an appropriate position on the screen, or by incorporating them in groups, using screen elements such as toolbars, menu bars, rulers, palettes and dialogue boxes. They can also appear as part of a dialogue box to bring the user's attention to some operation or consequence. For example, a 'stop sign' icon is often used to convey the message that a process has been halted prematurely, or that the program has stopped for the user to make a crucial decision such as whether or not to continue with the deletion of a file. Figure 5.33 illustrates the use of icons in a floating palette. Figure 5.34 illustrates the use of icons in a toolbar.



**Figure 5.34** A toolbar illustrating the use of icons on buttons to represent various menu choices. Some of the icons have obvious meanings, but others do not.

### Windows

One of the most widely used screen display elements is the window. **Windows** are areas of the screen that are enclosed by a border. The use of windows allows several differing items to be simultaneously displayed, enabling the user to quickly change from one to the other. The use of windows allows the user to imagine the computer screen as a desktop on which there are various items; one

of these items will be 'on top' of the pile and therefore able to be changed. Windows are most common in a graphical user interface. They can be employed in a text-based screen display, but with difficulty.

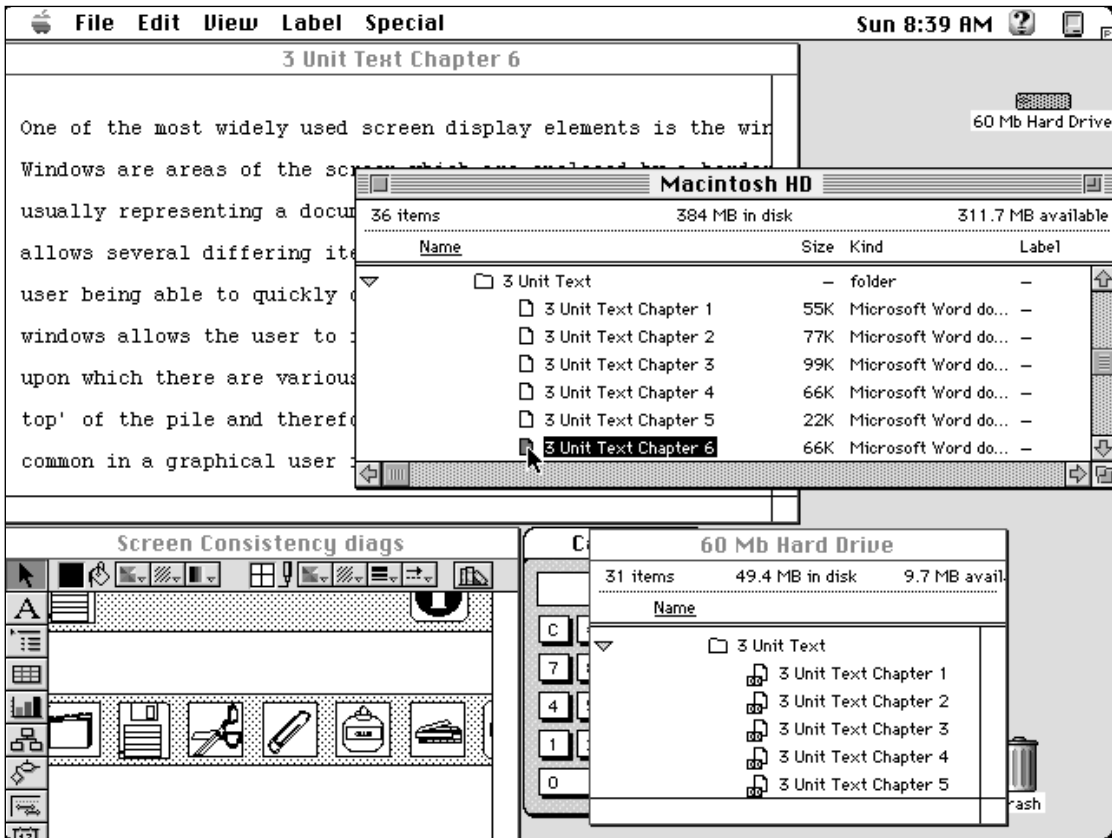


Figure 5.35 A number of windows visible on a screen.

### Prompts

Prompts are short messages to the user giving information about what actions can be carried out. Screen prompts are an important communication between the program and the user and as such should be clear and unambiguous. Good prompts should be non-threatening, should not talk down to the user and should avoid humour. The placement of prompts is also significant, as a prompt's positioning on the screen will often signal its importance. For example, a prompt in the centre of the screen will be very noticeable and is therefore most suitable for a warning message, whereas a prompt placed at the bottom of the screen is less noticeable and is often used as a help item.

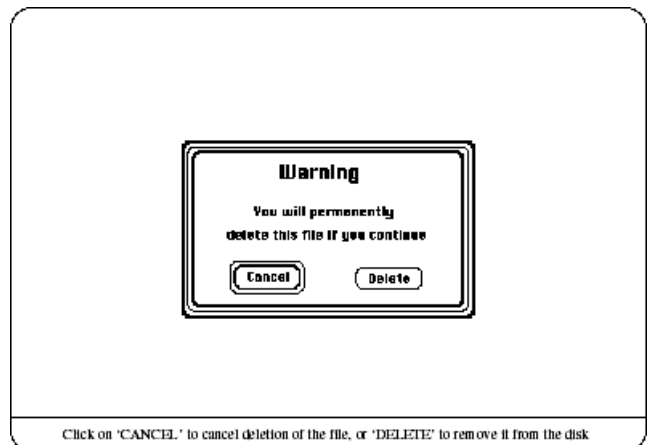


Figure 5.36 The placement of a prompt will often signal its importance.

## Graphics, pictures and text

During the twentieth century the printing industry has moved from a largely text-based system to one which incorporates graphics and pictures. A similar transformation has occurred with computer technology, and many printing techniques can be utilised by a programmer when designing a screen.

Graphics, pictures and text form the visual image which the user interprets. Often a message can be conveyed better through the use of a visual image such as a picture or a graphic, although on other occasions text is a better choice. There are times when two or more of these elements have to be combined and this is where a designer needs to exercise caution. The placement of these elements in relation to each other is important, as poorly placed graphics, pictures or text can result in the element(s) being ignored by the user.

When determining the placement of these items, the natural reading scan has to be taken into consideration. The reader of English scans the page from left to right and from top to bottom. Therefore elements in the top-right and bottom-left corners of the page or screen will have less impact. Similarly, when users first look at a screen, their attention is drawn to its centre, so elements in the centre of the screen will be more noticeable than those at the edges. The top of the screen will take precedence over the bottom, as the user's eyes are most likely to focus on the top first.

## Exercise 5.10

- 1 Complete each of the following statements with the most appropriate word from the list:  
check boxes, command line, default, GUI, icon, navigation, palette, prompts, radio buttons, window
  - a A user interface which uses pictorial methods to present information is known as a \_\_\_\_\_.
  - b An option which is most likely to be chosen by the user can be made a \_\_\_\_\_ value in a dialogue box.
  - c A text-based system is often known as a \_\_\_\_\_ interface.
  - d A ~~is an area~~ of the screen surrounded by a border.
  - e A graphical method of showing choice, known as a \_\_\_\_\_, is used as an alternative to text.
  - f \_\_\_\_\_ can be used to present a number of choices to the user when only one is able to be used at a time.
  - g Short messages called \_\_\_\_\_ are presented to the user to provide information about a choice.
  - h When two or more choices from a selection can be active at the same time, a screen designer may choose to represent them with \_\_\_\_\_.
  - i A number of related icons can be grouped together as a \_\_\_\_\_ on the screen.
  - j Movement from choice to choice within a program is known as \_\_\_\_\_.
- 2 Briefly describe, with examples, four different screen elements.
- 3 Explain how the choice of screen elements can assist the user.
- 4 Describe the screen elements that are available to the designer of a text-based screen.

- 5 Describe the advantages that a graphics-based screen has for the designer.
  - 6 Design an icon which is suitable for representing an 'undo' which reverses the previous action.
  - 7 Design a dialogue box which allows the user of a database to move between the 'add a record', 'delete a record' and 'update a record' modules.
  - 8 Design a screen for a program which has been created for a small child to practise the alphabet. The child has a letter displayed together with a picture of an object. Choices are to go on to the next letter, go to the previous letter, hear the letter name spoken, hear the name of the object spoken or leave the program.
  - 9 Examine and comment on the elements used in the design of a public domain software application.
  - 10 Compare the use of screen elements in the public domain application examined in question 9 with those of a commercial software application.
- 

## Documentation

### Types of documentation

Documentation can be categorised as external or internal. **External documentation** is needed for the development and use of software. **Internal documentation** forms part of the actual programming.

Several forms of documentation are required during the software development cycle. This documentation is not discarded at the implementation stage but is used later by system developers to ensure that the software functions correctly.

**Software specifications** define the nature of the problem and indicate the nature of the inputs, outputs and restrictions. **Test data** is also created at this stage. As part of the specifications, documents such as a program specification report, dataflow diagrams, a data dictionary, and input and output specifications are created.

The **algorithm description** forms the basis for program design and, regardless of the description method chosen, will give an unambiguous description of the processes which have to be coded. Common methods used for describing algorithms are the flowchart and pseudocode (the two approved methods of algorithm description for this course), structured English and Nassi-Schneiderman diagrams. Once coded in a particular language, the algorithm description becomes the program's **source code**. (Source code will be documented by intrinsic documentation such as the choice of appropriate identifier names and by other internal documentation such as comments or remarks.)

Implementation of the software requires the following forms of documentation for those who install and use the software:

- an installation guide to inform the installer of all steps necessary to successfully install the software on a computer system; this guide will also provide information on the minimum hardware requirements as well as on problems that may be encountered during this phase
- a user's manual (the main reference) to provide information on the tasks that the system can perform and ways in which the user can overcome any operating problems

- supporting documentation such as trouble-shooting guides and various forms of online documentation.

Each of these documents is produced at a particular stage of the development cycle. In order to achieve a successful solution, these documents are sorted into various sets, each set being appropriate to a particular group (or groups) of personnel. One set is created for the **development team** to use during any modifications; one set is designed for the **end user** and one for the **system administrator**. In this section we examine the documentation that is appropriate for each of these groups and discuss the standards that should apply to these documents.

During the development cycle a large amount of external documentation is produced. Some of these documents will form the basis of the product documentation; the remainder, known as process documentation, will become largely outdated.

**Process documentation** is documentation produced by the system development cycle. It will have served its development purpose by the time the cycle reaches the implementation stage, but should be retained to assist in system maintenance and later development.

**Product documentation** falls into two categories: that which becomes manuals and guides for system maintainers and developers, and that which forms user's guides. User documentation incorporates documents that describe the purposes of the system and how end users can use the software. Other user documentation is provided for system administrators whose task it is to keep the system running.

As can be seen from the above descriptions, final documentation of the product is not created as an afterthought but forms an integral part of each stage of the development cycle. It is extremely important that the documentation is kept current, by being updated each time changes are made.

### **Documentation for developers**

Documentation required by developers will consist mostly of process documentation, documenting the design process and the system.

The **design process** is an important aspect of documentation. The original design plan, together with subsequent modifications, can be useful in planning further design activities. Such a design history can be extremely useful in later projects. Design processes learnt during the development of one system can lead to greater success when new projects are attempted. Documents falling into this category include test schedules, memos, working papers and reports.

The purpose of **system documentation** is to provide a detailed description of the system and to provide information that will assist with the maintenance of the system. System documentation consists of many of the documents produced at various stages during the development cycle: the system requirements, system descriptions, algorithm descriptions, program source code, plus a system maintenance guide. They include dataflow diagrams, data dictionaries, output specifications reports and data files specifications.

### **Documentation for users**

Documentation required by users will consist of a number of documents: a functional description, an introductory manual, a system reference manual, a system installation guide and a system administrator's manual. In addition, support documents such as trouble-shooting guides, reference cards and online help are often provided.

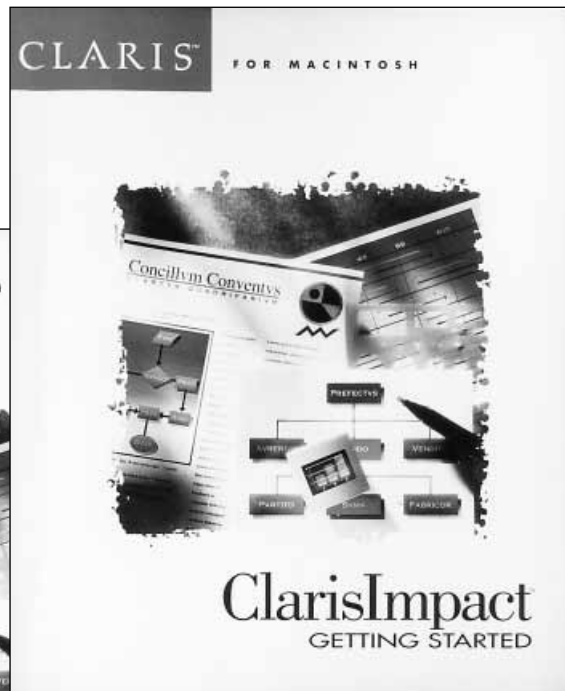
## Paper based documentation

A **functional description** contains a brief outline of the system requirements and the purpose of the system. An **introductory manual** describes the manner in which the system is started and the use of common system functions. The introductory manual should contain a series of tutorials that illustrate these common functions as well as methods of recovering from common errors. It is important that the introductory manual is written in an informal and non-technical manner so that a beginner will find it easy to follow (see Example 21).

### Example 21

ClarisImpact is a business graphics package which enables the user to produce multi-page drawings, reports and presentations. The Getting Started manual (Figure 5.37) is divided into six chapters. The first is an installation guide, the second provides a general overview of the application, and the other four deal with the major features of the program. Each of the operational chapters uses a series of tutorials to take a new user through the basic functions of the program. At the end of each chapter is a reference table which directs the user to the appropriate chapter in the User's Guide (Figure 5.38) where more information can be obtained.

**Figure 5.37** ClarisImpact Getting Started manual.



**Figure 5.38** The ClarisImpact User's Guide.

A more comprehensive **system reference manual** (or **user's manual**) will accompany the introductory manual. The system reference manual will contain a complete listing of all the functions of the system, arranged in a logical manner (for example listing the functions alphabetically or by family). Each of the functions should be described in detail, with a description of the inputs the function requires, the outputs it produces and any special features it may have. As well as these basic descriptions, the manual may contain samples of screen displays that the user may encounter while using the functions, and a graphical representation of the way in which the functions relate to each other. A second purpose of the manual is to provide a complete description of all known operational errors and how to recover from them.

The system reference manual should be written in a more formal style than the introductory manual and provide a comprehensive description, as its purpose is for reference. Readability is not as important as accuracy and clarity. Language used in this manual will tend to be more technical than in the introductory manual, although the use of technical terms should be kept to a minimum.

Like the ClarisImpact Getting Started manual, the User's Guide is arranged so that it incorporates tutorials that explain the functioning of each available option. The chapters are arranged according to tasks the application can perform; for example, there are separate chapters on making data charts, making timelines, making presentations and so on. Each of these chapters describes the program functions which are required by the task, illustrating their use by means of tutorial examples. The manual also provides a comprehensive alphabetical index.

The **installation guide** is usually aimed at the system administrator. It contains details of the minimum hardware and software support required for the installation as well as a description of the computer-readable media containing the installation files. A further purpose of the guide is to provide details of the permanent files that will be installed as part of the system.

How to start the program and how to configure it to meet a particular user's needs will also be described. The installation guide should be written with these aims in mind. Language should be non-technical and concise, and anticipated problems should be described in detail together with their solutions. Installation and configuration steps will often be presented in point or step form so that the installer can check off each step as it is completed.

A **system administrator's manual** is designed to provide the system administrator with a comprehensive description of the program's interaction with other elements of the system and other systems to which it is connected. It will document all messages created when the system interacts with other systems and how to respond to these messages. Language used in this manual will tend to be more technical than in the other documentation; however, technical terms should not be used unless absolutely necessary.

**Trouble-shooting guides** provide a detailed description of the solution to operating problems. The guides expand on the error messages displayed on the screen. Some guides are aimed at the user and contain common operational errors, and others contain system errors and are aimed at the system administrator.

**Reference cards**, which contain a brief list of common system functions and how to use them, can provide support to both experienced and first-time users. A reference card is usually one sheet containing a minimal description of common functions. It is designed for ease of use, its purpose being to provide information without reference to the main set of manuals. Colour coding of functions and key combinations will provide the user with visual clues to help in the use of the card.



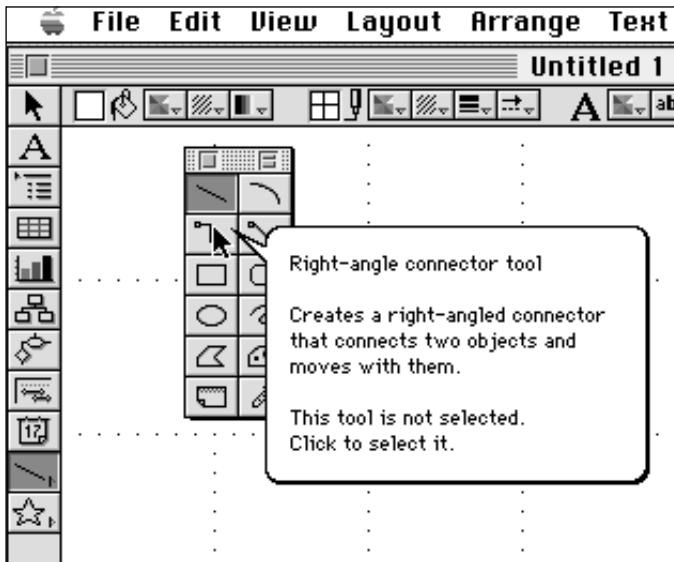


Figure 5.39 A part of the ClarisImpact Quick Reference Chart.

The Quick Reference Chart supplied with ClarisImpact (see Figure 5.39) provides a summary of the program's shortcuts and functions available in each of the modules. The card has a picture of each of the icons used in the program, with the name of the function it represents beside it.

### Online help

Online help is becoming a more common method of providing the user with assistance, as a reduction in the cost and an increase in the power of computer hardware have given programmers the opportunity to use the computer to provide some of the documentation required. This form of documentation has a great advantage over paper-based manuals since it is available to the user without leaving the computer. This is especially important for network users who may be in different locations. Online help can also be linked to the menu items, for example providing a description of the function of a button by means of a pointer. Several methods of providing online documentation have been devised; the most commonly used are balloons, user instructions and tutorial assistants.

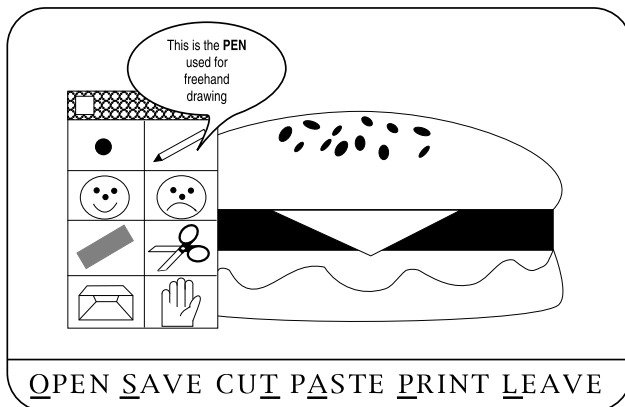


**Figure 5.40** Balloons in ClarisImpact give users information about a screen element that has been selected by the pointer.

Balloons contain a description of the menu or screen item that is pointed to. Balloon text is usually associated with a graphical user environment such as Windows and the Macintosh operating system. When enabled, balloon text will describe the action of a display element whenever that element is pointed to by the screen pointer (see Figure 5.40).

Help screens describe each function on-screen. A particular benefit of the help screen is that it can make use of a computer's ability to rapidly search through a database of items for a wanted function.

ClarisImpact provides two forms of online help: balloons and a topic-based help program. Balloon help gives the user information about the various screen elements (see Figure 5.40). The topic-based help program describes the steps taken to perform various tasks. Both forms of assistance are available from the menu bar.



**Figure 5.41** User instructions are a useful online aid. They are often placed at the bottom of the screen.

The placement of a set of **user instructions** on the screen is the easiest of the screen aids to implement, as it only requires the addition of a line or two of output text. However, it is often difficult to phrase the instructions so that they are clear to the user. The instructions should be placed in such a position that they leave a majority of the working screen for the application display. The most common positioning of user instructions is at the bottom of the screen; this is especially true in a text-based display (see Figure 5.41).

A documentation method which is gaining popularity is the use of a **tutorial-assistant** which helps the user complete a task. This kind of help presents options to the user at various stages of the process, assisting with the steps. These on-line assistants have been designed to provide the user with the necessary information to complete a task, often offering the most common choice as a default.

## Exercise 5.11

- Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.  
Documentaton avlae on the compuer is knon s \_\_\_\_\_. This type of help has become more opular as computers have become \_\_\_\_\_ and \_\_\_\_\_. There are several different types of \_\_\_\_\_ help; \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_ are popular. \_\_\_\_\_ places text in little areas of the screen when thepoiner is at mnu item. are usually placed at the same point on all screens and tell the user what needs to be done. \_\_\_\_\_ are designed to give the user experience in performing a task.
- Name which of the online help methods is best suited for each of the following tasks. Explain your choice.
  - describing how to create a birthday card
  - explaining the meaning of an icon
  - showing how to mail-merge a list
  - explaining how to deal with the choices of a dialogue box
  - showing which item on a toolbar will format a page with three columns
- Design balloon text for each of the following items:
  - a button which erases a disk
  - a menu for a word processor which contains all formatting commands
  - a default button on a dialogue box
- Create a tutorial for your word processor which shows a novice how to set up a letter. Your tutorial should contain a sample file and word-processed instructions. Save your tutorial on disk in a directory (folder) named WORDTUT.
- Design a single line of instruction which:
  - tells the user to press the ENTER key after data entry
  - warns the user that a file is about to be permanently changed
  - informs the user that the only keys to have effect are the A, B, C and D keys.

### Internal documentation

Internal documentation is contained in the programs. It assists programmers in modifying the program for maintenance and further development.

The **program listing** is probably the most obvious documentation associated with a computer-based solution to a problem. However, the manner of presentation of a program can affect both its legibility and its ability to be followed and/or modified. For example, compare the legibility of the following two samples of identical Pascal code. The first sample has not been formatted to any standards; the second has been coded with one instruction per line and indentation has been used to show the statement levels.

The second sample program is easier to follow not only because of the placement of single instructions as separate entries and the use of indentation to show the logical blocks such as loops and decisions, but also because the use of upper-case letters for the Pascal reserved words together with the separation, wherever possible, of the comments allows the structure of the program to be immediately visible. Thus any modification of the code can be accomplished efficiently.

## Example 22

```
program menu_test (input, output); var choice: char;
procedure Add_record; begin writeln('Add record stub') end;
procedure Delete_record; begin writeln('Delete record stub') end;
procedure Sort_records; begin writeln('Sort records stub') end;
procedure Invalid_input; begin writeln('Invalid choice stub') end;
begin writeln('Please press the letter key corresponding to your choice');
writeln('A to ADD a record'); writeln('D to DELETE a record');
writeln('S to SORT the records'); writeln('E to END this session');
readln(choice); while (choice <> 'E') and (choice <> 'e')
do begin if choice in ['A', 'a', 'D', 'd', 'S', 's'] then
 {this statement ensures that other characters are excluded .. otherwise}
 case choice of 'A', 'a': Add_record; 'D', 'd': Delete_record;
 'S', 's': Sort_records end {end of case statement} else
 Invalid_input;
writeln('Please press the letter key corresponding to your choice');
writeln('A to ADD a record'); writeln('D to DELETE a record');
writeln('S to SORT the records'); writeln('E to END this session *');
readln(choice); end end.
```

The programmer has a responsibility, when creating program code, to ensure that it can be easily modified and updated. One of the easiest ways of accomplishing this task is to employ internal documentation in the form of comments (or remarks) and meaningful identifier names.

Most of the languages from the second generation and above allow the programmer to insert text which is able to be ignored by the compiler or interpreter. This facility has been included in the language to allow a programmer to make notes (**comments** or **remarks**) inside the program code. The purpose of this type of internal documentation is to provide a means for the programmer to state the purpose of a section of code for later reference, for example in the debugging or maintenance stages of the code. Internal documentation is also very important if a team of programmers is working on the one problem, as it lets all team members know the processes that are being carried out in other modules.

The careful choice of identifiers can assist in the processes of development and maintenance. The use of meaningful names for variables and subprograms is known as **intrinsic documentation** (that is, the documentation is built into the code). For example, it is much easier to determine what is meant by the line of code `balance := balance-withdrawal` than by a statement such as `x := x-y` when written in a program to process withdrawals from bank accounts. The second statement will, with the same data as the first, produce the same result; however, its meaning may be lost to all but the programmer, and so maintenance could be a problem.

The use of procedures (subprograms) to carry out repeated tasks and the use of identifiers to represent constants in preference to using the value also assist with maintenance. Declaration of constant identifiers at the beginning of a program may seem to be a waste of time, but if it becomes necessary to change

## Example 23

```
PROGRAM menu_test (input, output);
VAR
 choice : char;
PROCEDURE Add_record;
 BEGIN
 writeln('Add record stub')
 END;
PROCEDURE Delete_record;
 BEGIN
 writeln('Delete record stub')
 END;
PROCEDURE Sort_records;
 BEGIN
 writeln('Sort records stub')
 END;
PROCEDURE Invalid_input;
 BEGIN
 writeln('Invalid choice stub')
 END;
BEGIN
 writeln('Please press the letter key corresponding to your choice');
 writeln('A to ADD a record');
 writeln('D to DELETE a record');
 writeln('S to SORT the records');
 writeln('E to END this session');
 readln(choice);
 WHILE (choice <> 'E') AND (choice <> 'e') DO
 BEGIN
 IF choice IN ['A','a','D','d','S','s'] {this statement ensures
 that other characters are
 excluded; otherwise...}

 THEN CASE choice OF
 'A','a' : Add_record;
 'D','d' : Delete_record;
 'S','s' : Sort_records
 END {end of case statement}
 ELSE Invalid_input;
 writeln('Please press the letter key corresponding to your choice');
 writeln('A to ADD a record');
 writeln('D to DELETE a record');
 writeln('S to SORT the records');
 writeln('E to END this session');
 readln(choice);
 END {end of while statement}
END.
```

the value of one of the constants after the program has been written, one occurrence only needs to be changed instead of a number throughout the program. The use of clearly defined subprograms within the code further assists a maintenance programmer in updating the program.

## Exercise 5.12

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.  
A programmer will produce two types of documentation, \_\_\_\_\_ and \_\_\_\_\_ documentation. \_\_\_\_\_ documentation is built into the code and takes the form of comments and \_\_\_\_\_ documentation. \_\_\_\_\_ includes manuals, \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_. The \_\_\_\_\_ documentation is important for maintenance as it describes the \_\_\_\_\_ that are being carried out in the program.
- 2 Explain the differences between internal and external documentation. Give an example of each type of documentation.
- 3 Why would a program's code listing would be useful after its implementation? Give reasons.
- 4 Explain the role that formatting plays in the understanding of a program listing.
- 5 Reformat the following program so that its structure becomes clear:  

```
on askQuestion ask 'Please enter X, Y or Z' if it is not empty then
if it is 'X' then do Xmenuitem else if it is 'Y' then do Ymenuitem
else if it is 'Z' do Zmenuitem else answer 'Please enter only X,Y
or Z' do askQuestion endif end askQuestion
```
- 6 Code a program for the algorithm described in Figure 4.5 using an approved programming language. In coding, pay attention to the formatting to ensure that your program is easily read.
- 7 Define the term 'intrinsic documentation' when applied to a program listing. What are the advantages of using intrinsic documentation?
- 8 Most high-level languages allow the programmer to insert comments or remarks which are ignored by the compiler. What is the purpose of these statements?

## Team Activity

Your team is given the task of designing the school's interactive guide. You are to discuss the screen design elements that are common to all screens of the guide. Once you have decided on the guidelines, publish them using a word processor. Each member of the

team is then to use these guidelines in designing one screen for the guide, using a graphics program. When all screens are completed, team members should examine all screens from the group and report on how well the team guidelines have been followed.

# Review exercises

- In English, a proper noun must start with an uppercase letter followed by at least one lowercase letter. Write a description of this syntax using:
  - BNF
  - EBNF
  - syntax structure chart
- Using Figure 5.10, the TUSIL structure diagrams, determine whether each of the following TUSIL elements is legal. If the element is not legal, state the error and correct the element.
  - Real number .543
  - Identifier hours-long
  - Assignment WEEKS\_WAGE = PAY - PENALTY
  - Integer 6543210
  - Real number 6543210
  - AVE RB SUM \* NUMBER
- Identify the probable control structure in each of the following code segments:
  - ```
if(paid) {
    ticket = 'yes'
} else {
    ticket = 'no'
}
```
 - ```
{ticket = 'yes';
ticket = 'no'
}
```
  - ```
case grade of
    'A', 'B' :
        good_letter := true;
    'C' :
        good_letter := false;
    'D', 'E' :
        bad_letter := true;
end;
```
 - ```
while index < 35 do begin
 index := index + 1;
 total := total +
 weight[index]
end;
```
- Explain why some languages require a programmer to declare the type of variables before they are used within a program.
- Describe the different types of error that may occur when a program is compiled and run on a computer. Give one example for each of the errors you have identified.
- Explain how stubs, flags and debugging output statements can be used to help the programmer find errors in the coded program.
- Explain how you would write a subprogram so that it can be added to a library of code. Use an example to illustrate your answer.
- Use a graphics program to design a screen for an interactive compact disk player. The screen should display the details of ten compact disks and allow the user to select one to be played.
- Design a text-based screen to perform the same task as in question 8. Your screen is restricted to 24 lines, each containing 80 characters. Use a word processor for the design. You need to use a mono-spaced font such as Courier for this exercise.
- Use a graphics program to design a suitable floating palette which gives the choice to the user of 10, 12, 14, 16 and 18-point font size within an application. Justify your choice of method.
- Design a trouble-shooting guide that will help a novice computer user find the reason why the computer is not starting up properly.
- Describe the differences between product and process documentation. Explain who uses each type of documentation and the purpose for its use.

# Chapter summary

- Metalanguages are used to describe the syntax of a language. The two metalanguages used in this course are BNF/EBNF (text-based) and syntax structure diagrams (graphical).
- Metalanguage constructions can be used to decide whether or not a statement follows the rules of a language.
- As for algorithm descriptions, there are three basic control structures in a language: sequence, selection and repetition.
- When a program is executed, memory locations have to be allocated for variable storage. Each language has its rules for determining the type of a variable so that this storage can be properly allocated.
- Errors in coded programs will either be syntax errors or execution errors.
- Syntax errors occur when a piece of code breaks one or more of the rules of the language.
- Execution errors occur when the program is being run.
- Stubs are small portions of code which represent yet-to-be-written sections and are used to test the logic of a program before it has been completed. Stubs may also be used to input test data items into a section of code.
- Debugging output statements are used to print out the value or values of variables at particular points in the program so they can be examined. They may also be used to trace the flow of execution through a section of code.
- A flag is usually a Boolean variable whose value is changed from 'false' to 'true' if a section of code has been executed.
- Programmers create and use libraries of already written code to shorten development time. When a module has proved successful it should be added to the library.
- Subprograms and modules should be written in such a manner that they can be easily modified to perform a new but similar task.
- The user interface is the only method of communication between the user and the program, so it should be carefully and thoughtfully designed.
- Consistency in screen design and the use of appropriate messages, text features and screen elements will assist the user in performing tasks as well as improving user confidence.
- Process and product documentation are produced during the development process.
- Process documentation is used to assist with maintenance of the system, and includes the algorithms, test data and code.
- Product documentation is used to assist the user and system administrators with the operation of the system, and includes manuals, tutorials and online documentation.
- Internal documentation is contained in the programming and assists in making modifications. It includes intrinsic documentation and attached comments.



# chapter 6

## → *Checking the software solution*

### Outcomes

- identifies the issues relating to the use of software solutions (P 3.1)
- investigates a structured approach in the design and implementation of a software solution (P 4.2)
- uses and justifies the need for appropriate project management techniques (P 5.1)
- uses and develops documentation to communicate software solutions to others (P 5.2)
- describes the role of personnel involved in software development (P 6.1)
- communicates with appropriate personnel throughout the software development process (P 6.2)
- designs and constructs software solutions with appropriate interfaces (P 6.3)

### Students learn about:

#### Test data

- selecting data for which the expected output is known
- the need for thorough test data
- the selection of appropriate test data, including:
  - data that test all the pathways through the algorithm
  - data that test boundary conditions — upper and lower values and values upon which decisions are based
  - data where the required answer is known
- testing both algorithms and coded solutions with test data, such as:
  - desk checking an algorithm
  - stepping through a coded solution line by line

### Evaluation of design

- comparing different solutions to the same problem
  - different interpretations of the design specifications
  - the advantages and disadvantages of different approaches to reaching the solution
- peer checking
- structured walk-through
- desk checking

### Evaluation of implemented solutions

- checking the solution to see if it meets the original design specifications
- user feedback
- social and ethical perspectives

## Students learn to:

- determine the expected result given the test data
- create a set of appropriate test data and use it to verify the logic in a solution
- use test data on algorithms and coded solutions
- communicate solutions to others
- critically evaluate their work and that of their peers and share good aspects of their solutions using elegant aspects of other students' solutions

## Personal Profile—Donald Knuth (1938– )

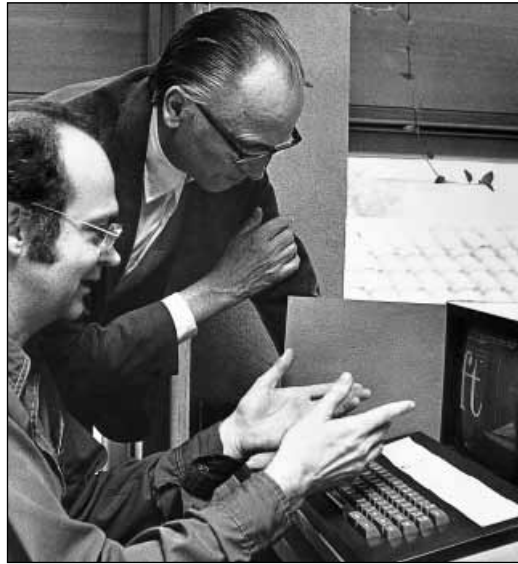
Donald Knuth was born in Milwaukee, Wisconsin on 10 January 1938. He demonstrated a number of varying skills while still at school. In eighth grade he won a competition for making the most words from a candy bar name, easily winning by forming over 4500 words. This win demonstrated his ability to recognise and manipulate patterns. Although interested in mathematics at school, young Donald spent more of his time playing and composing music. He was going to major in music once he went to college.

In 1956 Donald was offered a physics scholarship to Case Institute of Technology. In his first year at Case he taught himself to program the university's IBM 650 computer. This was no mean achievement, since at that period programming was performed in assembly language. He excelled in programming, writing programs with applications as varied as factorising numbers into primes, playing tic tac toe and rating the players of the school's basketball team. He graduated with both a BS and an MS in 1960, the Master's degree being awarded in recognition of his exceptional achievements. In 1963 he was awarded a doctorate from California Institute of Technology.

While still a graduate student, Knuth started writing compilers for various computers. He served as a consultant to the Burroughs Corporation from 1960 to 1968. He was approached by the publisher Addison Wesley to write a book about compilers. By 1966 he had written 3000 pages of draft, discovering a general method for determining the rules needed for a language. The publisher decided to expand the project to a seven-volume survey of programming. The *Art of Computer Programming* is a standard reference on the subject, at the moment consisting of three volumes, published in 1968, 1969 and 1973. Knuth has now retired academically to concentrate on finishing the book.

Knuth started working on a digital typography project in 1976. This resulted in the TeX document preparation system and the METAFONT system for type design. As a result of this work, the WEB and CWEB languages for structured documentation were also produced and the methodology was called Literate Programming.

Donald Knuth was Professor of Computer Science at Stanford University from 1968 to 1993 when he was appointed Professor Emeritus of the Art of Computer Programming. He has written a large number of research papers and a number of other books, including a novel. He is also an accomplished musician and composer who has designed his own pipe organ. He still lives at Stanford University.



## Test data

An item being manufactured will, at various stages of its construction, undergo testing to ensure that it will effectively perform the task for which it was designed. A computer program undergoes the same process. At each stage in its development the program is tested to determine whether it will solve the problem for which it was designed.

A program and its components cannot be tested unless some form of input is given. Similarly, the outputs from these inputs need to be known before the testing process is begun. A set of data known as **test data** has to be formulated to use for the testing process. Documentation associated with this data will also be needed in order to explain why certain elements have been chosen and what the expected outcomes of the processing will be.

### Requirements of test data

For a set of test data to be an effective assistant in the process of testing and debugging, it is necessary to carefully formulate and document each data item.

The foremost objective in testing an algorithm or program is to determine whether the original problem is solved by it. Thus the first step in test data design is to choose data which relates to differing cases, as described in the original problem. This process involves examining all decisions to be made in the execution of the program and ensuring that test data is created which causes program flow to be channelled along each of the paths within the algorithm or program. When data enters a decision process, one or more values are used for comparison; these values are known as **boundary values**. Test data needs to be constructed in order to test not only what happens on both sides of the boundary, but also the value itself.

A good set of test data should achieve the following three aims:

- to test all parts of the program
- to test each of the paths that can be taken during execution
- to test on each side of any boundary value as well as the value itself.

### Example 1

#### Problem: Typist rating

A program is to be written to grade a typist's speed. If the speed in words per minute (wpm) is below 25, the typist is to be graded as 'Unsatisfactory—needs training'; if the speed is 25 wpm up to 55 wpm inclusive, the typist is to be graded as 'Satisfactory'; if the speed is over 55 wpm, the typist is to be graded as 'Exceptional—receives a 10% bonus'.

#### Test data considerations

The boundary values in this problem are at 25 and 55 wpm. Test data should be chosen in the following ranges: less than 25 wpm, exactly 25 wpm, between 25 and 55 wpm, exactly 55 wpm and over 55 wpm. The other consideration for this set of test data is to ensure that negative and zero typing speeds are rejected as being out of range. (The inclusion of one of these data items can then be used to check **data validation** procedures within the processing.) Typing speeds in excess of 600 wpm (10 words per second) are unreasonable, so such a speed should be included to check for reasonableness. Sample test data are shown in Table 6.1.

*continued next page*

| Typing speed (wpm) | Expected output                  |
|--------------------|----------------------------------|
| 24                 | Unsatisfactory—needs tanning     |
| 25                 | Satisfactory                     |
| 26                 | Satisfactory                     |
| 55                 | Satisfactory                     |
| 56                 | Exceptional—receives a 10% bonus |
| -1                 | Invalid input—out of range       |
| 700                | Invalid input—unreasonable value |

**Table 6.1** Sample data for the typist-rating problem.

## Example 2

In this example the boundary values are not expressed directly within the problem.

### Problem: Letter postage

A program is to be written to calculate the cost of posting a letter according to the following rule: Letters with weights up to and including 150 g cost 50 cents. Each 100 g, or part, above 150 g is charged at a rate of 25 cents up to a maximum letter weight of 500 g. All letters over 500 g are rejected as parcels.

### Test data considerations

The boundary values in this problem are not clearly stated. We need to determine what those values are before choosing test data for this application. The formula states that the first 150 g are charged at 50 cents. The test data should therefore cover values which are under 150 g and exactly on 150 g. Since the problem specification also implies other boundary values, these should be taken into consideration when choosing other test values. This type of data is especially important if the algorithm ends up using a number of binary selections, or a multiple selection, as all branches need to be checked. Keeping this in mind, values in the following ranges also need to be included: over 150 g and under 250 g, over 250 g and under 350 g, over 350 g and under 450 g, over 450 g and under 500 g and over 500 g. The boundary values of 250 g, 350 g, 450 g and 500 g should also be part of the set. Finally, a negative letter weight is included to check the data validation section of the algorithm.

| Letter weight (g) | Expected output                |
|-------------------|--------------------------------|
| -100              | Invalid input<br>—out of range |
| 145               | 50 cents                       |
| 150               | 50 cents                       |
| 155               | 75 cents                       |
| 250               | 75 cents                       |
| 255               | \$.00                          |
| 350               | \$.00                          |
| 355               | \$.25                          |
| 450               | \$1.25                         |
| 455               | \$1.50                         |
| 500               | \$1.50                         |
| 505               | Parcel rates apply             |

**Table 6.2** Sample data for the postage problem.

## Example 3

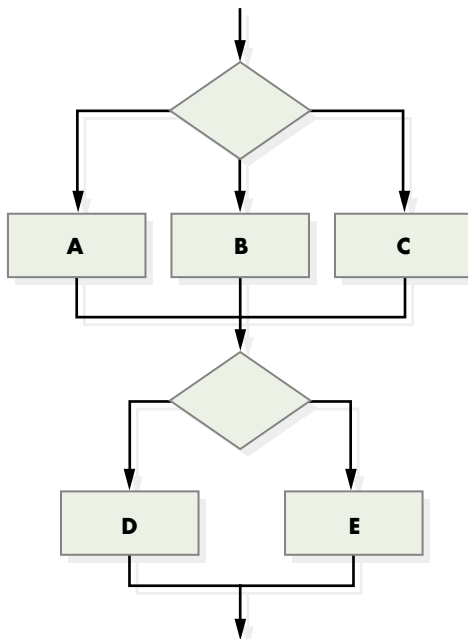
### Problem: Averaging in tens

A set of positive numbers, consisting of ten or fewer values, is to be averaged. If there are fewer than ten values, the end of the data will be signalled by a negative value. This process will continue until the first number input is negative, at which stage the process will stop.

### Test data considerations

Test data in this case should include the following: a set of exactly ten positive values; a set of fewer than ten positive values terminated by a negative value; two negative values in a row, one after a set of fewer than ten values followed immediately by another negative to terminate the program; a test for a negative after ten values; and, finally, a set of more than ten positive values in order to test that the program averages the first ten, then the others. The boundary conditions in this example are the maximum number of positive values to be processed and the negative value which terminates either the data input for a set of values or the program.

In order for the program to work properly, each of the paths through the algorithm must also be checked. The test data required for this purpose needs to be designed in a logical manner in order to ensure that no path is missed. The amount of test data required will also depend on the number of branches that can be taken during execution. There is a multiplying effect when a choice of path is made; for example, the addition of a two-way branch will usually double the amount of test data required, and a three-way branch will triple it. The possible paths through the algorithm represented by the flowchart in Figure 6.1 are:



through A and D, A and E, B and D, B and E, C and D, and C and E, giving six possible paths, all of which need to be tested. Many programs have a far more complex structure in which there are branches within branches of the main flow; problems posed by this structure are considered in the next section.

**Figure 6.1** Branching multiplies the number of possible execution paths through an algorithm.

## Example 4

### Problem: Postage classification

A postal item is classed as either a letter or a parcel. If it is a letter, it is charged at a rate of either 50 cents or 75 cents according to its weight. If the article is a parcel, it is charged at one of four different rates according to its weight (two categories) and distance travelled (also two categories). How many items of test data are required to test the charging algorithm for this problem?

### Test data considerations

We will assume, for this example, that validation test data is not included. Three items are required to test the letter module of the program (one each for both sides of the boundary and one on the boundary). The parcel data needs to include three different weights (below, on and above the boundary value) and three different distances (below, on and above the boundary value). This gives a total of nine different data items to test the parcel algorithm. Hence the total number of different test data items to fully test the algorithm is twelve.

## The test data dictionary

As with all stages of the systems development lifecycle, documentation plays an important part in the testing process. The purpose and expected output from each item of test data need to be known by the tester, who may not be the test data designer. A dictionary of data items is constructed to list all test data items, their expected outputs and the reason(s) for their inclusion in the set.

## Example 5

### Problem: Typist rating

A program is to be written to grade a typist's speed. If the speed in words per minute (wpm) is below 25, the typist is to be graded as 'Unsatisfactory—needs training'; if the speed is 25 wpm up to 55 wpm, the typist is to be graded as 'Satisfactory'; and if the speed is over 55 wpm, the typist is to be graded as 'Exceptional—receives a 10% bonus'.

### Test data dictionary

| Typing speed (wpm) | Expected output                  | Reason for inclusion                            |
|--------------------|----------------------------------|-------------------------------------------------|
| 24                 | Unsatisfactory—needs training    | Below the boundary value for satisfactory       |
| 25                 | Satisfactory                     | At the lower boundary value for satisfactory    |
| 26                 | Satisfactory                     | Within the satisfactory range                   |
| 55                 | Satisfactory                     | The upper boundary value for satisfactory       |
| 56                 | Exceptional—receives a 10% bonus | Above the upper boundary value for satisfactory |
| -1                 | Invalid input—out of range       | To test a validation procedure                  |

Table 6.3 Test data dictionary for the typist-rating problem.

## Example 6

### Problem: Letter postage

The boundary values are not expressed directly within the problem.

A program is to be written to calculate the cost of posting a letter according to the following rule: Letters with weights up to and including 150 g cost 50 cents. Each 100 g, or part, above 150 g is charged at a rate of 25 cents up to a maximum weight of 500 g. All letters over 500 g are rejected as parcels.

### Test data dictionary

| Letter weight (g) | Expected output                | Reason for inclusion             |
|-------------------|--------------------------------|----------------------------------|
| -100              | Invalid input—<br>out of range | To test validation procedures    |
| 145               | 50 cents                       | Under lowest boundary value      |
| 150               | 50 cents                       | Lowest boundary value            |
| 155               | 75 cents                       | Between next two boundary values |
| 250               | 75 cents                       | Boundary value                   |
| 255               | \$1.00                         | Between next two boundary values |
| 350               | \$1.00                         | Boundary value                   |
| 355               | \$1.25                         | Between next two boundary values |
| 450               | \$1.25                         | Boundary value                   |
| 455               | \$1.50                         | Between next two boundary values |
| 500               | \$1.50                         | Upper boundary value             |
| 505               | Parcel rates apply             | Above upper boundary value       |

**Table 6.4** Test data dictionary for the letter postage problem.

## Example 7

### Problem: Averaging in tens

Sets of positive numbers, consisting of ten or fewer values, are to be averaged; each set will consist of ten or fewer values. If there are fewer than ten values, the end of the data will be signalled by a negative value. Otherwise, the first ten values will be averaged, the eleventh value forming the first number in the next group. This process will continue until the first number input is negative, at which stage the process will stop.

### Test data dictionary

Each data item here is a list to be processed, instead of an individual item, as shown in Table 6.5.

*continued next page*



| Data item                                           | Expected output | Reason for inclusion                                                                                      |
|-----------------------------------------------------|-----------------|-----------------------------------------------------------------------------------------------------------|
| -1, 3                                               | Nil             | To test for termination on a first negative input                                                         |
| 3, 2, 1, 0, -1, -1                                  | 1.5             | To test for set of values with fewer than 10 elements, terminate the run afterwards                       |
| 10, 9, 8, 7, 6, 5, , 3, 2, 1, -1                    | 5.5             | To test for a set of values with exactly 10 elements, terminate the run afterwards                        |
| 10, 9, 8, 7, 6, 5, , 35.5, 2, 1, 0, 1, 2, 3, -1, -1 | 1.5             | To test for succeeding sets of values, the first with 10, the second with fewer                           |
| 0, 1, 2, 3, -1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, -1   | 5.5             | To test for succeeding sets of values, the first with fewer than 10 elements, the second with 10 elements |

**Table 6.5** Test data dictionary for the averaging in tens problem.

## Exercise 6.1

- Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.  
The main reason for testing a program is to find whether it \_\_\_\_\_ the \_\_\_\_\_ problem. In designing \_\_\_\_\_ data, we need to \_\_\_\_\_ all the decisions in the \_\_\_\_\_. Decisions involve \_\_\_\_\_ with a value known as a \_\_\_\_\_ value. The \_\_\_\_\_ data we design must also have \_\_\_\_\_ outcomes which can be compared with the \_\_\_\_\_ of our tests to determine whether the program works.
- Explain the reasons for including a dictionary of test data items in the system documentation.
- Explain the meanings of the terms 'test data' and 'boundary value'.
- What is the purpose of using test data during the program development cycle?
- A computer program is being designed for the police department to process the photographs from 'speed cameras'. Create a set of test data for the processing algorithm together with the expected outputs and reasons for choosing each item of data. Write an algorithm for this problem and test it using your test data. Code the program in one of the approved languages and use your test data to check the program. The fines are calculated according to Table 6.6.

| Speed above limit | Penalty                                     |
|-------------------|---------------------------------------------|
| 1 to 15 kmh       | Fine \$50 plus 4 points                     |
| 16 to 25 kmh      | Fine \$50 plus 6 points                     |
| 26 to 35 kmh      | Fine \$150 plus 8 points                    |
| over 35 kmh       | Fine \$280 plus loss of licence for 2 years |

**Table 6.6**

- 6 A plumber charges a service call fee of \$50 which includes the first 15 minutes labour. Labour after the first 15 minutes is charged at a rate of \$25 per 15 minutes or part thereof up to a maximum of 2 hours. After 2 hours the rate becomes \$60 per hour or part thereof. The plumber works a maximum of 8 hours in any one day. Create test data that will test a program module to calculate the bill, including data validation test data. Present your test data in a suitable form.
- 7 A particular algorithm contains three branches, the first with four choices, the second with two choices and the third with two choices. Draw a flowchart similar to Figure 6.1. Describe all possible execution paths through the algorithm.
- 8 Name all the different possible execution paths through each of the algorithm structures in Figures 6.2 and 6.3.

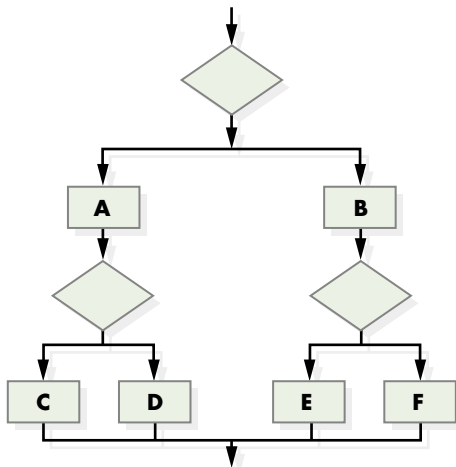


Figure 6.2

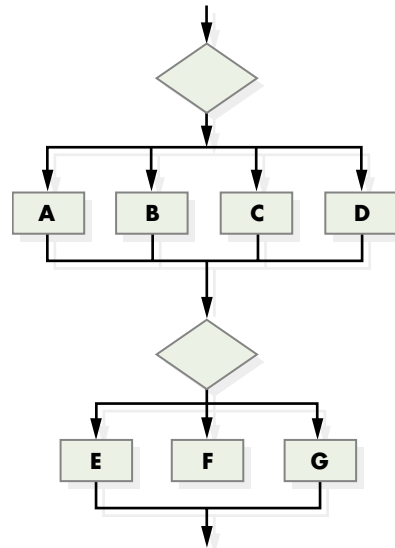


Figure 6.3

- 9 Determine the total number of test data items to test an algorithm for the following problem. Validation test data need not be included.  
 A store employs a credit rating system for customers who wish to pay off items on time payment. Customers are rated as 'ineligible' or 'eligible' according to their past credit records. Those customers rated as eligible are further rated according to their ability to pay. These ratings are 'A++', 'A+' and 'A' and determine the limit of their allowable borrowings.
- 10 Create a set of test data values for this problem. You do not, at this stage, have to determine the expected output values from the algorithm or program.  
 The federal government wishes to introduce a new scale of taxation based on the amounts shown in Table 6.7. A module is required to calculate an employee's weekly tax when the weekly wage in whole dollars is used as input.

| Weekly wage    | Tax payable                                 |
|----------------|---------------------------------------------|
| up to \$175Nil |                                             |
| \$176 to \$300 | 20 cents for eh dollar over \$175           |
| \$301 to \$500 | \$2 plus 35 cents for ach dollar over \$300 |
| over \$500     | \$95plus 55 cents for ch dollar over \$500  |

Table 6.7

- 11 Design a set of test data for this problem, giving the expected output from the algorithm for each item of test data. Data validation items are not required.

The Big Bytes Computer Shop sells floppy disks for \$20 a box. If customers buy more than one box, a discount system applies: The first 10 boxes are sold at full price, for orders up to 100 boxes; the remainder are given a 10% discount. For orders of 100 boxes or more, a bulk discount of 20% is given for the whole order.

### Testing both algorithms and coded solutions with test data

The success of a computer application is measured by its ability to perform the task as required. A program that contains errors or does not fulfil the design requirements could not be considered a success. A comprehensive program of testing will ensure that the computer application measures up to the required standards of performance. The main aim of testing is to cause and discover errors, not to show that a program functions correctly.

Before beginning the desk check of the program, it is a good idea to devise a **testing strategy**. The complexity of the program will determine whether the program can be tested as a whole or whether it is more efficient to test each module individually before examining their interaction. A small program is easy to test as a complete unit, as the number of possible execution paths is relatively low.

The process of **desk checking** involves tabulating all the variables and keeping track of their values while following the execution path. A method of accomplishing this is shown below.

#### Example 8

A Pascal program calculates the postage on a letter using the following charging scale: A letter weighing up to 150 g costs 50 cents, and each additional 100 g attract a charge of 25 cents, with a maximum weight of 500 g. Any letter over 500 g is rejected as a parcel with the message 'Overweight letter—parcel rates apply'. A number of letter masses are entered. A zero letter mass will terminate the program.

As the program was developed, the pseudocode description (next page) of the algorithm was created and then rewritten in Pascal. Test data was created as illustrated in Table 6.8.

| Letter weight (g) | Expected output (\$)                                 | Reason for inclusion                     |
|-------------------|------------------------------------------------------|------------------------------------------|
| 100, 150          | 0.50, 0.50                                           | Normal value, boundary values            |
| 200, 0            | 0.75, end                                            | Terminate with zero                      |
| 0, 100            | end                                                  | Terminate with no data entry             |
| 250, 300, 350     | 0.75, 1.00, 1.00                                     | Normal value, boundary value             |
| 400, 0            | 1.25, end                                            | Normal value                             |
| 450, -100, 1.25   | out of range, end                                    | Boundary values, invalid mass            |
| 475, 500, 550, 0  | 1.5, 1.50, overweight letter—parcel rates apply, end | Normal value, boundary value, overweight |

**Table 6.8** Test data for the letter postage program.

*continued next page*

### *Pseudocode*

```
BEGIN
get mass from user
REPEAT
 IF mass <= 150 THEN
 set cost to 0.5
 ELSE IF mass <= 250 THEN
 set cost to 0.75
 ELSE IF mass <= 350 THEN
 set cost to 1.0
 ELSE IF mass <= 450 THEN
 set cost to 1.25
 ELSE IF mass <= 500 THEN
 set cost to 1.5
 ELSE output 'Overweight letter:
 parcel rates apply'
 ENDIF
ENDIF
ENDIF
ENDIF
 get mass from user
UNTIL mass = 0
END
```

### *Pascal*

```
program Letter_rates;
 var
 mass, cost: real;
begin
 write('What is the mass of the letter?');
 readln(mass);
 repeat
 if mass <= 150 then
 cost := 0.5
 else if mass <= 250 then
 cost := 0.75
 else if mass <= 350 then
 cost := 1.0
 else if mass <= 450 then
 cost := 1.25
 else if mass <= 500 then
 cost := 1.5
 else
 writeln('Overweight letter – parcel rates apply');
 writeln('The cost of the letter is $', cost : 4 : 2);
 write('What is the mass of the letter?');
 readln(mass);
 until mass = 0
 end
```

*continued next page*

When desk checked, the results as outlined in Table 6.9 were obtained, and were compared with the expected output in the test dataset.

Note that as the desk check progresses with each data set, the old value of the variable is crossed out lightly and its new value is written by the side.

Two problems were found from this desk check; the terminating value of zero generates a charge of 50 cents before the program finishes, as does a negative mass. To overcome this problem, a software patch could be written which allows the cost to be printed out only if the mass of the letter is over 0 g. A better programming practice would be to rewrite the algorithm slightly so that these letter masses do not cause the program to generate a charge in the first place.

| Run number | Data values mass (g)  | Output (\$)                                         |
|------------|-----------------------|-----------------------------------------------------|
| 1          | 100, 150, 200, 0      | 0.50, 0.50, 0.75                                    |
| 2          | 0                     | 0.50*                                               |
| 3          | 250, 300, 350, 400, 0 | 0.75, 1.00, 1.00, 1.25                              |
| 4          | 450, -100, 0          | 1.25, 0.50*                                         |
| 5          | 475, 500, 550, 0      | 1.50, 1.50, overweight letter—<br>parce rates apply |

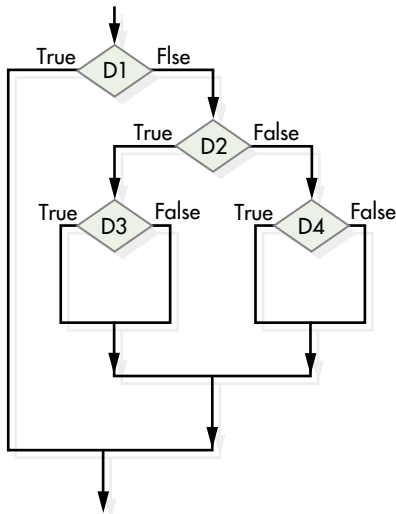
\* indicates an unexpected value which needs to be examined.

**Table 6.9** Desk check of the letter postage program.

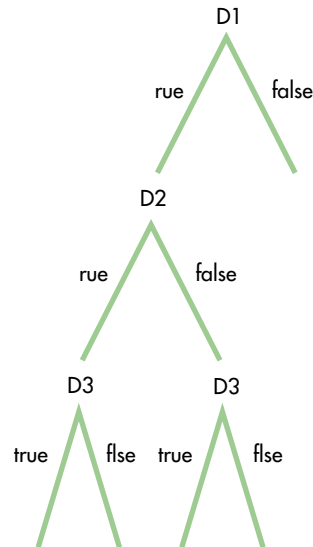
The above method of desk checking works well with small programs and with small subprograms or procedures, but to test a large program in this manner would be an almost impossible task. The programmer has to devise a strategy which is easily documented and which will obtain the desired result (a well-tested program). **White-box testing** is a process of thoroughly testing a module, rigorously following each of the execution paths. This type of testing was applied to the letter postage program above.

A thorough testing procedure will first check each of the modules within the application. This may necessitate using stubs to provide data values to be used as input rather than values supplied by earlier modules. For example, the testing procedure for the validation module in a barcode reading and validating program can be easily tested if an input stub is provided to accept the barcode in preference to a barcode scanner and its associated modules. Quite often in module testing, all paths will be checked using a method known as **statement-coverage testing**. In this method the decisions are tabulated, often with the aid of a decision tree or decision table, and test data items are created to force control through each of the possible paths. A problem with this method is that even for a smallish number of decisions a large number of test data items is needed. For example, up to sixteen test data items are needed for four binary selections (see Figures 6.4 and 6.5 for a sample algorithm and its execution paths). By keeping the size of the program modules down, exhaustive testing of each module can be undertaken. On completion of this process for a number of modules, the modules may be tested to examine their interaction. Test data for the typist rating program

discussed earlier illustrates how the test data is designed to cover all possible paths through an algorithm. When using this test data with the written program, the programmer will be statement-coverage testing.



**Figure 6.4** An algorithm including four binary selections within a module.



**Figure 6.5** The different execution paths that can be taken through the algorithm in Figure 6.4.

### Error types

Errors that can occur within modules include arithmetic, comparison, control logic, data structure and input/output errors.

#### Arithmetic errors

Arithmetic errors are those that occur in the processing of numerical data, the most common being divisions by zero and problems due to the order of calculation and truncation. For example, a program to calculate the length of a rectangle when the width and area are given as input data will not function properly if a negative or zero width is entered. The following BASIC program will exhibit arithmetic errors if a zero or negative length is entered. (Remember that the numbers before the instructions are line numbers which are used to order the statements.)

```

10 INPUT("WHAT IS THE AREA OF THE RECTANGLE"),AREA
20 INPUT("WHAT IS THE LENGTH OF THE RECTANGLE"),LENGTH
30 LET WIDTH = AREA / LENGTH
40 PRINT("THE WIDTH OF THE RECTANGLE IS",WIDTH)
50 END

```

#### Comparison errors

Comparison errors occur in the attempt to compare two different types of data elements (for example an integer with a character). They may also occur in the ordering of relational operations and nested comparisons as the results of these

are hard to predict. A comparison error can easily occur in a decision statement where the programmer wants two different conditions to be true at the same time, for example A to be greater than B and A to be greater than C. In English we tend to say 'If A is greater than B and C ...'. However, if we translate this into a language such as Pascal, taking A, B and C as integers, for example, the statement

```
if A > B and C
```

will produce a comparison error since the logical operation called AND will be applied to the two integers B and C, giving a Boolean result. When the computer tries to put this result in the location represented by the identifier A, an error will occur, since A is meant to store an integer and we are attempting to place a Boolean result there.

### Control logic errors

Control logic errors occur within the selection and repetition structures of the program. Selection structures need to cater for all possible selections and allow control to exit the structure for each of these selections. The main causes for concern in repetition structures are the proper initialisation of variables at the beginning of such a structure, incrementing by the correct amount and correct termination of the repetition.

For example, a program requires a loop which is to perform a particular operation three times. The programmer devises the algorithm shown in Figure 6.6 which is supposed to execute a particular operation (represented by a subprogram) three times. This structure contains a control logic error as the loop will only be executed twice: once when the variable index has a value of 2 (the first time around) and once when it has a value of 3 (the second and last time through). This error can be corrected simply by setting the initial value of index to 0 instead of 1.

### Data structure errors

Data structure errors include those in the initialisation of variables, tests for proper indexing of arrays and consistency in the use of identifiers. A data structure error would occur in a program where, for example, an array was indexed from 1 to 30 but the program tried to access data element number 31 from the array.

### Input/output errors

Input/output errors include those associated with data validation, and opening and closing files as well as special cases such as empty records or end-of-data markers. One of the most disastrous examples of this type of error is the case in which a file is updated by the program but the updated file is not saved to disk for later use.

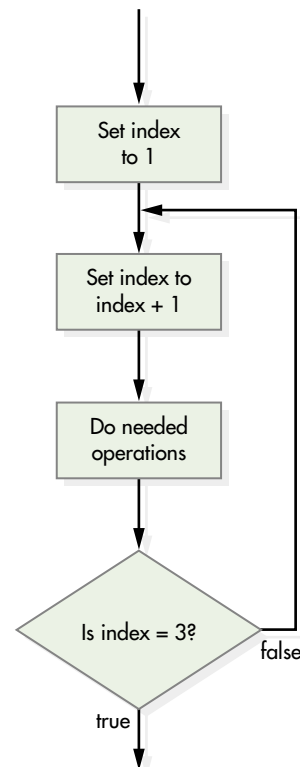
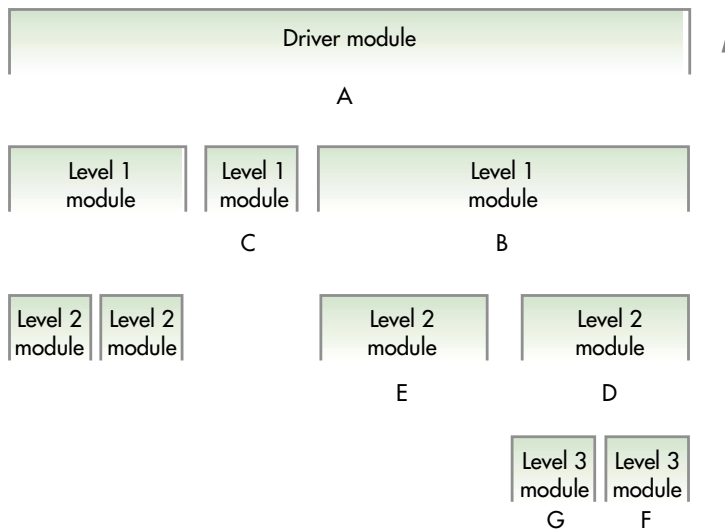


Figure 6.6 Algorithm with a control logic error.

## Bottom-up and top-down testing

The testing methods described above are usually described as **bottom-up testing**. This process involves testing from the lowest levels (simplest) of the module hierarchy upwards to the highest level (the main or control module). The bottom-up testing method complements the top-down programming process, as each of the modules developed can be tested as soon as it has been created.

Bottom-up testing is illustrated by Figure 6.7. Modules G and F are tested first to ensure that they work. Once these modules are functioning properly, module D (which lies on the next level above) can be tested. All other level 2 modules are similarly tested. The testing process can now be applied to the modules in level 1. When all modules at level 1 have been tested, the driver module can undergo its testing.



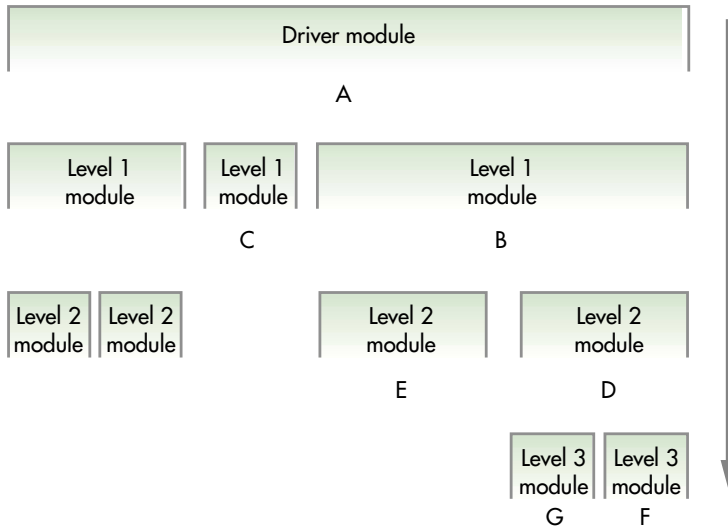
**Figure 6.7** Bottom-up testing starts at the smallest modules and works up to the whole program.

**Top-down testing** involves first testing the main or driver module, then working down the hierarchy to the lowest level modules. Top-down testing relies on the use of stubs to represent unwritten or untested modules at that stage. An advantage of top-down testing is that the main modules can be tested in a situation that simulates operation and a partial system may be implemented at an earlier stage of development. However, top-down testing has some disadvantages, including difficulty in simulating the actions of lower-level stubs and trouble with the creation of suitable output from lower-level stubs.

Top-down is illustrated by Figure 6.8. The driver module A is tested first using stubs to represent the modules at level 1. Once module A is functioning properly, modules B and C and all other level 1 modules will be tested using stubs for level 2 modules. Similarly, level 2 modules are tested next. When all modules at higher levels have been tested, modules F and G, at the lowest level, can be tested.

When the program has successfully passed through the desk-checking stage, it is ready for run-time checking. In this process, the program is run on a computer using first the test data, then some real data for which the outputs are already known.





**Figure 6.7** Top-down testing starts at the driver module and works down to the smallest modules.

## Exercise 6.2

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

Errors within a program or a module are \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_ errors. \_\_\_\_\_ errors occur in the processing of numerical data. Some of these errors may be \_\_\_\_\_ by \_\_\_\_\_ the order in which operations are performed. Logic errors occur inside a \_\_\_\_\_ or \_\_\_\_\_ structure. \_\_\_\_\_ structure errors may occur during the \_\_\_\_\_ of variables, \_\_\_\_\_ of arrays and \_\_\_\_\_ in using identifiers.

- 2 Describe in your own words the process of desk checking a program.
- 3 Describe the different types of error that may occur in a program, giving examples to illustrate your answer.
- 4 Perform a desk check on the following LOGO procedure to draw a polygon with a given number of sides (called SIDE), each side having a length of 10 units. (For the desk check, 10 units is a length of 5 cm.)

```
TO POLYGON : SIDE
 REPEAT SIDE(FORWARD 10 RIGHT (360/SIDE))
END
```

Use a sheet of drawing paper, a ruler and a protractor to perform a desk check using the following values of SIDE: 3, 4, 2, 1 and 0. Does the program fulfil the design brief? What errors are there in the program?

- 5 The following code, which performs the task of adding up the marks for a test and outputting the results, contains an error. Determine the nature of the error and correct it. (A knowledge of the imaginary language is not necessary to answer this question, as there are no syntax errors in this code.)

```

MODULE addmarks ! **This module adds the marks for each question **
 INPUT: question[1..10]!
 OUTPUT: total_mark !
 VARIABLE
 index is INTEGER !
 BEGIN
 index <- 0 !
 total_mark <- 0 !
 WHILE index <= 10
 total_mark <- total_mark + question[index] !
 index <- index + 1
 ENDWHILE
 END
ENDMODULE ! **total_mark now contains the total score for the test**

```

- 6 Describe a problem of your own choosing, and devise a full set of test data for the problem together with the expected outputs. Create an algorithm to solve the problem and desk check your algorithm. Code your solution in a programming language and test the program.
- 

## Evaluation of design

Evaluation of software design can be thought of as two processes—**verification** and **validation**. Verification is the process in which the software is tested to see whether it performs its functions correctly. Validation is the process where the software is evaluated against the original specifications to see whether it will perform the tasks specified in the problem definition.

Although these two processes may appear similar, in fact they are completely different. It is quite easy to build a software solution and check that it performs a task. However if the task performed is not going to completely solve the problem, then the software is not successful.

In the Cumfy Coaches example in Chapter 3, the aim of the software is to manage the bookings for one or more coach tours. If the software allows bookings to be made and recorded, but will not allow changes to be made to customer details or bookings to be cancelled, then the software will not completely perform the required tasks. In this case the software would pass the verification tests, as it does allow customers to book a tour and it also stores the details. However, the same software application would fail when it comes to validation as it would not have the required editing functions.

### Comparing different solutions to the same problem

We often think that there is only one correct solution to a problem, but there are often a large number of alternative solutions. For example, if the problem were how to travel from Sydney to Darwin, we might decide on the solution of flying there directly, as this seems to be the most appropriate. But this solution is not the only way of travelling to Darwin. Others that come to mind are to drive (this solution itself has many different solutions as the route taken could vary), ride a pushbike, or even go on a cruise liner that is heading to Darwin. Each solution

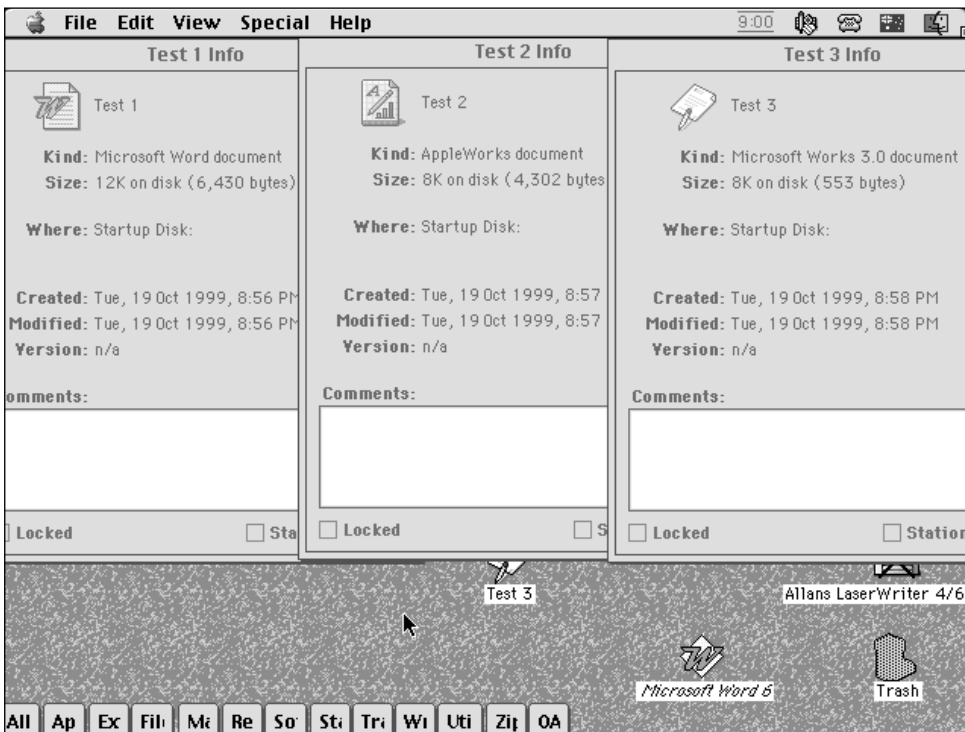
will take you to Darwin, but the circumstances of the trip may make one of the solutions a better proposition than the others. Thus if we want to travel there as quickly as possible, we will most likely fly. However, if we wish to see the coast between Sydney and Darwin and are in no hurry, we may decide to ride a bicycle.

The same ideas apply in software development, although not necessarily in as extreme a form as the above example. Different programmers will see a particular problem in their own different ways. When given a problem to solve, each programmer may design a completely different solution from that of their colleagues. If each of the solutions completely solves the problem, which is correct?

Differing interpretations of a set of design specifications by different people may lead to completely different interpretations of what the problem is in the first place. This can cause problems as an interpretation of the design specifications may miss out on important points within the specifications. If this occurs, it is possible that the software may not fully meet all the requirements.

When a number of software solutions to a problem are compared and evaluated, their differences may be quite obvious. For example, the interface of one of the solutions may be more intuitive, whereas another solution may provide a faster processing time or more efficient data storage. Each of these solutions has its advantages and its disadvantages.

We can easily see the differences in approach to a solution when we compare commercial software applications that perform a similar task, for example a number of word-processing applications. Each of these applications performs the same basic task, but some word processors will produce files that are much larger than others, since the way in which the formatting information is coded into the file affects the size of that file. This can be illustrated by the information in



**Figure 6.9** Files containing the same data may occupy different amounts of storage if created by different applications. These three files contained no typed text, but used up vastly different amounts of storage.

Figure 6.9. Each of the files was created by a word processor or a word-processing module of an integrated package. These files contained no visible characters, but their sizes on disk varied from 553 bytes to 6430 bytes.

An advantage for a programmer in examining different solutions to a problem is that his or her experience is broadened, and the programmer can draw upon this experience in the future to help solve other problems. The sharing of these experiences among programmers also enhances the working relationship needed within a design group.

## Methods of evaluation

### Peer checking

Peers are people of the same standing in an organisation or group of people. One of the most effective ways of evaluating a solution is for a peer to check a design. Peers feel more comfortable with each other as they are all at the same 'level' of the organisation, and so feel that they can express an honest opinion (whereas they may not want to risk offending someone who has authority over them). Honesty is one of the most valuable tools when it comes to evaluation of a software design, as a lack of constructive, honest criticism might allow a faulty product to be released.

Peer checking of a software solution is also beneficial because the checker is not directly involved with the design of the product and therefore does not take anything for granted. When people check their own work, errors can remain undetected, since assumptions can be made about the way in which parts function.

### Structured walk-through

A structured walk-through is a clearly defined process in which a number of peers take on particular roles. The main aim of a structured walk-through is to identify errors that might otherwise remain undetected. The word 'structured' in this context means that the process is well organised. The term 'walk-through' refers to the process of passing through the product step by step in order to find the errors. The philosophy behind a structured walk-through is similar to the philosophy behind the process of peer checking, that is, that it is easier for someone other than the designer to identify faults in a product. The aim of the walk-through is solely to find the errors. It does not attempt to rectify the errors, nor does it allot the blame to anyone.

There are a number of roles that are performed by the members of a walk-through team. They are:

- The **presenter** submits the product for review. In software development this is usually the author of the program.
- The **coordinator** is responsible for the overall running of the walk-through. The coordinator acts as a chairperson ensuring that discussion remains on track.
- The **scribe** is responsible for keeping an accurate record of the results of the walk-through.
- A **maintenance representative** is concerned with the future ease of maintenance of the product.
- The **standards leader** is responsible for the consistency of the product and for ensuring that it meets the standards set down by the customer or organisation.
- A **customer representative** ensures that the product meets the needs of the user.

These people may be joined by others to form a larger group, although the walk-through group should not become too large as all members need the opportunity to make their contribution. For some projects some of the roles may be combined; for example, the coordinator might also act as the scribe.

Before the walk-through the members of the team should be provided with the documentation so they can read through it. During this process they should attempt to provide one positive and one negative comment about the product, although this may not necessarily happen. The coordinator needs to select an appropriate time and place for the walk-through. The process usually takes between half an hour and one hour, although it might last up to two hours. Any time longer than two hours becomes counter-productive as team members become tired and their concentration lapses. If a walk-through appears to need more than two hours then the product should be divided into smaller parts.

At the appointed time and place the coordinator begins the proceedings by giving the presenter the floor. The presenter first gives an overview, and then a detailed presentation of the product piece by piece. The reviewing team should make constructive comments, criticisms and suggestions, and provide lists of errors to both the scribe and the producer of the product. The comments should always be about the product and not the producer. The walk-through is not an opportunity to attack the producer of the software. Similarly, the producer should not be defensive about the product, any criticisms being purely of the product. The scribe makes notes of the proceedings, incorporating any written comments that come from the team members.

At the conclusion of the walk-through, the team makes a recommendation:

- to accept the product in its present form; or
- to accept the product with the suggested revisions; or
- that the product be revised, and then pass through another walk-through.

### Desk checking

Although a desk check is used to detect errors in an algorithm or coded solution, it also provides an opportunity to evaluate software design. As a program is being desk checked, observations can be made about the overall design of the part being tested, and judgments made about the manner in which the software works. For example, if the desk checker identified that certain steps were repeated in different parts of the code, the programming team would be given this information and could then form these steps into a subroutine and thus decrease the amount of code in the final application.

## Evaluation of implemented solution

It must be remembered that the aim in writing a computer program is to solve a particular problem. Programmers should not get carried away while writing a program and provide a number of fancy additions but not completely solve the original problem. For example, adding a module to a word processor that allows the word processor to speak the typed text is of no use if the word processor does not properly save the text to disk. Evaluation of the proposed solution to a problem is a very important part of the software development cycle.

As well as being able to perform its task, a software application needs to have a **human orientation**. This means that the user should feel comfortable using the software, and that it should reflect the appropriate social and ethical perspectives.

## Checking the solution to see if it meets the original design specifications

The design specifications for a computer solution are a detailed description of what the program should do. Within the specifications should be a set of criteria to measure the performance of the system. This is best illustrated by looking at a non-computer example. If you decide to buy a motor vehicle, you decide on your requirements before you go to look at those for sale. Each vehicle you examine is measured up against the list that you have created. You then decide on the best one for the job. In software development, we do not look at a large number of different solutions, but we do have our list of requirements in the form of the software specifications. The solution is then measured up against those specifications.

Good software specifications are written in a form that allows us to measure the product against measurable requirements. By ensuring that we have a measurable scale for comparison, we can state whether the software fulfils the specifications. For example, a graphics program may rescale a drawing, but if it takes half an hour to perform this task, then the software may have failed to measure up to the time specified in the original requirements of the program.

## User feedback

Evaluation information provided by a user can be more valuable than that from the development team. First, the user is familiar with the existing system; and second, the user is not involved with the development process. Because of their familiarity with the way things are done in the present system, users can quickly determine whether a task is not being performed satisfactorily. Being away from the development process, they can also comment independently on the operation of the software.

User feedback should be sought on the following:

- the operation of the system
- the ease of use and suitability of the user interface
- response times for various tasks
- the overall 'feel' of the software application
- ease of learning of the software
- functions that may be improved or that have been omitted.

## Social and ethical perspectives

As we noted in Chapter 1, there are a number of social and ethical perspectives which need to be addressed during the process of software development. The software needs to be evaluated in terms of:

- *Copyright*. Has there been a use of copyrighted code? If there has, then have all the requirements of the licences been met?
- *Ergonomics*. Does the software provide a sound ergonomic working environment?
- *User friendliness*. Does the software perform in a user-friendly way?
- *Inclusivity*. Is the software designed for use by the widest possible audience? Does it provide an environment which is free of economic, cultural, gender and social bias and provide equal opportunity for those with disabilities?

## Exercise 6.3

---

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

The \_\_\_\_\_ of the design of a software solution consists of two processes: \_\_\_\_\_ and verification. \_\_\_\_\_ determines whether the \_\_\_\_\_ meets the original \_\_\_\_\_ specifications. Verification is the process which \_\_\_\_\_ whether the software \_\_\_\_\_ its \_\_\_\_\_ correctly.

- 2 Devise a set of specifications for a word processor that would be suitable for children up to the age of ten. Evaluate the word-processing program that you use at school against these specifications. Write a report on your findings. Make recommendations as to how the word processor could be improved for that group of users.
  - 3 Choose an algorithm that you have created earlier in the course and organise and run a structured walk-through to evaluate the algorithm. Examine the results of the walk-through and modify your algorithm to take the suggestions into account.
  - 4 Choose a productivity software application that you have not used before and evaluate its performance. Your evaluation should be in the form of a report and cover all the aspects listed in the section above titled 'User feedback'. Suggest any improvements that could be made to the software.
  - 5 Compare as many different spreadsheet applications as you can for their suitability for use within the school. Describe the similarities and differences in the way that each of them performs its functions. Present your findings as a report.
-

# Review exercises

- 1 Describe the purpose of test data during the systems development cycle.
- 2 Determine the boundary values for the following problem: A real estate agent charges according to a certain commission structure: 'Commission is calculated at 2% for the first \$150 000 of the selling price, 1.5% for the next \$200 000 and 1% on the amount thereafter.'
- 3 An algorithm is to be designed which contains a 3-way branch and two 2-way branches. Draw a diagram showing the structure of the algorithm (see Figure 6.1 for an example) and determine the number of test data items required to test every path (excluding validation).
- 4 Design a set of test data, excluding data validation, for the following problem and present it in an appropriate form: A refrigerator alarm system is to be installed in a commercial kitchen. The alarm is to sound if the door has been left open for more than 45 seconds or if the internal temperature rises above 2°C. Inputs to the system are taken from a stopwatch, which starts when the door is opened and which stops and resets when the door is closed, and from a temperature sensor. These values are converted by the system into numerical values, which represent the number of seconds and the temperature in degrees Celsius, and then sent to the processor.
- 5 The algorithm in Figure 6.10 has been designed to total a number of scores and output that total. The first value

in the set is the number of scores to be totalled; the numbers following are the data. Create a set of documented test data to fully check this algorithm. Use your test data to determine whether the algorithm performs the task for which it was designed. What problems, if any, are there with the algorithm?

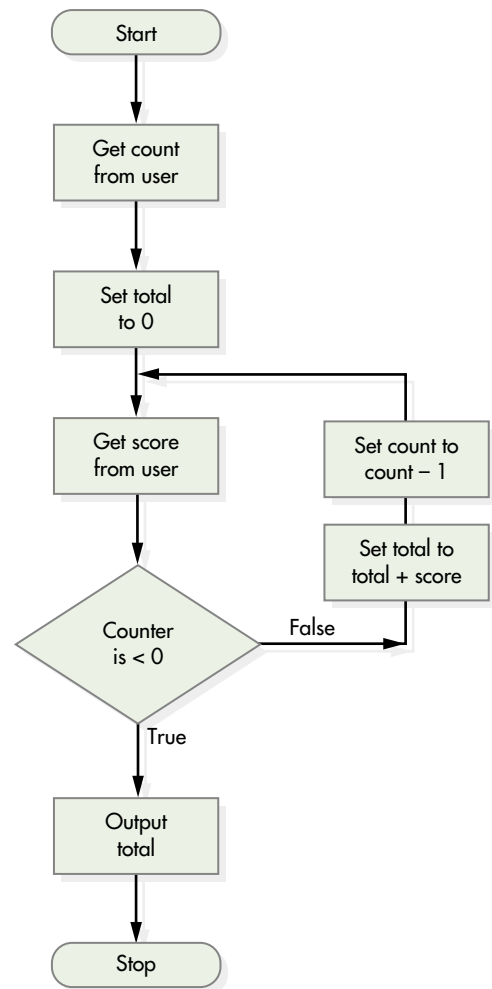


Figure 6.10



# Review exercises

- 6 Describe the social and ethical aspects that are relevant to the Cumfy Coaches case study encountered in Chapter 3.
- 7 Organise a team to perform a structured walk-through. Name each member of your team and describe the role that each is to perform during the walk-through. Perform the walk-through on the algorithm in question 5.
- 8 Code the corrected algorithm from question 5 in a programming language and perform a desk check on it. Evaluate it against its design specifications

## Team Activity

Choose a class of software such as word processors and draw up a design specification for that software application. As a group, evaluate a software application of that type using your specifications. Write a

report that gives your evaluation of the chosen application, making suggestions for the improvement of the software application based on your findings.

# Chapter summary

- A program is tested at each stage of its development to make sure that it will perform the required tasks.
- The process of testing is designed to cause and discover errors.
- Test data are devised for use in the testing process. The output for each item is also known.
- Test data are carefully chosen to test all parts of the algorithm or program.
- Test datasets should include each of the boundary values together with items that are on both sides of each boundary value.
- Both the algorithm and the coded solution should be tested for errors using the set of test data.
- Desk checking is a testing process in which the steps of the algorithm or program are followed manually, at the same time keeping track of the values of variables on paper.
- Errors may be classified as arithmetic errors, comparison errors, control logic errors, data structure errors and input/output errors.
- Arithmetic errors occur in processing numerical data.
- Comparison errors occur within decision statements and involve data.
- Control logic errors occur within selection and repetition structures and involve execution following a wrong path for one or more data values.
- Data structure errors occur in the initialisation or use of variables.
- Input/output errors occur in file reading/writing and in data validation modules.
- The evaluation of software design consists of verification and validation.
- Verification is the process in which the software is tested to see whether it functions correctly.
- Validation is the process in which the software is measured against the original specifications to see whether it will perform the required tasks.
- Different solutions to the same problem may be compared in order to evaluate the effectiveness of the different solutions.
- Peer checking is a process in which software is tested by someone other than the designer who is at the same level in the organisation.
- A structured walk-through is a formal process in which the developer describes the workings of the program while the members of the group evaluate it as a product. People in a structured walk-through have designated roles.
- Software is evaluated with regard to a number of criteria.
- The software must meet the original design specifications.
- The software is evaluated by the end user.
- The software is evaluated in terms of its social and ethical perspectives.

# chapter 7

## → *Modifying software solutions*

### Outcomes

- describes and uses appropriate data types (P 1.2)
- explains the effects of historical developments on current practices (P 2.2)
- identifies the issues relating to the use of software solutions (P 3.1)
- analyses a given problem in order to generate a computer-based solution (P 4.1)
- investigates a structured approach in the design and implementation of a software solution (P 4.2)
- uses a variety of development approaches to generate software solutions and distinguishes between these approaches (P 4.3)
- uses and justifies the need for appropriate project management techniques (P 5.1)
- uses and develops documentation to communicate software solutions to others (P 5.2)
- describes the role of personnel involved in software development (P 6.1)
- communicates with appropriate personnel throughout the software development process (P 6.2)
- designs and constructs software solutions with appropriate interfaces (P 6.3)

### Students learn about:

#### Reasons for maintenance coding

- changing user requirements
- upgrading the user interface
- changes in the data to be processed
- introduction of new hardware or software

- changing organisational focus
- changes in government requirements
- poorly implemented code

#### Social and ethical implications

- plagiarism

#### Features in source code that improve its maintainability, including:

- use of variables instead of literal constants
- use of meaningful variable names
- explanation comments in the code
- use of standard control structures
- a clear and uncluttered mainline
- one logical task per subroutine

#### Interpretation

- reading original documentation in order to understand the code
  - documents for the user (including user manuals)
  - documents for software developers
- reading original algorithms to identify:
  - inputs to the algorithm
  - processes used
  - the types of variables used
  - outputs
- creating algorithms for source code when they are not available

#### Documentation

- using supplied documentation to:
  - identify the control structures that have been used
  - explain how variables have been used

## Students learn to:

- identify features in code, scripts or macros that allow them to be easily maintained and explain how this can be achieved
- create solutions to ensure ease of maintenance
- modify original statements obtained from a variety of sources
- convert a fragment of source code, macro or script into its equivalent algorithm
- define the purpose of the code, macro or script to be maintained

## Personal Profile—Kay McNulty Mauchly Antonelli (1921– )

Kay McNulty Mauchly Antonelli was born in Donegal, Ireland on 12 February 1921. In 1924 her family emigrated to the United States of America, settling in Philadelphia where she attended school. One of only three mathematics majors in a class of 93 at Chestnut Hill College for Women, Kay graduated in June 1942.

An army advertisement in a newspaper calling for women with a degree in mathematics caught her eye. She applied for the job and was accepted for a position as a computer at the Moore School. A computer performed the calculations needed to create tables of trajectories for the war effort. In order to do the job, Kay had to perform a mathematical process known as numeric integration. Unfortunately, she had no idea of how to do numeric integration. Kay and the other women employed at the same time were supplied with textbooks, calculators and paper by the army but no instruction. Having taught themselves the process, the group of women were put to work.

Kay's office contained twelve women and four men. They were kept very busy with calculators and large sheets of columned paper. The men in the office often gave the women a hard time, and Kay sought an escape from the office by venturing down to the basement where there was a differential analyser. Kay was quick to learn how to operate the differential analyser, becoming very skilled in the complex task of setting it up and obtaining the results of its calculations. Working eight-hour days for six days a week, Kay became very familiar with the way in which it worked. As the few men left were drafted to the war effort, Kay took on a leadership role with the differential analyser.

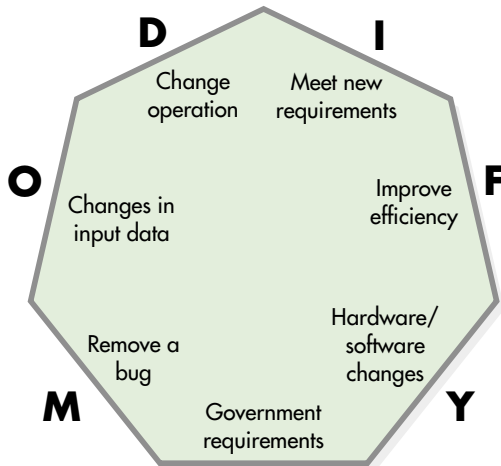
Together with a number of other women, Kay then began work on the new digital computer, ENIAC. Programming ENIAC consisted of plugging in connections between panels and working switches. The women running ENIAC became well-versed in its operation, learning how to debug the machine as well as to program it. ENIAC holds a special place in Kay's life as not only did she work with the machine but she also married one of its inventors, John Mauchly.

Kay was one of the original six ENIAC programmers and, as such, had no handbooks or instructions on how the computer worked. She is now recognised as being one of the first digital computer programmers of all time. She was honoured at the 50th anniversary celebration of ENIAC for her contributions to computer programming.



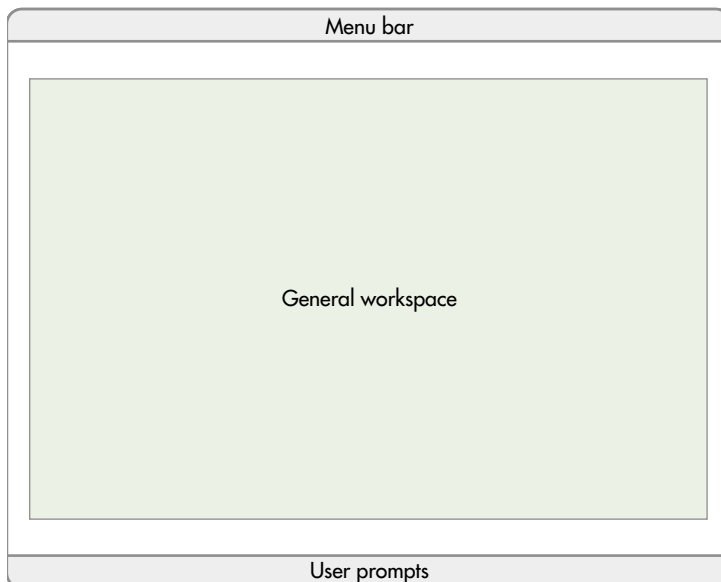
## Reasons for maintenance coding

Once written, a computer program may still need to be modified. The modification of existing code is known as **maintenance coding**. The purpose of maintenance coding is to remove a bug, to improve the efficiency of the program or to change the manner in which the program operates, to allow the program to cope with changes in the input data, to meet new requirements within the organisation or to comply with new government requirements. Maintenance may also be necessary when changes are made to hardware or to software such as the operating system.



**Figure 7.1** Maintenance may be required for a number of reasons.

A well-written and documented program is easier to maintain as the internal documentation provides a guide to the present operation. Modularisation also assists in maintenance as a module may be removed, modified or replaced without affecting the rest of the program. As we have seen, modularisation provides a further benefit in that the programmer is able to use a subprogram, or modified subprogram, in another application, thus saving development time. A library of subprograms can be developed, each needing only minor maintenance in order to be used in a different application. For example, a main menu screen can be designed to be **generic**; that is, it is applicable to a large number of programs, each individual program using a customised version.



**Figure 7.2** A generic screen design can be easily modified to suit a large number of different applications.

## Changing user requirements

As a user becomes more familiar and competent in using a computer program, shortcomings in its operation may become evident. These shortcomings may be in the way that the program works or, more often, the user may wish or need it to perform a further task.

### Example 1

The Cumfy Coaches case study presented in Chapter 3 looked at a software solution that would manage the bookings for the company. During the year following the introduction of this system, the users have found some shortcomings in the software. The first involves the manner in which the user passes from screen to screen and the second involves the manual preparation of confirmation letters that include the amount still owing for the trip. These confirmation letters are sent to passengers four weeks prior to the tour's departure. A request is made to the original development team to modify the software so that these needs are met.

## Upgrading the user interface

When software is originally developed, the user interface receives a lot of attention. However, it is often found that the user interface contains a number of aspects that would benefit from redesign.

Redesign of the user interface may be needed because the user, in becoming more familiar with the original interface, has discovered design problems. It is common for this to happen as, although the user is involved with the original interface design, its long-term use may bring to light problems that either were not thought of or have been brought on by slightly different work practices. This can be especially true in the design of menus and key combinations used for shortcuts.

In the Cumfy Coaches example, the user has reported that it is cumbersome to have to navigate back to the main menu in order to correct an incorrectly entered name. In order to solve this problem, the program is modified to take the

**Cumfy Coaches**  
CUSTOMER RECORD ENTRY

Name.....  
Address.....  
Tour booked.....  
Deposit taken \$.....

To edit this record, please press the **ESCAPE** key  
and choose **EDIT** from the main menu

**Figure 7.3** The screen design has been found to be less easy to use than the designers thought it would be.

user's recommendations into account and the interface is redesigned to allow the editing of the incorrect name from within the data entry module of the program.

A further problem that was identified by the user was the placement of some of the menu items. The suggestion was made to group the items differently, placing those most frequently used at the top of each of the drop-down lists.

### **Changes in the data to be processed**

Changes in the form of the data to be processed will often lead to a need to modify the software. The most recent worldwide example of this problem was in the correction of the so-called 'Millennium Bug' in which the two-digit representation of the year in a date, widely used in the last half of the 1900s and identified as a potential source of problems, needed to be changed to four digits. Date-dependent software had to be modified in order to accommodate the new representation of the year.

Expansion of the existing system to allow for a greater number of data items than was originally envisaged when the software was designed may involve revising the representation of data items and/or data structures. These revisions then have to be incorporated into the software if it is to function properly. An example of this in Australia's recent history was the conversion of all telephone numbers to eight digits in the late 1990s in order to increase the number of possible telephone connections.

Another source of change in the data to be processed is brought about by the changing needs of the organisation. In this case new data items and processes may need to be incorporated into the software application in order to obtain the required outputs. For example, the federal government introduced legislation requiring all citizens to have a Tax File Number. This meant that banks had to modify their software to allow Tax File Numbers to be stored and retrieved as well as to be linked to interest calculations.

### **Introduction of new hardware or software**

Custom software generally has a long life within an organisation as it is initially expensive to produce. Cases of software lives of ten to fifteen years are not uncommon. However, during the life of the software, technology is improving both the hardware and the operating system software. For software to last as long as it does, maintenance will be required first to allow the software to run with the new technology and second to take advantage of the benefits that the new technology offers.

When automatic teller machines (ATMs) were introduced by the banks, the existing software was not discarded in favour of completely new software. Rather, the ATMs were integrated into the existing system. The banks took this path for two reasons: a reduction in development time and a reduction in the cost of ATM introduction. The development time needed for the introduction of the ATMs was reduced by rewriting parts of the software to allow for ATM transactions. And since the existing software performed its tasks well, it was more economical to integrate the ATMs into the existing system.

### **Changing organisational focus**

An organisation is not static. Its goals and purpose move through a continuous process of evolution. As the organisation evolves, its focus changes. For example, P&O started as a shipping company, with its focus on providing a service to Asia and Australasia. However, P&O has now evolved into a company that owns cruise



liners and holiday resorts, provides freight services, runs a cleaning service and operates wharves.

As the focus of an organisation moves through change, the software needs also change. Companies generally move into areas that are related to their original purpose, so the new software needs will generally be similar to their previous ones. Rather than create a completely new software package to carry out the new processing required by the change in focus, a company usually finds that its software can perform a majority of the required tasks, so, again, it is more economical in both time and cost to modify the current software application.

For example, Cumfy Coaches decides to run a regular bus service to Sydney. It will need a booking system in order to perform this service. The booking system will have a number of similarities to their present business, but the software will need to be modified to allow bookings to be entered into their system by Sydney agents.

### Changes in government requirements

Government regulations and laws are continually changing to reflect social, political and economic circumstances. As these laws change, it may be necessary to modify an organisation's software in order to comply with the new requirements.

In the case of Cumfy Coaches, the introduction of the Goods and Services Tax in mid-2000 meant that customers had to pay the tax on their bookings. The software needed to be modified to both charge the tax and keep track of the tax that was due to be paid to the Taxation Department.

Areas of government regulation in which these changes are most evident to organisations are those that involve the payment of taxes, but there are other cases in which software may need modification. For example, the government may impose laws or regulations that require a company to track the use of particular chemicals or to furnish details of the purchasers of their products. In these cases as well, software will need modification in order to comply with the government's wishes.

### Poorly implemented code

As discussed earlier, software often has a life of many years. In the past the development of software was usually not as structured as it is today; now the emphasis is on the software being able to perform its task no matter what the cost. As software that has been in use for a number of years comes up for maintenance it may become evident that it is not as efficient as it could be, or

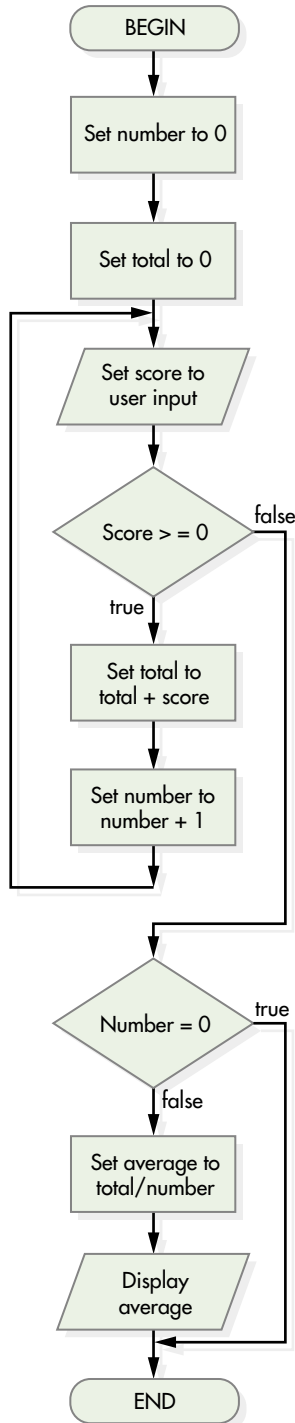


**Figure 7.4** The introduction of ATM technology into banks was accompanied by maintenance of the software to accommodate the change.

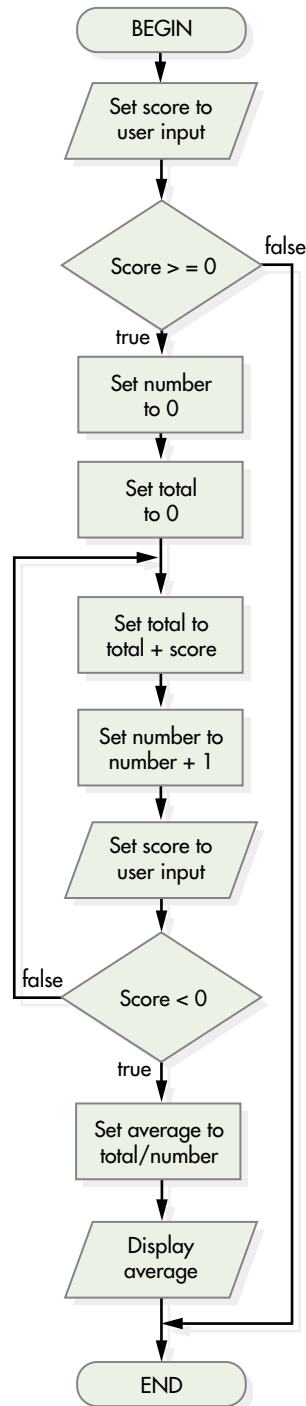


**Figure 7.5** Changes in federal or state legislation may force an organisation to maintain software to meet its new legal requirements.

that a software patch has been used to work around a problem. Maintenance is a good time to evaluate the code for its efficiency and properly overcome earlier problems rather than relying on a software patch.



**Figure 7.6** The original algorithm with the patch.



**Figure 7.7** The rewritten algorithm that performs the same task more efficiently.

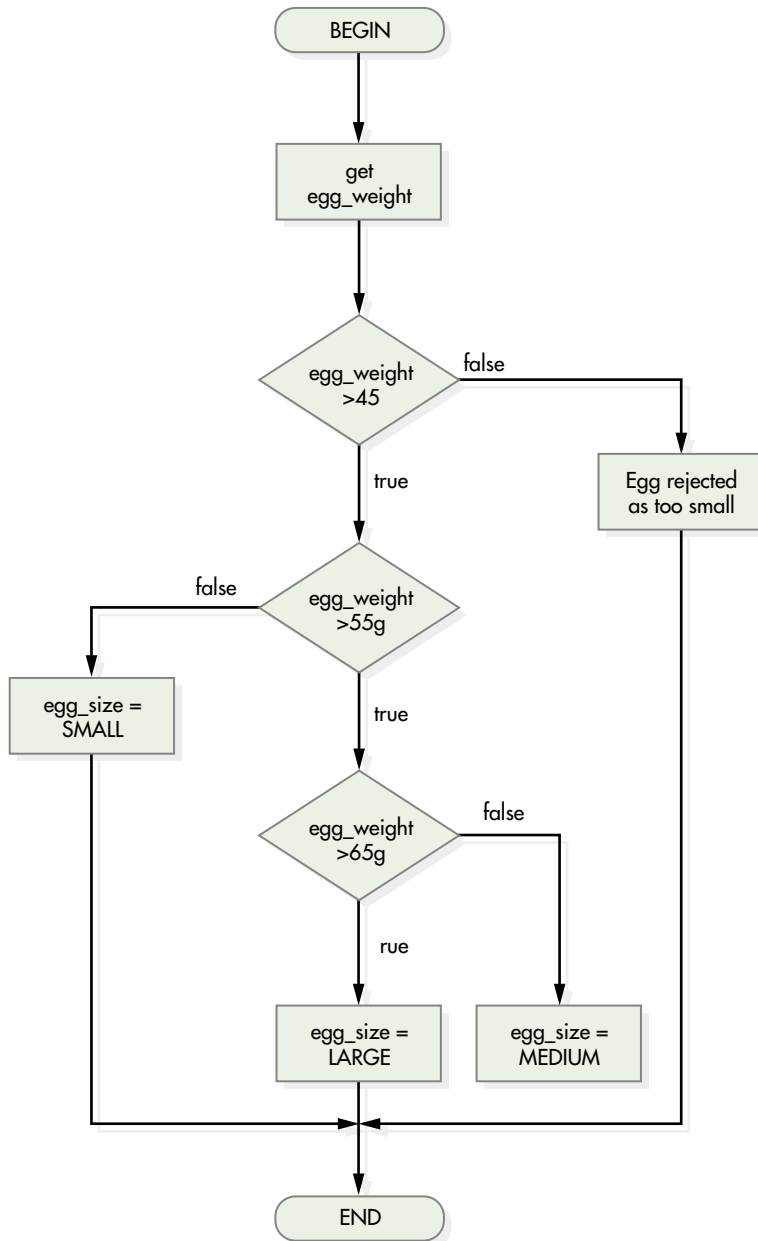
For example, the following algorithm is to average a number of scores, terminating when a negative score is entered. However, in the original implementation it was found that, if a negative score was entered first an error occurred, so to overcome this problem a 'patch' was added. The purpose of the patch is to allow the calculation of the average only if there is an entry of at least one positive score. This is not an entirely satisfactory solution, so the algorithm was rewritten to ensure that if a negative score is entered first, then the average will not be calculated.

## Exercise 7.1

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

The modification of a computer program is known as \_\_\_\_\_. This process may need to be performed if the program has a \_\_\_\_\_, to improve its \_\_\_\_\_, to change the way the \_\_\_\_\_ operates to allow the \_\_\_\_\_ to cope with changes in \_\_\_\_\_ or to meet new \_\_\_\_\_ of either the \_\_\_\_\_ or the \_\_\_\_\_.

- 2 A school library management program is being reviewed in order to perform maintenance on the software. The following shortcomings have been identified. Identify the modification that needs to be made using the descriptions in this section, giving reasons for your choice.
- a The library was using a four-digit code to represent each of the books in the collection. The collection is now approaching 9500 volumes and so the code is to be changed to a six-digit code.
  - b The library system is to be integrated with the administrative system in order to use student data files from the administrative system.
  - c The clerical staff want to use the system to produce overdue notices to be sent to students' homes.
  - d The federal government passed a law that required all libraries in the country to furnish borrowing details to the National Library in Canberra so that authors can be paid a royalty fee each time a book has been borrowed from a library.
  - e The library computer system is being updated to use the capabilities of a new generation of computer terminal.
  - f Students' fingerprints will be used to identify the borrower of a book instead of the magnetic student card now used.
  - g At present the software cannot process a book number that ends with 000. The librarian would like all book codes to be able to be used.
  - h When a book is withdrawn from the library, the book number cannot be reassigned to a new book. The librarian would like to be able to reuse these numbers.
- 3 A supermarket is introducing portable customer terminals (PCT) which automatically register the price of an item as it is placed in the supermarket trolley. At the checkout the PCT is connected to the cash register by the checkout operator. The register then prints out an itemised docket and the customer pays in the normal way. What modifications to the existing barcode-based checkout system would be needed?



**Figure 7.8** The original egg-grading algorithm.

- 4 The algorithm shown in Figure 7.8 is used to grade eggs as 'small' (45 g to 55 g), 'medium' (56 g to 65 g) or 'large' (over 65 g). Eggs that are under 45 g are supposed to be rejected, but it has been found that 45 g eggs are also being rejected. Modify the algorithm so that it works correctly. Code your algorithm in a suitable language and then test the program.
- 5 Because of a glut of eggs on the market an egg farmer now wants to sell grown chickens. Modify your program in question 4 so that it grades chickens as 'small' (< 1.2 kg), 'medium' (from 1.2 kg to 1.7 kg) and 'large' (over 1.7 kg).

# Social and ethical implications

As we discussed in Chapter 1, software is a product that results from a creative process. Modification of software must be performed within legal and ethical bounds. We cannot take an original program, modify it in some small way and release it as our own product.

If a software application has been bought off the shelf, it may be customised for the user's particular application as long as the code is not changed. Modification of the code of such an application breaks the licence agreement and renders the modifier liable to prosecution for a breach of copyright.

Within a program development system, the licence agreement will specify that the supplied library routines may be incorporated in saleable software, but the same routines may not be modified or sold separately. Licence agreements also specify that the supplied library routines may not be 'backwards engineered' to find out how they work or to modify them, as this constitutes plagiarism.

## Plagiarism

**Plagiarism** originally referred to the copying of text from a book and claiming that it was original. For example, a student copying out an article from an encyclopaedia word for word, and submitting it to the teacher as an original assignment, is guilty of plagiarism. Plagiarism is more than just a breach of copyright, as actual ownership of the article is being claimed. A breach of copyright means that the intellectual property is being used illegally, but the authorship of the work is undisputed. The difference between the two can be illustrated by using this book as an example. If the whole book were to be photocopied by a student, that student would be breaching the copyright of the book. However if a student were to copy out all the text from this book and took it to another publisher claiming that he or she wrote it, the student would be guilty of plagiarism.

Within the context of software development and modification, plagiarism includes the use of code within an application without acknowledging its source and the reverse engineering of code to determine the algorithm which is then recorded.

## Exercise 7.2

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

When code is \_\_\_\_\_, we must make sure that we abide by any \_\_\_\_\_ agreements. A \_\_\_\_\_ involves using \_\_\_\_\_ illegally, whereas \_\_\_\_\_ is claiming that it was created by you. \_\_\_\_\_ also includes the reverse \_\_\_\_\_ of code to find how it works so that the \_\_\_\_\_ can be written in another \_\_\_\_\_.

- 2 Explain why a student cannot submit a program seen in a textbook as his or her work. Is this an example of breach of copyright or plagiarism? Justify your answer.
- 3 Examine the licence agreement of a program development system to discover the references to library code and supplied subprograms. Explain, in your own words, how you are allowed to use these samples of code.

# Features in source code that improve its maintainability

## Documentation within the code

When creating program code, the programmer has a responsibility to ensure that it can be easily modified and updated. One of the easiest ways of accomplishing this is to employ internal documentation in the form of comments (or remarks) and choose meaningful identifier names. The use of procedures (subprograms) to carry out repeated tasks and the use of identifiers to represent constants in preference to using actual values also assist with maintenance. Declaration of constant identifiers at the beginning of a program may seem to be a waste of time, but, if it becomes necessary to change the value of one of the constants after the program has been written, one occurrence only needs to be changed instead of a number throughout the program. The use of clearly defined subprograms within the code further assists a maintenance programmer in updating the program.

Some programming languages allow for constants to be given identifiers (names) that can be used within a program. The value of the constant is usually declared at the beginning of the code and, unlike variables, that value cannot be changed as part of the processing. For those languages that don't have this feature, we can assign the value of the constant to a variable, then not change its value during processing. As mentioned before, the purpose of this activity is to make the code easier to modify. For example, if we have a program that calculates the circumference and area of a circle, we may want to use 3.142 for the value of pi. If at a later date we need to use a more accurate value, such as 3.141 59, the pi value needs to be changed only once, where the constant is declared. We don't have to hunt through the program to find all the places where 3.142 occurs. The following Pascal code illustrates the way in which the constant is used. This is a very simple example. If the constant pi had occurred twenty times in the program instead of twice, the time saved in changing it would obviously be much greater.

```
PROGRAM circle_calculations (input, output);
CONST
 pi = 3.142;
VAR
 radius, circumference, area : real;
BEGIN
 write('Please type in the radius of the circle');
 readln(radius);
 circumference := 2*pi*radius;
 area := pi*radius*radius;
 writeln('The area of your circle is ',area);
 writeln('The circumference is ', circumference)
END
```

The careful choice of identifiers can also assist in the processes of development and maintenance. The use of meaningful names for variables and subprograms is known as **intrinsic documentation**; that is, the documentation is built into the code. For example, it is much easier to determine what is meant by the line of code `balance := balance - withdrawal` than by a statement such as `x := x-y` when written in a program to process withdrawals from bank accounts. The second statement will, with the same data as the first, produce the same result; however, its meaning may be lost to all but the programmer, and so maintenance could be a problem.

Most of the languages from the second generation and above allow the programmer to insert text which is able to be ignored by the compiler or interpreter. This facility has been included in the language to allow a programmer to make notes (**comments** or **remarks**) inside the program code. The purpose of this type of internal documentation is to provide a means for the programmer to state the purpose of a section of code for later reference, for example in the debugging or maintenance stages of the code. Internal documentation is also very important if a team of programmers is working on the one problem, as it lets all team members know the processes that are being carried out in other modules. The benefit for a maintenance programmer is that, although the original creator of the code may not be available, the purpose of the code section is clearly spelled out.

The area and circumference program above illustrates the use of intrinsic documentation. If the code is read, the purpose of the program and the processes involved are clear without the need for further explanation.

## Presentation of a coded solution

The manner in which a coded solution is presented can have a large bearing on the ease of maintenance. Factors that affect the presentation of the code are the use of standard control structures, a clear and uncluttered mainline and the allocation of one logical task to each subroutine. These can be shown in the code by using levels of indentation to show the relationship between the various portions of code. Indentation is ignored by the compiler or interpreter, these features being included solely to help us understand the structure of the program.

For example, compare the legibility of the following two samples of identical Pascal code. The first sample has not been formatted to any standards; the second has been coded with one instruction per line and indentation has been used to show the statement levels.

### Example 2

```
program menu_test (input, output); var choice: char;
procedure Add_record; begin writeln('Add record stub') end;
procedure Delete_record; begin writeln('Delete record stub') end;
procedure Sort_records; begin writeln('Sort records stub') end;
procedure Invalid_input; begin writeln('Invalid choice stub') end;
begin writeln('Please press the letter key corresponding to your choice');
writeln('A to ADD a record'); writeln('D to DELETE a record');
writeln('S to SORT the records'); writeln('E to END this session');
readln(choice); while (choice <> 'E') and (choice <> 'e')
do begin if choice in ['A', 'a', 'D', 'd', 'S', 's'] then
{this statement ensures that other characters are excluded .. otherwise}
case choice of 'A', 'a': Add_record; 'D', 'd': Delete_record;
'S', 's': Sort_records end {end of case statement} else Invalid_input;
writeln('Please press the letter key corresponding to your choice');
writeln('A to ADD a record'); writeln('D to DELETE a record');
writeln('S to SORT the records'); writeln('E to END this session *');
readln(choice); end end.
```

### Example 3

```
PROGRAM menu_test (input, output);
VAR
 choice : char;
PROCEDURE Add_record;
 BEGIN
 writeln('Add record stub')
 END;
PROCEDURE Delete_record;
 BEGIN
 writeln('Delete record stub')
 END;
PROCEDURE Sort_records;
 BEGIN
 writeln('Sort records stub')
 END;
PROCEDURE Invalid_input;
 BEGIN
 writeln('Invalid choice stub')
 END;
BEGIN
 writeln('Please press the letter key corresponding to your choice');
 writeln('A to ADD a record');
 writeln('D to DELETE a record');
 writeln('S to SORT the records');
 writeln('E to END this session');
 readln(choice);
 WHILE (choice <> 'E') AND (choice <> 'e') DO
 BEGIN
 IF choice IN ['A', 'a', 'D', 'd', 'S', 's'] {this statement
ensures that other characters are excluded .. otherwise}
 THEN CASE choice OF
 'A', 'a' : Add_record;
 'D', 'd' : Delete_record;
 'S', 's' : Sort_records
 END
 {end of case statement}
 ELSE Invalid_input;
 writeln('Please press the letter key corresponding to your choice');
 writeln('A to ADD a record');
 writeln('D to DELETE a record');
 writeln('S to SORT the records');
 writeln('E to END this session');
 readln(choice);
 END
 {end of while statement}
END.
```



The second sample program is easier to follow not only because of the placement of single instructions as separate entries and the use of indentation to show the logical blocks such as loops and decisions, but also because the use of uppercase letters for the Pascal reserved words together with the separation, wherever possible, of the comments allows the structure of the program to be immediately visible. Thus any modification of the code can be accomplished efficiently.

## Exercise 7.3

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

The use of carefully chosen \_\_\_\_\_ names an help in \_\_\_\_\_ a program. This type of documentation is known as \_\_\_\_\_ documentation as it is \_\_\_\_\_ into the program code. Other forms of \_\_\_\_\_ documentation consist of \_\_\_\_\_ or remarks that can be included with the \_\_\_\_\_ code. Well written code will \_\_\_\_\_ contain \_\_\_\_\_ to explain what is occurring as the code should be almost self-\_\_\_\_\_.

- 2 Explain, in your own words, why it is important to include documentation in the coded solution of the problem.
- 3 Copy and rewrite the following Pascal code so that its structure is much more clear than it is here. There are no syntax errors in the code; it is only the presentation that is wrong:

```
PROGRAM picture_framing;CONST frame_rate=15.65;glass_rate=7.55;
VAR picture_length, picture_width, frame_length, glass_area,
cost:real;BEGIN writeln('What is the length of the picture?');
readln(picture_length);writeln('What is the width of the
picture?');readln(picture_width);frame_length:=2*(picture_length+
picture_width);glass_area:= picture_length* picture_width;
cost:=frame_length* frame_rate+glass_area* glass_rate;writeln
('The picture will use',frame_length, 'metres of framing');
writeln('The picture will use',glass_area, 'square metres of
glass');writeln('The total cost will be $',cost)END.
```

- 4 Create an algorithm for the following program, and code it in an appropriate language. Use internal documentation to make your program easy to maintain. A parking station charges \$2 per half hour or part thereof up to a maximum of \$10 for a day. Write a program that will calculate the cost of parking for each customer whose time is entered. The program is to keep asking for the next customer's time until terminated by a time of -24 hours. The program is also to calculate and display the total day's takings.

# Interpretation

The program code is not the only source of information available about the processes in a segment of code. We can also gain an insight into the workings of the segment by looking at other documentation associated with the application. This documentation consists of product documentation such as user manuals and process documentation such as algorithms and system descriptions. It is also possible to take the source code and convert it back into an algorithm in order to perform a modification.

## Reading original documentation to understand the code

Original documentation can be very useful when it comes to understanding the workings of software. Well-written documentation can often provide as much information as examining the code itself. As we have seen, there are two types of documentation associated with software—product documentation and process documentation. Both are a valuable source of information about the software and its operation.

**Product documentation** such as user manuals and even tutorials can help us to understand both the processes that are being carried out by the software and the ways in which these processes are structured. If we understand the processes from a user's point of view, we are better placed to make appropriate changes to the operation of the software.

**Process documentation** consists of all the documents produced during development of the software. The process documents would include systems descriptions, the original specifications, algorithm descriptions, memos, and any notes made by the programming team. This documentation is most important as it describes the development process. Looking through these documents gives us an insight into the ways in which the algorithms were designed. We can also glean a lot of information about problems that were encountered by the development team.

For example, the Cumfy Coaches software development process was accompanied by a number of documents. When the company decided to operate a daily coach service to Sydney, the system needed to be modified, and these documents were read by the modification team. This allowed the maintenance team to understand the ways in which the data was organised and allowed them to choose a similar format for the new data. By using the existing screen layouts they were also able to integrate the new modules into the existing software so that it looked as if the whole program was created at the same time. The benefit of this for the users was that they felt familiar with the new modules as soon as they saw them, even though they had not previously been used. An added benefit in this case was that, since a lot of the processing required by the new application was similar to that of the existing software, some of these modules could be used for both tasks. The combination of these factors, together with others, meant that the new requirements were met by the software much more quickly and cheaply than they would have been without this approach.

## Reading original algorithms to identify inputs, processes and outputs

As noted, the original algorithm descriptions form part of the process documentation formed during the development stage of the software lifecycle. These algorithm descriptions show exactly how the software functions. They are a valuable tool during maintenance as an existing algorithm may only need to be

modified slightly in order to meet some of the new requirements. Even if the algorithm descriptions don't lend themselves to modification, the algorithms may be able to be used to pinpoint the source of problems.

In reading the algorithms we must be able to follow the way in which they work. This means that we must be able to find the inputs and determine the variables used, the processes that take place and the outputs from the algorithm. The structure of the algorithm will help us identify these key elements.

**Inputs** may come from the user, a file or from some peripheral and, in most cases will be fairly obvious. An examination of the algorithm will identify the variables that were used. The data structure represented by each of these variables should also be apparent.

Following the **processes** is a much more difficult task. We need to trace through the algorithm to find out what goes on. A big help is to obtain the set of original test data and work through the algorithm with that data. By treating the exploration of the algorithm as a desk check, we are forced to examine each step carefully and therefore think about, and become more familiar with, the processes as a whole.

**Outputs** are usually fairly readily identified, but we should be on the lookout for any outputs to storage such as disk files.

In the sample algorithm in Figure 7.9, the first operation can be seen as reading in the whole customer data file from disk. A customer number is then input and tested to see if it has been allocated. If there is a customer record associated with the given customer number, that record is located and displayed; otherwise a new record is created. The current record is then displayed and updated. After the change, the data file is stored on disk and the option of continuing to process a new customer or closing off the process is given.

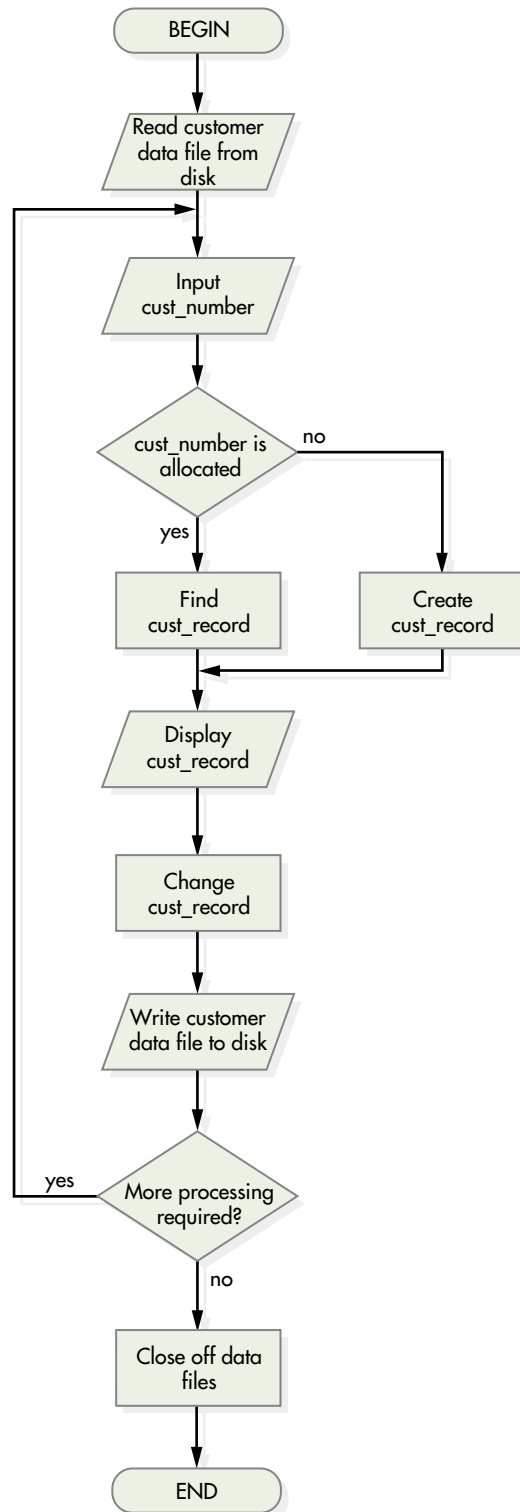


Figure 7.9 Sample algorithm 1.

Within this structure we can identify the customer data file as a file of records. Each of the customer records contains a number of fields, which can be obtained by looking at the subprograms which process the `cust_record` data item. However, we can identify that `cust_number` is one of the fields required. We can also identify the use of a variable in the decision to continue processing.

Although this algorithm is written as the combination of a number of large processing modules, the use of various variables is quite evident. If we analyse the subprograms, we can find even more detail about the processing. For example, the `find_cust_record` subprogram is given in the following pseudocode representation.

*Sample algorithm 2 pseudocode*

```
BEGIN find cust_record
 set flag to 0
 set index to 0
 WHILE (flag = 0) AND (index < end)
 set index to index + 1
 IF cust_record[index].number = cust_number THEN
 set flag to index
 ENDIF
 ENDWHILE
 IF flag <> 0 THEN
 display cust_record[index].name
 display cust_record[index].address
 display cust_record[index].order
 display cust_record[index].order_cost
 display cust_record[index].balance_owing
 display cust_record[index].credit_rating
 ELSE
 display 'The customer record is not in the file'
 display 'The file is damaged'
 display 'Contact the systems administrator'
 ENDIF
END
```

In this subprogram, we can identify the WHILE loop that passes through each record in the file comparing the customer number with the wanted one. More importantly, the data stored within `cust_record` is stated. The record consists of fields for name, address, the order, the cost of the order, the balance that the customer owes and the customer's credit rating. We should also not forget that the customer's number is also stored as one of the fields.

We now have a good idea of the data structures used in the algorithm, together with the processes that take place. Once all the subprograms within sample algorithm 1 have been analysed in the same way, we will be in a position to modify the solution to meet the new demands.

### Creating algorithms for source code when they are not available

Since an algorithm is essential when writing code, it is sometimes necessary to convert an existing coded solution into an algorithm in order to modify it later. This is a reasonably easy task to accomplish as long as it is done in a systematic

way. Code which has been written without proper planning, however, may require some work before the underlying algorithm becomes evident. This is especially true of programs that have been written 'at the keyboard' with little or no structure. This type of code is often called **spaghetti code** as following the execution paths through the code is very much like trying to unravel the strands of spaghetti on a plate.

### Example 4

This code is written in Algol.

```
begin comment Example program in Algol;
 real a, b, c, x, y;
 read (a,b,c,x);
 y := c + x * (b + a * x);
 print (y);
end
```

Consider the Algol code in Example 4. We can first identify the variables used as  $a$ ,  $b$ ,  $c$ ,  $x$  and  $y$  as they are declared as real numbers. As we follow the instructions, we find that, in the third line of code, three of the variable values,  $a$ ,  $b$ ,  $c$  and  $x$  are input. These values are used to evaluate an expression in the next line which is output as  $y$ . We are now in a position to translate this simple program into an algorithm (Figure 7.10).

*Pseudocode*

```
BEGIN
 set a to user input
 set b to user input
 set c to user input
 set x to user input
 set y to c + x(b + ax)
 display y
END
```

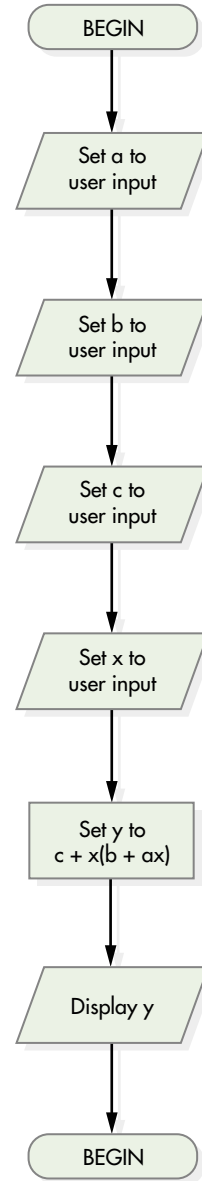


Figure 7.10

## Example 5

This code is written in a version of Fortran.

```
DIMENSION X(500)
READ (1,5) N
5 FORMAT (I5)
 READ(1,10) (X(I),I=1,N)
10 FORMAT (8F10.0)
 J=2
 XMAX = X(1)
 XMIN = X(1)
 SUM = X(1)
 SUMSQ = X(1) * X(1)
15 IF (X(J).GT.XMAX) XMAX = X(J)
 IF (X(J).LT.XMIN) XMIN = X(J)
 SUM = SUM + X(J)
 SUMSQ = SUMSQ + X(J) * X(J)
 J = J + 1
 IF (J.LE.N) GO TO 15
 RN = N
 XMEAN = SUM / RN
 STANDEV = SQRT ((SUMSQ - (1.0/RN)*(SUM * SUM))/(RN - 1.0))
 WRITE (1,20) XMIN , XMAX
20 FORMAT(1H1,7X,19HTHE RANGE IS FROM, F12.4, 4H TO, F12.4)
 WRITE (1,25) XMEAN , STANDEV
25 FORMAT(10HWITH MEAN, F12.4, 21H STANDARD DEVIATION, F12.4)
 STOP
 END
 EXTERNAL CDI,PRINT
 CALL DEBUG
```

Consider the code in Example 5. Since this code is written in Fortran, the variables have not been chosen to represent the names of the processed data items. The major reason for this was that the language used the initial letter of a variable to determine its data type. (Variables starting with the letters *I, J, K, L, M* and *N* were all classed as integer variables in Fortran.) The first variable we reach is the array *X* which has its size set to a maximum of 500 elements in the first line. As we pass to the second line, the variable *N* is input. The `FORMAT` statement 5 indicates to the computer the form that the input data will take. The value of *N* read at this stage will determine the number of data items that are put into the array on this run. Following the `FORMAT` statement we come to a single line which reads each of the values from  $X(1)$  through to  $X(N)$ .

Progressing through to the processing section of the program, we find that *J, XMAX, XMIN, SUM* and *SUMSQ* are introduced and initialised. The next four lines (from line 15) in order take element *J*, determine whether it is

greater than the maximum, determine whether it is less than the minimum, add it to the sum of the scores and add its square to the sum of the squares. When  $J$  is incremented, it is tested to see if it is less than the number of scores,  $N$ , and if it is, then the steps are repeated. Following this repetition, the mean and standard deviation ( $XMEAN$  and  $STANDEV$ ) are calculated and the required results are output. The statement in which  $RN$  is given the value of  $N$  had to be performed as Fortran would not allow operations which involved a mixture of real and integer variables. The end of the processing within the program is at the STOP statement. The END statement tells the compiler that it has reached the end of the code to be converted and the last two statements are also messages to the compiler to obtain some routines from the library.

We can now convert the code into an algorithm. This has been done below. In the algorithm descriptions, we will keep the original variable names to make the structure more closely identifiable with the Fortran code. These names are not the best ones for use, as some of them don't give any indication of the purpose of the data item.

### *Pseudocode*

#### **BEGIN**

set  $N$  to user input

set  $I$  to 1

**WHILE**  $I \leq N$

set  $X(I)$  to user input

set  $I$  to  $I + 1$

**ENDWHILE**

set  $J$  to 2

set  $XMAX$  to  $X(1)$

set  $XMIN$  to  $X(1)$

set  $SUM$  to  $X(1)$

set  $SUMSQ$  to  $X(1) * X(1)$

**REPEAT**

**IF**  $X(J) > XMAX$  **THEN**

set  $XMAX$  to  $X(J)$

**ENDIF**

**IF**  $X(J) < XMIN$  **THEN**

set  $XMIN$  to  $X(J)$

**ENDIF**

set  $SUM$  to  $SUM + X(J)$

set  $SUMSQ$  to  $SUMSQ + X(J) * X(J)$

set  $J$  to  $J + 1$

**UNTIL**  $J > N$

set  $RN$  to  $N$

set  $XMEAN$  to  $SUM / RN$

set  $STANDEV$  to the square root of  $((SUMSQ - (1/RN)*(SUM*SUM))/(RN - 1))$

output  $XMIN$ ,  $XMAX$ ,  $XMEAN$  and  $STANDEV$

**END**

## Exercise 7.4

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

A number of documents, including \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_, can be used to obtain information about the process in a . By reading \_\_\_\_\_ documents produced during the \_\_\_\_\_ of a program we can better \_\_\_\_\_ how the program works. The original \_\_\_\_\_ tell us how the software \_\_\_\_\_ and is, therefore, extremely useful during \_\_\_\_\_. When the \_\_\_\_\_ of a program are not available, we can create them from the source code in order to help us to maintain the program.

- 2 Describe the differences between product documentation and process documentation. Explain how these documents can help us in the process of program maintenance.
- 3 Examine the following pseudocode algorithm. List the inputs and outputs together with the variables used. Describe the processing that takes place.

```
BEGIN
 set minimum_total to 150
 set total to 0
 WHILE total <= minimum_total
 set amount to user input
 set total to total + amount
 display total
 ENDWHILE
 display 'You have now completed the task'
END
```

- 4 Modify the algorithm in question 3 so that it displays the amount still remaining after the input value has been counted.
- 5 Construct the algorithms represented by the following sections of code. Describe the purpose of each of these program segments.

```
a PROGRAM part_a (input,output);
 VAR
 count : integer;
 price, total_price : real;
 BEGIN
 count := 0;
 total_price := 0;
 read(price);
 REPEAT
 count:=count + 1;
 writeln(count, price);
 total_price := total_price + price;
 read(price)
 UNTIL price=0;
 writeln('The total price is', total_price)
 END
```



```

b 1 S = 0
 2 AREA = 3.1415927*S*S
 3 WRITE(1,4) S,AREA
 4 FORMAT (1X,F5.1,F10.5)
 5 S = S + 0.1
 6 IF (S.LE.10.0) GO TO 2
 7 STOP
c 10 DIM Z(10)
 20 LET I = 9
 30 FOR J = 1 TO I STEP 1
 40 LET K = J + 1
 50 FOR L = 10 TO K STEP -1
 60 IF Z(L) > Z(J) THEN GOTO 100
 70 LET T = Z(L)
 80 LET Z(L) = Z(J)
 90 LET Z(J) = T
 100 NEXT L
 110 NEXT J
 120 END

```

---

## Documentation

As already seen, documentation is a very valuable tool in the maintenance phase. We can gain an insight into the structure of the program by looking at the way in which the program is structured and the way in which the processing involves the variables. This structure is most evident from the algorithm descriptions created during the development stage.

### Using supplied documentation to identify the control structures and explain how variables have been used

It was noted in the previous section that stepping through an algorithm can help us become more familiar with the processes. A further benefit is that it allows us to identify the structures that form the algorithm. We need to study these structures before we begin the modification process; during the modification process some of the algorithm's structures may be changed, rearranged, completely rewritten or even eliminated altogether from the algorithm.

Data items being processed by an algorithm are represented by the variables used. As we follow a data item through the algorithm, the manner in which it is used becomes clearer. Performing a desk check on the algorithm using test data items from the development process allows us to more clearly identify both the variables used and the ways in which they have been used.

Earlier we looked at an algorithm that uses a customer number to locate and work on a record within a file. In the flowchart in Figure 7.9, we can see a repetition and a binary selection. The loop is a post-test loop that requires the user to decide whether there are any more data items to be processed. The binary selection is used to create a new customer record if the customer number has already been allocated.

The variable `cust_record` is the one which is used throughout the algorithm to determine the flow of control. As we saw, the fields within the record selected by the customer number are also involved in the algorithm.

You might like to create some data items and work through the algorithm as written. This process will help you better understand how it works.

## Exercise 7.5

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.

By studying an \_\_\_\_\_ we can become familiar with the processes being \_\_\_\_\_ We also follow a \_\_\_\_\_ item as it is being processed by looking for the \_\_\_\_\_ that has been used to represent it. \_\_\_\_\_ check also helps us understand the being carried out, as we are forced to work through the \_\_\_\_\_ step by step. Once we know the variable we are in a position to be able to \_\_\_\_\_ the algorithm.

- 2 Explain why we need to identify the variables before beginning to modify a solution.
- 3 Describe the processes that are carried out in the following algorithm. In your description, name all the variables used and identify how they are used.

```
BEGIN
 set number to user input
 set factorial to 1
 WHILE number > 0
 set factorial to factorial * number
 set number to number - 1
 ENDWHILE
 display factorial
END
```

- 4 Obtain the source code for a program in either BASIC or Pascal. Create an algorithm from the code and then code the program in a language of your choice.

## Team Activity

In Chapter 4 you designed a program to keep the rainfall records for the past thirty days. Use this design to create a fully working program. As a team, evaluate the performance of his program and suggest ways in which it could be improved. Discuss

these improvements and implement the improvement that the team thinks is most important. During this process keep a diary of the development and modification, detailing the discussions that took place and the results of those discussions.



# Chapter summary

- Maintenance coding of a program may be required for a number of reasons: to remove a bug, to improve the efficiency, to change the way a program works, to allow the program to cope with changed data, to meet new requirements of the organisation or to comply with government regulations.
- Maintenance can be needed if hardware or software have been changed.
- Changing user requirements may lead to maintenance.
- One of the user items that can need maintenance is the user interface.
- A generic user interface can be created which can be modified for many different applications.
- Expansion of a system to cope with a greater number of data items can mean that the software needs to be modified.
- Poorly implemented code, such as the use of software patches, can be improved during maintenance.
- Software may need to be customised if it has been bought off the shelf.
- Software modification has to be carried out within legal and ethical bounds.
- Plagiarism is the use of intellectual property without acknowledging its source.
- Documentation within the code can help with maintenance.
- Intrinsic documentation is built into the code by the use of appropriate identifiers for constants, variables and subprograms.
- Internal documentation involves the use of remarks or comments within the code to explain what processes are being carried out.
- The presentation of the coded solution can also help in the modification. Indents will help identify the major sections and groupings within the code.
- Original documentation can be used to help understand the software that is to be modified.
- The original documentation consists of product and process documentation.
- Product documentation consists of user materials such as manuals.
- Process documentation consists of documents such as algorithm descriptions and assists in system maintenance.
- The original algorithms tell us how the program works and how data has been represented.
- Documentation can tell us about the control structures used and the ways in which data has been processed.

# chapter 8

## → *Developing software solutions*

### Outcomes

- describes and uses appropriate data types (P 1.2)
- describes the interactions between the elements of a computer system (P 1.3)
- identifies the issues relating to the use of software solutions (P 3.1)
- analyses a given problem in order to generate a computer-based solution (P 4.1)
- investigates a structured approach in the design and implementation of a software solution (P 4.2)
- uses a variety of development approaches to generate software solutions and distinguishes between these approaches (P 4.3)
- uses and justifies the need for appropriate project management techniques (P 5.1)
- uses and develops documentation to communicate software solutions to others (P 5.2)
- describes the role of personnel involved in software development (P 6.1)
- communicates with appropriate personnel during the software development process (P 6.2)
- designs and constructs software solutions with appropriate interfaces (P 6.3)

### Students learn about:

#### Implementing a project

- the steps in implementing projects, including:
  - defining the problem
  - understanding the problem
  - identification of inputs, processes and outputs to be applied to the problem
- planning
  - identification of a suitable development approach
  - design of appropriate algorithms
  - determination of appropriate data structures
  - identification of relevant subroutines
  - the design of test data and expected output

- the desk check of algorithms
- identification of existing code that can be used
- building
  - implementation of the solution in an appropriate language
  - testing of the solution using test data
  - documenting the solution, including algorithms, tutorials, test data and expected outputs, data dictionaries
- checking
  - testing of the solution using test data
  - evaluation of the completed solutions
- modifying
  - changing the solution to meet the specifications

#### Project management techniques

- identification of tasks
- identification of techniques to assist project management, including:
  - Gantt charts
  - logbooks
  - identification of subgoals
- allocation of resources
- identification of major milestones and stumbling blocks
- regular backup
- regular reporting
- response to difficulties
- evaluation

#### Project documentation

- algorithms
- manuals
- data dictionaries
- CASE tools
- Gantt charts
- system documentation
- diaries

#### Social and ethical issues related to project work

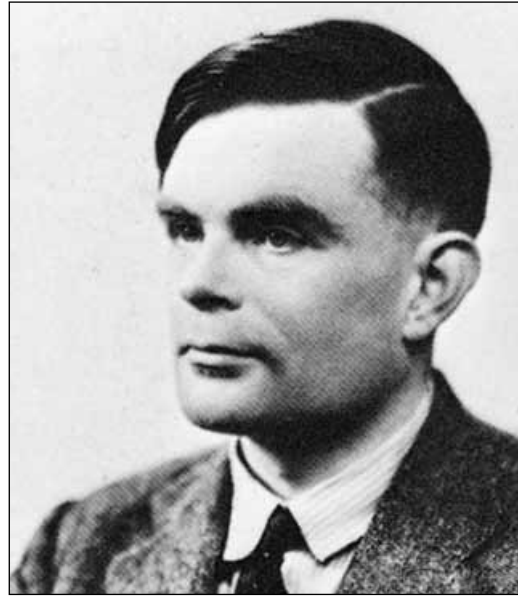
- ease of use
- accessibility of technical language
- ergonomics
- gender bias
- copyright

## Students learn to:

- design and implement a software solution to a selected problem using project implementation steps
- use Gantt charts and logbooks
- devise a management plan and use it when undertaking a software development project
- use appropriate application packages in creating documentation to support the development of a project
- prepare suitable documentation to accompany software solutions
- ensure that relevant social and ethical issues have been addressed
- evaluate the project in relation to the original understanding of the problem
- evaluate the quality of the solution

# Personal Profile—Alan Mathison Turing (1912–1954)

Alan Mathison Turing was born on 23 June 1912 in London. After attending secondary school, he enrolled at King's College, Cambridge where he studied mathematics. In 1936 he published a paper called 'On Computable Numbers' in which he described a machine that could theoretically solve any problem which could be solved by a machine. His so-called 'universal computing machine' later became known as a Turing Machine and is the basis for all current computers. The concept behind a Turing Machine is that the machine contains a control unit that can read and write to a tape. Computation in a Turing Machine consists of a number of steps executed by the control unit with symbols being written to the tape. The state of the machine at any instant determines the action to be performed by it.



In 1936 Alan was accepted as a graduate student at Princeton University, where he carried out his research under the supervision of Alonzo Church. His research resulted in the award of a doctorate in 1938 for a thesis entitled 'Systems of Logic Based on Ordinals'. The Church–Turing thesis states that there is no process describable by an algorithm that cannot be computed by a Turing Machine.

During the Second World War Alan worked on the top-secret Ultra project which cracked the German Enigma code. After the war Alan joined the National Physical Laboratory in London working on the project to design and build the Automatic Computing Engine (known as ACE). In 1948 he moved to Manchester University where the first running electronic stored-program computer was under construction. His work also included theories of artificial intelligence and applying mathematical theories to biology. In 1954 he was found dead from cyanide poisoning while he was conducting an experiment.

Turing will be best remembered as the man whose theories of computational machines made it possible to bring the electronic computing device from a dream to a reality.

# Implementing projects

In previous chapters the theory and practice of the software development cycle have been examined. In this chapter the theory is applied to a sample project.

## Case study

### Pizza Express

Pizza Express wishes to establish a Web presence. The company has heard that e-commerce is the way of the future and believes that sales could be greatly increased with a Web site. Pizza Express offers pizzas in three sizes: large, medium and small. There are also a dozen pre-defined pizzas on the menu and customers are allowed to customise their pizzas if they wish. From time to time there are

some special deals on offer—for example, two small pizzas, garlic bread and a 1250 mL soft drink for \$14.90. From Monday to Thursday nights they offer half-price pizzas for people who pick up their own orders. In addition to pizzas, Pizza Express also sells Italian dishes such as Spaghetti Bolognese and various salads. In summer they sell gelato.

As with any software solution we will be following five steps in implementing projects: defining the problem, then planning, building, checking and modifying the solution (see Figure 8.1).

### Defining the problem

Defining the problem is broken down into two subsets: understanding the problem and identifying the inputs, processes and outputs to be applied to the problem.

### Understanding the problem

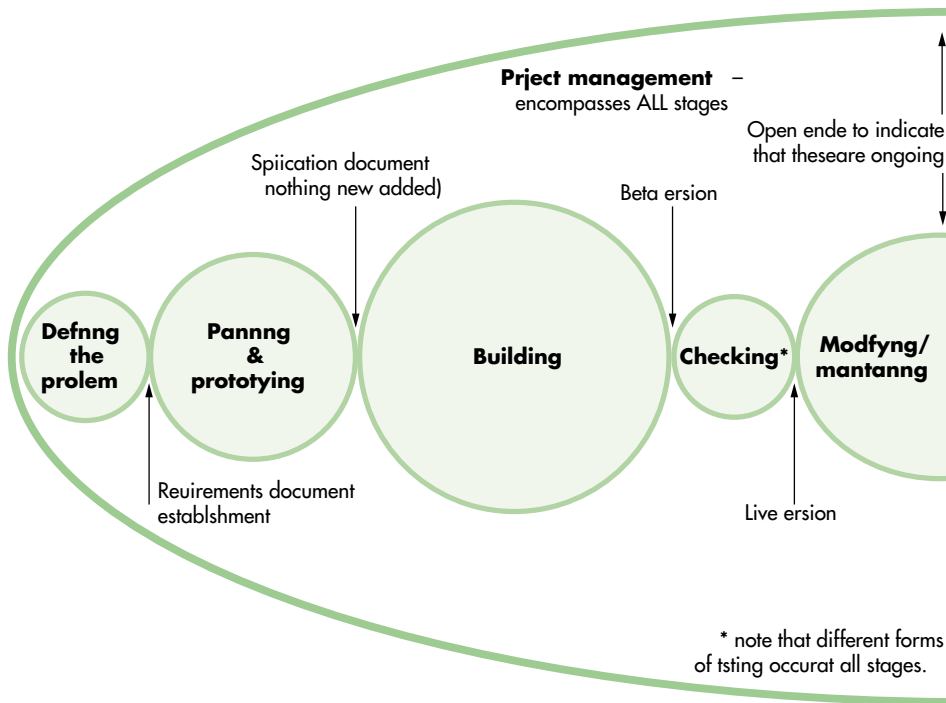
In the Pizza Express case study we need to carefully examine what the problem actually consists of. We need to break the problem down into its component parts to better understand its full scope.

The business is interested in a Web presence to advertise its services and sell its products. A solution to this problem will consist of a number of parts.

The first of these parts will be the construction of a **Web site** for the company. The site will need to contain a number of areas. These will include an area for general information about the company and its products and a secure area where customers can place their orders. The site will need to have a database of the company's products. This database will need to be able to be queried by customers and will also need to be updated as items are ordered and sold. Prices of customer orders have to be calculated and cooking and delivery schedules need to be generated. Confirmations of orders need to be created. Integration of this system with the existing system of telephone orders and orders placed by customers in person is also essential. As this is a very new undertaking, the owners of the company will need to be assured that their system will be secure and safe from malicious attack.

The second element of the solution is the development of a **database** of the company's products and stock levels. This database will need to be linked to the





**Figure 8.1** The stages of the software development cycle. The sizes of the circles indicate the relative amount of time devoted to each stage.

Web site and be searchable through a Web browser. As pizzas are ordered, the database needs to be updated and management are to be alerted when stocks need replenishing. Management should also be able to query the database to determine trends in orders so as to allow them to streamline their stock of toppings according to demand.

This is by no means an exhaustive breakdown of the problem. More details will emerge as we go further into the software development cycle. It is important to remember that each step is not a discrete entity; rather, each stage merges into the stages preceding and following it.

### **Identification of inputs, processes and outputs**

The next stage of the development cycle is to identify the inputs, processes and outputs to be applied to the problem. In our example these need to be broken down for a number of subsections of the problem. There will be inputs, processes and outputs for the Web site and others for the database, and there will possibly be some overlap of these.

Inputs identified at this stage are:

- customer details: name, address, phone number, credit card details
- order details: number of pizzas required, size, toppings, other items
- stock details: items, number of each on hand, number of each on back order.

Processes involved are:

- displaying Web pages as requested by user
- processing orders

- delivering orders
- updating the database
- updating Web pages
- calculating cost
- debiting credit card account
- ordering new stock
- performing stocktake
- cooking pizzas/food
- giving change
- generating reports on trends.

Outputs required are:

- Web pages
- completed orders
- pizzas/food
- payments
- reports.

We need to take this information and create an IPO chart. This will help us to see the interactions more clearly.

| Input          | Processes                                                                                         | Output                             |
|----------------|---------------------------------------------------------------------------------------------------|------------------------------------|
| Customer detls | Order form completed by customer                                                                  | Updated database entry             |
| Order detls    | Customer requirements entered<br>Order recorded and passed on to cooks<br>Stock database adjusted | Completed order ready for delivery |

**Table 8.1** IPO chart.

Other methods used to help understand the problems include dataflow diagrams (see Figure 8.2) and system flowcharts (see Figure 8.3).

### Exercises

Exercises in this chapter should be carried out as a team activity as they focus on the software development process. The projects in Exercise 8.1 are only suggestions to give an idea of the level required. You may even decide to develop an application that you can use after finishing the course.

As you progress through this chapter, in each of the exercises the first question or questions will involve the development of the Pizza Express solution. Questions at the end of the set will relate to the development of your own project. You should use computer technology to produce as much of the documentation for your project as possible, ensuring that it is presented in a suitable form. It is suggested that you work through the exercises of this chapter doing the Pizza Express question(s) in each section first, then pass onto your own project after finishing these questions.

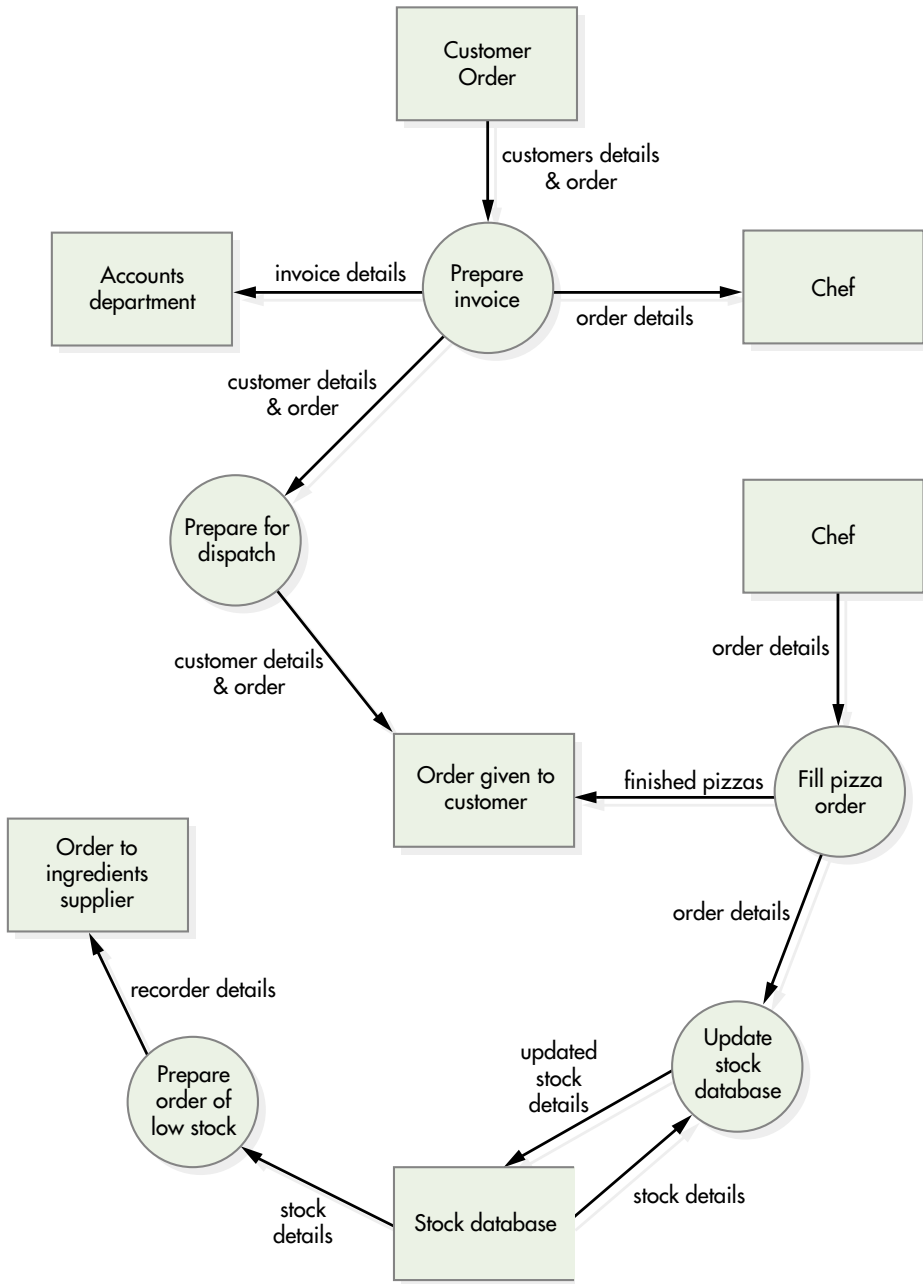
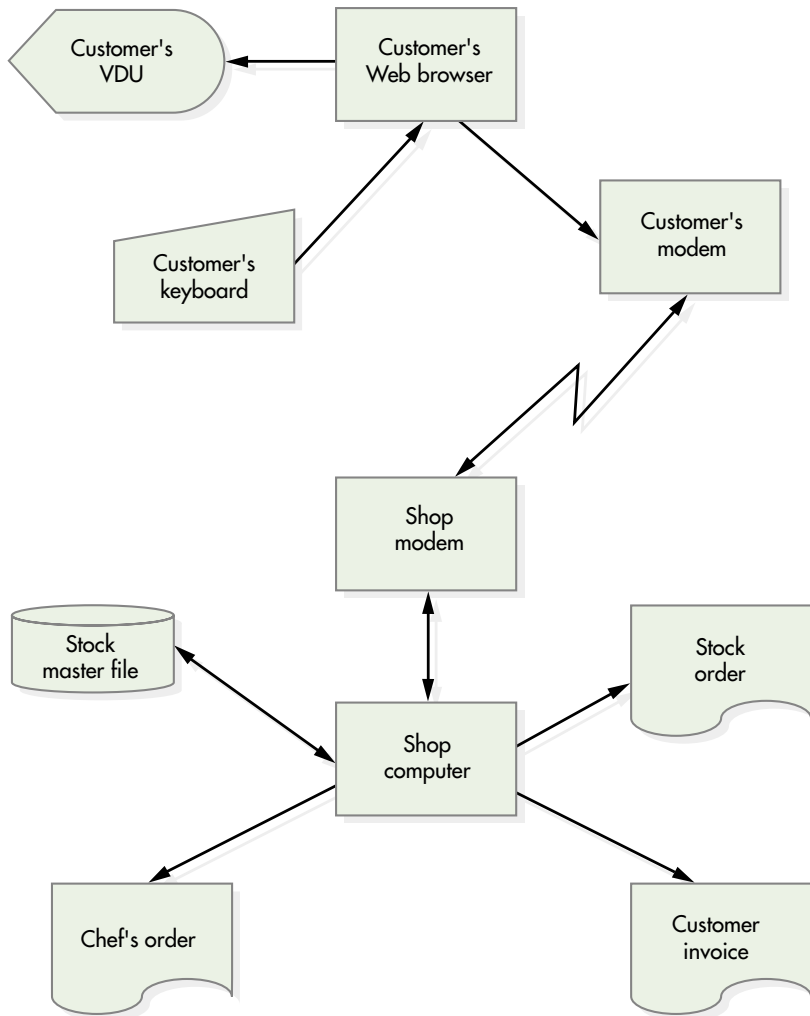


Figure 8.2 Dataflow diagram for Pizza Express.

## Exercise 8.1

- 1 You have been asked to design the Pizza Express Web page 'order form'.
  - a Describe the problem presented by this design brief.
  - b Construct an IPO chart for this aspect of the final solution.
  - c Draw a dataflow diagram representing the order form.



**Figure 8.3** System flowchart for Pizza Express.

- 2 Choose a project and produce the following documentation for your project: a statement of the problem, an IPO chart describing the whole project, a dataflow diagram representing the system, and a system flowchart.

Suitable projects include, but are not limited to:

- a Simulate a supermarket cash register. A barcode is to be input from the keyboard, and the product price and description are to be displayed on the screen. Once all items have been entered, the amount due should be displayed. An appropriate message must be displayed if the entered barcode does not have a product associated with it. The program should accept an amount tendered by the customer and calculate the change due. The product description file should contain at least twenty different products.
- b Produce a warehouse management program that is to keep a database of at least twenty items. The program is to keep track of the items entering and leaving the warehouse. When the quantity of an item falls below a certain level, the system is to create an order to the supplier.

- c Construct a simple library catalogue system that allows the user to add a book to the library and remove one from the library. The catalogue system should be able to be searched for books by a particular author and also by subject keywords for books on a particular topic. Your database of books should contain at least twenty entries.
  - d Create an interactive tour around your school. The tour should allow the visitor to visit points of interest and be given information about that place. The guide should use digitised images of the school to illustrate the various areas.
  - e Simulate the operation of a digital clock or watch with a reminder alarm. The clock should be able to display the 'time' in both 12- and 24-hour modes. The time should be able to be set, as should a reminder alarm. The display should simulate the seven-segment displays usually found on these devices.
- 

## Planning

Having identified and understood the problem, we now enter the planning stage of the software development cycle. This consists of a number of substages:

- identification of a suitable development approach
- design of appropriate algorithms
- determination of appropriate data structures
- identification of relevant subroutines
- the design of test data and expected output
- the desk check of algorithms
- identification of existing code that can be used.

### *Identification of a suitable development approach*

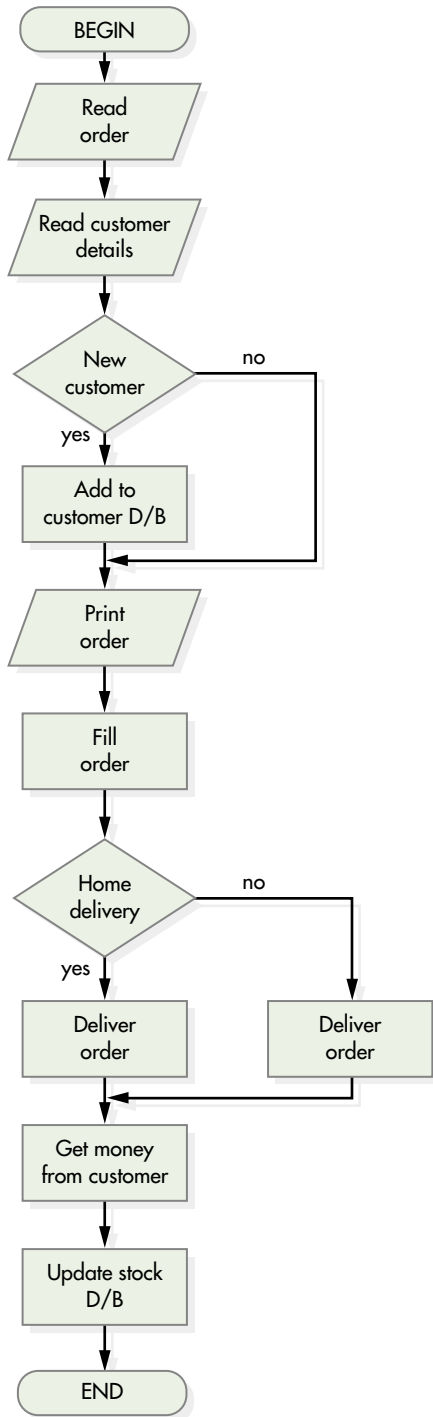
Software development approaches range from a completely unstructured or ad hoc approach to a highly structured approach. Ad hoc approaches are typically used to build a personal support system, such as a spreadsheet to keep the budget for a manager or a personal system to store addresses. A more structured approach is required in the development of larger and more complex programs. The program developed must be correct and must perform the tasks that are expected of it. In order for correctness to be achieved, the program needs to be defined exactly before construction can begin.

For the pizza-ordering system a structured approach is to be followed. As the program is to be a more or less trial project, the structured approach will not be strictly adhered to and there will at times be room for ad hoc processes.

### *Design of appropriate algorithms*

The next stage is to develop appropriate algorithms that describe the solution. Algorithms can be represented in a number of ways, including flowcharts and pseudocode. Flowcharts give the reader a graphical representation of the problem, while pseudocode is written in language that is close to standard English but is also very close to many programming languages such as Pascal. Many programmers prefer to use pseudocode because the transition to a programming language is easier.

The first algorithm that we will develop for our case study will describe the overall system. This algorithm will contain subroutines or modules that are



**Figure 8.4** The online ordering system expressed as a flowchart.

shown by underlining the appropriate sections. These modules will then be developed separately and brought together later in the development cycle.

#### *Pseudocode*

```

BEGIN On_line_pizza_orders
 read order
 read customer details
 IF new customer THEN
 add to customer database
 print order
 fill order
 IF home delivery THEN
 deliver order
 ELSE
 wait for customer to
 collect order
 get money from customer
 update stock database

```

END

This algorithm is an extremely simplified example. It will be modified and expanded a number of times throughout the software development cycle. Also, each of the underlined elements represent subprograms or modules which will need to be developed. For example, the get money from customer module will need to show how prices are calculated and will need to show how change is calculated and given if required.

### **Determination of appropriate data structures**

In writing the algorithms for our solution we need to consider the data structures that will be appropriate. Since we are creating a database for this project we will need to consider using structured data types rather than simple data types. Our database will consist of customer records, as well as records relating to stock. We will need to use a number of files that will have to be linked, so a relational database will be used. As the clients are an off-the-shelf database program (for example FileMaker Pro or Access) to create the database. The creators of the **database management system (DBMS)** will have made the bulk of the decisions about manipulation of the data structures. We will need to make decisions about the field data types as we create the database files.

### Identification of relevant subroutines

As mentioned earlier, the main problem can be broken down into a number of smaller problems. Each of these problems is then solved and the solutions are combined to solve the main problem. In the algorithm given earlier there are nine modules or subroutines. For example, deliver order is the subroutine to be designed to deliver the order to the customer. This subroutine can also be broken down into smaller parts, each of them a subroutine of the deliver order subroutine. One of these subroutines would be to collect the amount owing, which itself could also be broken into even smaller subroutines.

### The design of test data and expected output

Test data designed for the algorithms should test the limits of expected data. For example, a minimum price of \$12 is expected for an order that is to be home-delivered as this limit is specified in the advertising. Minimum order for home delivery \$12. There would be no upper limit to the amount ordered, but an order of, say, \$100 or above may be treated as a special order and processed using another algorithm. Some values that need to be entered to test the price calculation algorithm boundary values would be those above the upper limit (>\$100), those below the lower limit (<\$12) and those within the range specified, as well as the two boundary values \$12 and \$100.

| Data items—price | Expected output                | Reason for inclusion                                  |
|------------------|--------------------------------|-------------------------------------------------------|
| \$10             | Amount too all for home livery | To test for piceselow minimum boundary                |
| \$12             | Coninue with order             | To test fo the minimum boundary                       |
| \$15             | Coninue with order             | To test that progam continues—value within boundaries |
| \$100            | Uselarge order agorthm         | To test for the maximum boundary                      |
| \$150            | Uselarge order agorthm         | To test for prices above the maimum boundary          |

Table 8.2 Test data for home delivery price calculation.

### The desk check of algorithms

Once the algorithms have been written they need to be checked to ensure that they actually solve the problem that they have been created for. The process of desk checking is now used. This involves taking the test data that have been designed (as outlined in the previous section) and manually processing it using the algorithm. This process is usually carried out away from a computer, hence the term 'desk checking'.

Each of the algorithms will need to be tested in turn until are all tested. After testing, the algorithms will be able to be used as is or they will need to be modified. If an algorithm is modified, it must pass a desk check before it can be used.

### Identification of existing code that can be used

One advantage of the structured approach to software development is that problems are broken down into easily manageable parts. The smaller problems

are solved individually, with modules being created that may be reused in future programs. The developers need to identify any existing code that may be used for Pizza Express, thereby saving the company both time and money.

## Exercise 8.2

- 1 Describe the data items you need for the modules to read the order and customer details and choose an appropriate data structure to represent these items. Create test data and algorithms for the two modules read\_order and read\_customer\_details used in the online pizza orders program. Check that your algorithms perform the tasks required by the problem statement.
- 2 For your own project, determine the data structures required, create the test data, design any algorithms needed and check the algorithms. Identify any code or modules provided by your chosen language, or developed by you earlier in the course, that may be appropriate. Again use appropriate computer software, where available, to produce your documentation.

### Building the solution

Building the solution involves the translation of algorithms into an appropriate language. It is at this stage that the design specifications are handed over to the programmers. Programmers will code the solution using an appropriate language, test the solution using the existing test data, and document the solution by providing algorithms, tutorials, test data used, test outputs and data dictionaries.

### Implementation of the solution in an appropriate language

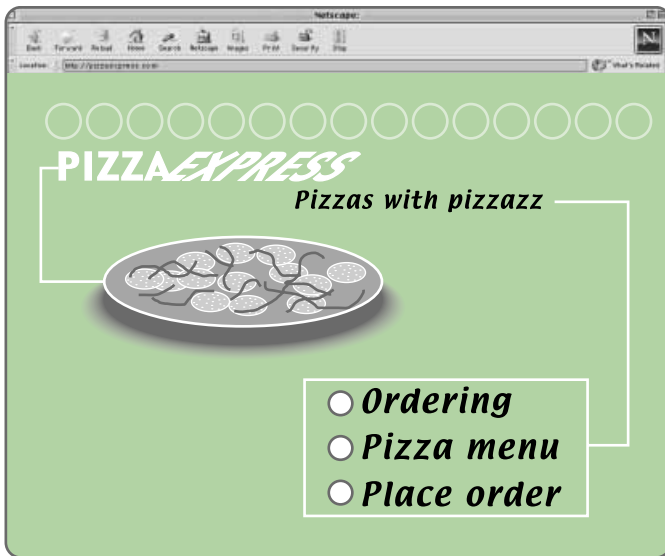
The solution for the pizza shop owners is to create a Web site and a database to allow customers to order pizzas via the Internet. For the Web site, the appropriate language will be **HTML (HyperText Markup Language)**. **CGI (Common Gateway Interface)** scripts will also need to be written. These can be written in most popular programming languages. Two of the most commonly used languages for CGI scripts, at present, are Perl and C++. As stated earlier, the database will be created using an off-the-shelf DBMS and so the only programming required for the database will be the construction of scripts that streamline searches and the generation of reports from the database.

### Testing the solution using test data

Once the solution has been coded it needs to be tested. This stage is similar to the desk-checking stage. The major difference is that we now test the coded program on computer rather than using pencil and paper and manually performing the testing. The Web pages have been created and the CGI scripts and database are in place on a test server (see figure 8.5). Programmers create test data that are to be used to test the relevant parts of the solution. This is to ensure that the solution does what is required of it.

Dummy data consisting of a number of test customers together with a variety of test orders is created. The test data is structured in such a way as to test the limits of the solution. As stated earlier, orders for home delivery will be a minimum of \$12 and a maximum of \$100. Test data includes orders that are both below and above this range, as well as falling within the range. Test data outside of the range should generate appropriate messages. Test data within the range





**Figure 8.5** The Web page has been created and it is now time to test it.

should result in orders being processed correctly. The boundary values (in this case \$12 and \$100) are also included to test whether the program performs the correct processing when these items are encountered. Other test data is used to test the database entries. For example, in the name field anything other than a text string should generate an error message.

It is a good practice at this stage to employ non-programmers or people who are not familiar with the system to perform some of the testing. These people will bring to light problems that may not have been anticipated by the programmers. The person who creates the program or indeed a person who creates the documentation often takes for granted things that a novice would not. Programmers often become 'blind' to certain faults in their programs for the simple reason that they have become accustomed to dealing with the problem in a certain way. There are often small things that need to be fixed but that are neglected. It is only when someone who has not been involved with the project comes to use the program that these small problems are identified.

### **Documenting the solution**

Keeping documentation is a crucial part of the software development process. Documentation is kept on each stage of the process and is updated as specifications change or as changes are made to any aspect of the project. In documenting the solution, programmers produce algorithms, tutorials, test data and expected outputs, and data dictionaries.

Algorithms help the programmers to see the logic of the solution before any coding begins. Algorithms are also a good reference point once the coding has commenced.

No program can be completely intuitive, so tutorials in using the program need to be written. Help pages on the Web site can form part of these tutorials. The tutorials will also instruct the owners in the use of the database and give instructions on how to extract useful statistics from the database.

Test data and expected outputs are created at both the desk-checking and program-testing stages. These are recorded together with the actual testing results. Data dictionaries describing the data used in the program are also created.

| Field          | Type                 | Size                | Range               | Example        |
|----------------|----------------------|---------------------|---------------------|----------------|
| Surname        | Sting                | Up to 20 characters | A to zzzzzz...      | Smith          |
| Frst_name      | Sting                | Up to 15 characters | A to zzzzzz...      | John           |
| Street_address | Sting                | Up to 35 characters | 0 to zzzzzz...      | 1 First Avenue |
| Suburb         | Sting                | Up to 20 characters | A to zzzzzFairfield |                |
| Postcode       | Sting                | 4 nueric characters | 0000 to 9999        | 2165           |
| Phone_noSting  | 10 nueric characters | 00000000            | 000 to 999999999    | 556            |

**Table 8.3** Pizza online data dictionary—customer details.

## Exercise 8.3

- 1 Using the algorithms you wrote for the two modules `read_order` and `read_customer_details`, implement your algorithm on a computer system using an appropriate language or development system. Check that your program performs the tasks required by the problem statement.
- 2 Create appropriate user documentation for the modules that you created in question 1. This may be online or hard copy, depending on what you think is appropriate for the application.
- 3 Code and test your own project algorithms. Create the appropriate documentation.

### Checking the solution

Checking is an indispensable stage of the software development cycle. In fact, checking is carried out throughout the entire cycle. Checking at this stage, however, may be broken into two parts: testing of the solution using test data and evaluation of the completed solution.

### Testing of the solution using test data

Once the modules have been integrated and the solution is considered to be complete, the solution needs to be tested. In the case of Pizza Express this involves testing the ordering system as it will appear to the customers. The Web site, complete with the database, is loaded onto a Web server and testers are given test data to enter. The test data is designed to test as many scenarios as possible, and will contain items that fall outside the acceptable range for inputs, inside the acceptable range, and on the boundaries. This is the final testing cycle before the Web site ‘goes live’ and is able to be accessed by the public and to take orders.

### Evaluation of the completed solution

Once the solution is delivered and implemented it will need to be constantly monitored and evaluated. This initial Web site is the pizza shop owner’s means of testing the market to see if there are any benefits in having an online presence. Close comparisons will be made between the online ordering system and traditional in-person or telephone systems. The Web site will not be economically viable if only a handful of people use it. The owners will need to gain a significant amount of increased business through the Web site to justify the expense.

Further evaluation will also tell the owners which areas of the online service are working well and which are not. Some adjustment to the system will probably be needed so that it works efficiently.

## Exercise 8.4

- 1 Evaluate the Pizza Express Web page for customer input by testing it on people outside your team. Document the evaluation and suggest ways in which the module can be improved.
- 2 Test your own project, using people outside your group to do the testing. Document the results and make suggestions as to how the product can be improved.

### Modifying the solution

Once the solution is in place it will be monitored closely. The owners will be keen to see if the new Web site leads to an increase in business. As the owners monitor the 'final' solution they may well decide that certain aspects do not look or work as they had anticipated. If this is the case, there will be a need to make modifications. For example, it may be necessary to change the database from a generic to a **purpose-built database**, that is, one designed specifically for this client.

### Changing the solution to meet the specifications

As the solution is being created there may be deviations from the original specifications. These deviations can occur for a variety of reasons. Depending on the nature of the deviations, some time may need to be spent on refining the solution so that it more closely resembles the original specifications.

## Exercise 8.5

- 1 Make any necessary changes to the modules you created for Pizza Express to reflect the information reported by your testers.
- 2 Make the modifications to your own project suggested by the evaluation. Don't forget to properly document these changes.

## Project management techniques

A project has a specific goal or objective that needs to be accomplished in a finite time with finite resources. The goal of the case study in this chapter is to have, at the end of a finite amount of time, a Web site that advertises the company's product and delivers its customers an online ordering service. The Web site must also be completed within a specified budget.

At first a task such as this seems daunting. Careful reading of what is required reveals a complex set of interrelated tasks. Assuming that we already have a project team with a project manager, then one of the initial stages of project management is to identify the various tasks involved and to break the project up accordingly. These tasks are then assigned to the appropriate team members.

## Identification of tasks

Let us look back at the original brief. We need to pick out the actual tasks from the written description. We do this by focusing on key phrases or words.

*Pizza Express wishes to establish a **Web presence**. The company has heard that **e-commerce** is the way of the future and believes that **sales** could be greatly **increased** with a Web site. Pizza Express offers pizzas in three sizes: large, medium and small. There are also a dozen pre-defined pizzas on the menu and customers are allowed to **customise their pizzas** if they wish. From time to time there are some **special deals** on offer—for example two small pizzas, garlic bread and a 1250 mL soft drink for \$14.90. From Monday to Thursday nights they offer **half-price pizzas** for people who **pick up their own orders**. In addition to pizzas, Pizza Express also sells **Italian dishes** such as Spaghetti Bolognese and various salads. In summer they sell gelato.*

Many of the highlighted terms are not actually tasks, but they may give us a clue as to what the possible tasks might be. For example, *Web presence* suggests a number of tasks: providing space on a Web server, creating Web pages, providing graphics and text. The term *e-commerce* suggests the provision of secure areas on the Web site to provide safe ordering for customers and the owners of the business. There are tasks that are not specified explicitly in the brief, such as the creation of the database and CGI scripts.

Many of the tasks will emerge as a result of team meetings in which the project team will examine the brief and brainstorm possible solutions. These tasks can be represented as a **task allocation table**, showing the tasks, the start and finish times and the resources.

| Task                       | Start  | Finish | Resource                |
|----------------------------|--------|--------|-------------------------|
| Requirements specification | Day 0  | Day 2  | Project team, client    |
| Analyse current system     | Day 2  | Day 5  | Systems analyst, client |
| Create algorithms          | Day 6  | Day 13 | Programmer/analyst      |
| Code program               | Day 15 | Day 22 | Programmer              |

Table 8.4 Task allocation table

## Exercise 8.6

- 1 Identify the tasks that need to be performed in the creation of the Pizza Express customer order page and create a task allocation table assigning each of the team members, or team subgroups, to one or more of the tasks.
- 2 Create a task allocation table to assist your team with the management of your own project. Assign the tasks to individual team members or to subgroups.

## Techniques to assist project management

There is a wide range of techniques that are employed to help with project management. These include Gantt charts, logbooks and the identification of subgoals.

## Gantt charts

A Gantt chart provides a means of seeing the overall project, including deadlines and resource allocation, in a pictorial manner. The Gantt chart is basically a table that has the tasks listed down the left-hand side and the calendar or days listed across the top. Figure 8.6 provides an example.

The chart shows the tasks that are to be completed. It shows the relationships between the tasks, when each task begins and ends, and each task's duration. You can see from the chart that there are some tasks that can be performed at the same time as other tasks, and there are those tasks that rely on the completion of others before they can be started. The chart in Figure 8.6 is incomplete. It does not show all the tasks and the resources required to complete each task have not been recorded. When this is done the chart will change, as some tasks may require resources that are being used for other tasks. This will mean that the timing of tasks and the overall timing of the project will probably change. One of the goals of the project manager is to complete the project in the shortest possible time with the most efficient use of the available resources.

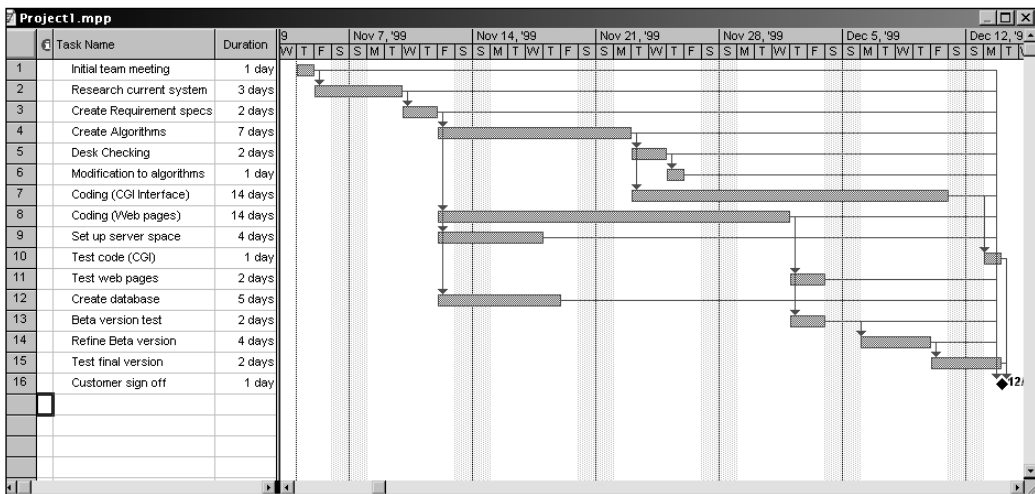


Figure 8.6 Gantt chart.

## Logbooks

Logbooks (see Figure 8.7) are a form of process documentation (see Chapter 5). Members of the project team keep a running log of their work on the project. This helps to track changes. Also, if any team member leaves the disruption to the project is minimal as the new team member is able to refer to the logbooks left by their predecessor.

## Identification of subgoals

It is obvious that the main goal of the project is to create a Web site with an integrated database that allows customers to order pizzas online. A crucial task for the project team is to identify subgoals. Just as the programming task is broken down into modules, the main goal can be broken down into many smaller goals. When the subgoals of the project are identified they are assigned to relevant team members. Some of the subgoals of this project include the creation of graphics, the development of a navigation system for the Web site, the creation of the database, the coding of programs, the coding of Web pages and the testing of modules and the completed components.

|                |      |
|----------------|------|
| Project title  | Date |
| Team member    |      |
| Stage/activity |      |
| Notes          |      |

**Figure 8.7** One of a number of possible layouts of a logbook page.

## Exercise 8.7

- 1 Identify and describe the subgoals of the project to create the Pizza Express Web page(s) for customer ordering. Present your document using a word processor. Create a Gantt chart to assist with the completion of this task.
- 2 For your own project, identify the subgoals and assign responsibility for each one to a member of your team. In this project some team members may need to have responsibility for more than one goal; some of the goals will be simple and some will be more complex. Having responsibility for a goal might mean that the team member oversees the goal rather than completing it alone.
- 3 Create the necessary project management tools such as Gantt charts. These should be created using the appropriate software. You will also need to design a format for the process diary that accompanies your project.

### Allocation of resources

In any project the amount of effort required is equal to the duration multiplied by the number of resources required at each stage. It is crucial that resources are allocated judiciously. Certain tasks within the project may require the same resources as other tasks, and, if so, the use of resources needs to be scheduled carefully. This process usually takes place after the tasks have been scheduled. Resources take on the dates of the tasks to which they are allocated. This can be shown in a **resource allocation table**, which is the task allocation table rearranged.

| Resource                | Task                       | Start  | Finish |
|-------------------------|----------------------------|--------|--------|
| Project team, client    | Requirements specification | 0      | Day 2  |
| Systems analyst, client | Analyse current system     | Day 2  | Day 5  |
| Programmer/analyst      | Create algorithms          | Day 6  | Day 13 |
| Programmer              | Code program               | Day 15 | Day 22 |

**Table 8.5** Resource allocation table.

## Identification of major milestones and stumbling blocks

**Milestones** in a project are the significant points along the path to completion of the work. Milestones can be the start or finish of some important phase, the completion of a task, the receipt of a progress payment, or any other event worth recording.

**Stumbling blocks** in a project are those events or obstacles that can affect the progress of the project. For example, the unavailability of resources at a specific time will be a stumbling block. These obstacles need to be identified early in the process and steps taken to avoid them wherever possible.

## Regular backup

There is a saying in the computing industry that the only alternative to regular backup is to do it again. Sound advice indeed. There is nothing more costly than spending hours and expensive resources on a project only to have them wasted because there has been no backup of the work completed to date. Efficient backup can mean the difference between success and bankruptcy for a company.

Regular backups are made on removable media such a DAT tape or a CD-ROM and at least one copy is kept off-site. (That is, the copy or copies are kept in a building or location that is remote from the original building.) This is done as a defence against theft or fire or some other similar disaster.

## Response to difficulties

Difficulties are sure to arise even in the most carefully planned project. It is impossible to avoid problems. Rather, processes need to be in place that will ensure that recovery from these difficulties becomes a simple thing.

## Regular reporting

An important feature of any project is regular reporting. Reports need to be given to a variety of people involved in the project. The client will be keen to see regular reports, particularly if they are required to make progress payments throughout the project. Regular reporting also allows the project manager to track the progress of all aspects of the project. It alerts the project manager to any difficulties and assists with the reallocation of resources as they become available.

## Evaluation

Evaluation of a project should be an ongoing process. It should not happen only at the end of the project. It is good practice to step back regularly and evaluate the work and there are a number of ways to do this. Evaluation can take the form of structured walk-throughs, requirements' reviews, design walk-throughs and useability inspections. Regular evaluation can help teams deliver better products faster. It has the added bonus of creating team members who learn to avoid the mistakes in the first place. This can improve the process as well as the product.

## CASE tools

The acronym CASE stands for **computer-aided software engineering**. CASE tools are used predominantly for systems analysis and design. Most of the tools that are available to software developers or project managers belong to one of five categories:

- Project management tools such as Microsoft Project help a team to estimate, plan and track schedules, resources effort and costs.

- Analysis and design tools assist in the documentation, analysis and management of requirements or in the creation of design models.
- Coding tools include code generators and reformatters and code analysis and reverse engineering tools.
- Quality improvement tools include test planning and execution tools and static and run-time code analysers.
- Configuration management tools let you track changes and defects, control access to files, and build a product from its components.

These tools help to save time and reduce errors by automating a part of the development and management process. An important point to remember is that these tools only help to implement the processes; they are not a substitute for having established processes.

## Exercise 8.8

- 1 Create a resource allocation table for the Pizza Express customer Web page required.
- 2 Identify points in the development of the Web site for Pizza Express that you would consider to be milestones. Explain why you have made your choices. Try to identify some points in the development where stumbling blocks could occur, and explain how you might overcome them.
- 3 Describe the types of CASE tool that you think would be appropriate for the development of the Web site for Pizza Express. At what points would you use these tools? Give reasons.
- 4 For your own project, identify the milestones that you will look for. Record these milestones in the project log and the dates they have been reached. Also identify any stumbling blocks that occur and explain how you overcame them. (This will also help you overcome or avoid similar problems in later projects.)
- 5 For your project, devise a suitable backup system and log these backups as your project progresses.
- 6 At regular intervals evaluate the progress of your project, reporting on it and noting the results in your project log.

## Project documentation

All projects need to be documented from start to finish. The documentation generated at each stage of the project must be kept. This section looks briefly at the nature of documentation.

Relevant documentation generated for any project should include algorithms, Gantt charts, manuals, system documentation, data dictionaries and diaries or logbooks. As well, any memos or notes made during the development process should be included with the process documentation.



## Algorithms

The final algorithms are kept for future reference in case the software needs to be modified. They allow new programmers or project team members to see what has gone before. An examination of the existing algorithms gives the new team members a good overview of the software and enables them to be able to avoid duplicating tasks that have been completed previously.

## Gantt charts

Gantt charts are a useful form of documentation during the course of the project; they let the project manager keep track of the progress of the various aspects of the project. It is also useful to keep the Gantt charts as a record of how the project progressed to its completion. Project teams who undertake similar projects in the future can examine these charts to see where improvements might be made, or they might be able to reuse parts of the charts that are relevant to the new project.

## Manuals

Manuals include user manuals, installer manuals and other types of help manuals and documentation that relates to the software. User manuals are usually written after the software is complete and the focus is on showing the end user how to use the program. Installer manuals are written chiefly for systems administrators; they guide these people through the installation process, and they may include instructions on how to load the software onto stand-alone machines, network servers and networked workstations.

Other types of manuals might include the internal documentation in the software itself. This is intended for use by programmers who need to debug or rewrite the software. It is usually in the form of comments throughout the code of the program. The computer ignores these comments as they are intended only for humans. Special characters that tell the computer to ignore them usually precede the comments. For example, in Java any such comments are preceded by the hash symbol (#). In the following two lines of Java code the first line is the comment and the second is the code that is executed.

```
Get the date
chomp ($date = '$DATE');
```

## System documentation

System documentation includes items such as algorithms and dataflow diagrams that show how the overall system works. The purpose of system documentation is to provide a detailed description of the system and to provide information that will assist with the maintenance of the system.

## Data dictionaries

Data dictionaries are the tables that show the attributes of each item of data in a database. The table usually includes the field name, the data type, the size, and an example of an entry, often together with a reason for the inclusion of the data item.

## Diaries or logbooks

Each member of the project team keeps a diary or logbook. If anyone leaves the team, their replacement will be able to refer to the diary or logbook to minimise

the amount of time needed to become a productive member of the team. A diary also allows programmers to progressively keep track of changes that they have made to the program. This enables them to easily undo any change that may not work as expected.

## Exercise 8.9

- 1 Create a database of the documentation that has been produced for the customer order Web page for Pizza Express. This database will help you track the progress of the project as well as being a record of the documentation associated with this case study.
- 2 Use an appropriate filing method to keep the documentation for your own project current and useable. You might like to use a clone of the database you created in question 1 to help the management of the documentation.

## Social and ethical issues related to project work

Software is built for people's use. Without users, software becomes a meaningless collection of bits. Whenever we embark on a software project we need to consider the social and ethical issues that relate to both project development and the finished product.

Some of the relevant issues are ease of use, accessibility of technical language, copyright and ergonomics.

### *Ease of use*

Software has to be easy to use. It doesn't matter how fancy the programming is, if it is difficult to use or it doesn't take into account the users' needs it will not fulfil its purpose. When developing software you need to spend time learning the requirements of the software and designing a user interface that can be understood by the potential user.

### *Accessibility of technical language*

When designing the user interface we need to avoid the use of technical language as much as possible. Jargon or language that is too technical puts off the user and can get in the way of what the software is actually trying to achieve. Where technical language is required it should be accompanied by a brief explanation.

### *Copyright*

Piracy or the illegal use of software is one of the most common computer crimes. Copyright laws cover most software. Every effort must be made to gain the permission of the copyright holder for any material that is used in your software. This includes all graphics, text and modules that you use. It is too easy to copy software. It is estimated that software piracy costs the Australian computer industry at least \$400 million dollars per year.

## Ergonomics

Since software is the link between the computer and the operator, it should be ergonomically designed to make the operator feel relaxed and comfortable. In designing the Web pages for the online pizza ordering system, for example, every effort should be made to ensure that the text is easy to read, that navigational elements are in consistent places, and that the pages load quickly so as to minimise the amount of waiting time. For example, the choice of text and background colours can mean the difference between customers persisting and actually placing orders and moving away from the Web site to other more visually pleasant sites. A trend in Web pages for some time was to use red text on a black background. This combination is actually quite hard on the eyes and Web developers have moved to a more natural white background with black text. Good use of white space also helps enhance the legibility of the page.

### Exercise 8.10

- 1 Evaluate the Web site you designed for Pizza Express in terms of ergonomics. Can you find any areas for improvement? Compare the ergonomics of your solution with those of other teams.
- 2 What are the social and ethical issues that must be considered in the design of the Pizza Express Web site? Explain how you have addressed these issues or how you could change the solution in order to address the issues. Report on your findings.
- 3 Evaluate your own project in terms of its ergonomics. Report on your findings.
- 4 Discuss the social issues that may have become evident as you created your project. Report on how these issues have been addressed. Are there any issues that you have missed in the development of the project? How can you modify the project to address these issues?

### Team Activity

Evaluate the product developed by another team from your class. Your evaluation should focus on whether the software solves the stated problem, how well it solves the problem, the type and quality of user and development documentation and the user interface. Present your report as a word-processed document.

When you receive the report on your team's project, discuss ways in which your team might have improved on those areas identified as weaknesses in the development and list those areas of strength. Keep this document for reference when you develop your individual projects for the HSC Course.

# Review exercises

- 1 Copy the following passage and complete it by filling in the blanks with the appropriate terms or phrases.  
The main steps in implementing a project are \_\_\_\_\_ the problem, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_ the solution. The first of these steps involves \_\_\_\_\_ the problem and then identifying the inputs, \_\_\_\_\_ and \_\_\_\_\_.  
Once we have \_\_\_\_\_ the problem we need to decide upon a suitable \_\_\_\_\_ approach
- 2 Name the main steps in implementing a software project and briefly describe what happens at each step.
- 3 A software solution is needed for the management of a large personal collection of videos and DVD discs. Describe the inputs, processes and outputs that would be necessary for this solution. Draw up an IPO chart to represent this system.
- 4 Describe the development approach that you would follow in creating the software solution for the problem stated in question 3.
- 5 Explain why it is desirable to break the video management program into smaller modules for development. Identify the major subroutines that would be necessary for this solution.
- 6 Design an algorithm that describes the operation of the problem in question 3.
- 7 Design a suitable set of test data for the video management problem in question 3. Arrange these data items into a table which includes the expected outputs and the reasons for including each test data item.
- 8 Describe the process of desk checking in your own words. Why is it essential for an algorithm to be desk checked before implementation in a language?
- 9 What language would be appropriate for the video management program in question 3? Justify your choice by giving reasons.
- 10 Explain why documentation is important in the process of software development. Describe the documentation that will be produced during the development of the video management program in question 3 and explain its purpose.
- 11 Explain the reasons for using project management techniques during the software development cycle. Are these techniques relevant for the video management program discussed in earlier questions? Justify your answer.
- 12 Identify the tasks that will be needed in the video management program in question 3. Create a task allocation table for this project.
- 13 Describe the tools that might assist a project team in managing a project. Explain the purpose of each of the tools you describe, giving an example of where its use is appropriate.
- 14 Explain the purpose of logbooks and diaries in the software development process. Is it appropriate to use a logbook or diary when developing the video management program discussed in earlier questions? Give reasons for your answer.
- 15 Describe the social and ethical issues that may arise during the process of software development. Which of the issues would be relevant in the video management program discussed in this set of exercises? Explain your answer.

# Chapter summary

- The steps in implementing a project are to define the problem, plan the solution, build the solution, check the solution and modify the solution.
- Defining the problem consists of understanding the problem and identifying the inputs, processes and outputs.
- To understand the problem, we break it down into smaller, more manageable parts.
- For each of the parts of a problem, we identify the inputs, processes and outputs.
- Tools such as IPO charts and dataflow diagrams help us to understand the problem.
- Planning the solution consists of identifying a suitable development approach, designing the algorithms, determining the data structures, identifying subroutines, designing test datasets, desk checking algorithms, and identifying existing code that can be used.
- Development approaches range from ad hoc approaches to highly structured ones.
- Ad hoc approaches are used to develop personal support systems such as spreadsheets.
- Structured approaches are used to develop larger and more complex programs.
- Algorithms can be represented in a number of different ways. Approved methods for this course are as flowcharts and pseudocode.
- Test data should test the limits of expected data as well as any boundary values. Each test data item should be accompanied by the expected output.
- Desk checking is used to find out whether the algorithm's logic is correct.
- Identifying existing code that can be used will help to reduce development time and costs.
- Building the solution involves translating the algorithms into an appropriate language.
- Programmers code and test the solution.
- An appropriate language for the program is chosen.
- The program is first checked by running it on the computer using the set of test data. It is then checked in a 'real' situation with people who have not been involved in its development.
- The solution is documented as it progresses through its development.
- Documents created will include systems representations such as dataflow diagrams, algorithms, tutorials, test data, a data dictionary and logbooks.
- Once the solution has been completed and delivered it is constantly monitored for performance and evaluated.

# Chapter summary

- Tasks involved in the development of a project need to be identified and assigned to appropriate team members.
- Tasks can be identified by focusing on key terms in the program specifications.
- Tools to help in project management are Gantt charts, logbooks, and the identification of subgoals.
- A Gantt chart shows the time allocated to each of the tasks in a graphical form.
- Logbooks are used to keep track of changes and to allow members joining the team to quickly understand what has to be done.
- Resources need to be scheduled, as the same resource may be used by different team members. A task allocation table can help with this.
- Milestones are significant points in the development path.
- Stumbling blocks are events or obstacles that affect the progress of the project.
- The project should be backed up regularly and stored off-site.
- Processes need to be in place to deal with any unforeseen difficulties.
- The project manager should receive regular reports on the progress of the project.
- Evaluation of the project is an ongoing process.
- CASE tools can be used to help with the software development process.
- All documentation associated with the development process should be kept.
- Social and ethical issues need to be addressed during the software development process.
- Relevant social issues are ease of use, accessibility of technical language, copyright and ergonomics.

# Appendix I Sample Examination Paper

This paper contains sample questions on the Preliminary course. It does not purport to be an examination for the course. Each question has been matched with a Preliminary outcome. Although all care has been taken, the author is not responsible for any mismatching with the outcomes.

## QUESTIONS 1–20

Select the alternative A, B, C or D that best answers the question.

1. Touch typing allows a user to avoid mistakes by:  
A looking at the keyboard                      B using all the fingers on the keyboard  
C looking at the document                      D avoiding strain and injury      (P 3.1)
2. A single-user software licence agreement allows the user to load the software on:  
A all the office computers                      B any computer she uses  
C a single network                                D a single computer                      (P 3.1)
3. The graphical user interface was first used by:  
A Xerox on their Alto Star computer  
B Apple on their Lisa computer  
C Apple on their Macintosh computer  
D Microsoft in Windows                      (P 2.2)
4. Computer programs are protected by copyright because:  
A programs can be easily copied by users  
B programmers are creative people like artists and authors  
C software companies want to make money  
D other programmers might use their ideas in their programs      (P 6.1)
5. Which of the following is a not a peripheral device?  
A mouse                                              B hard disk  
C random access memory                      D video display unit                      (P 1.1)
6. A program designed to copy files from one location to another is:  
A an application                                      B a file manager  
C the operating system                              D a utility                                      (P 1.1)
7. The first programming languages that were designed to be independent of the processor used are:  
A first generation                                      B second generation  
C third generation                                      D fourth generation                      (P 2.1)
8. The people who look after the computer resources in a large system are called:  
A computer operators                              B data entry operators  
C information systems managers                      D maintainers                              (P 6.1)

9. Jasmine wants a computer program to manage her finances. She is most likely to develop it using a:
- A structured approach
  - B prototyping approach
  - C rapid application development approach
  - D end-user development approach
- (P 4.3)
10. An integer data type is most suitable to store:
- A a person's height in metres
  - B a telephone number with its STD code
  - C an employee's wages
  - D the number of woolly jumpers in stock
- (P 1.2)
11. The order of filling in an IPO chart is:
- A inputs, processes, outputs
  - B outputs, inputs, processes
  - C processes, outputs, inputs
  - D inputs, outputs, processes
- (P 4.2)
12. Which one of the following algorithms is an example of a post-test repetition?
- A IF first < second  
THEN  
    display first  
ELSE  
    display second  
ENDIF
  - B FOR counter goes from first  
    to last  
    process arrayelement(counter)  
NEXT counter
  - C process arrayelement(counter)  
WHILE counter < last  
    increment counter  
    process arrayelement(counter)  
ENDWHILE
  - D REPEAT counter < last  
    increment counter  
    process arrayelement(counter)  
UNTIL counter > last
- (P 5.2)
13. Given that the following are syntax definitions in a programming language:
- <identifier> = P | Q | R  
 <operator> = & | @  
 <simple statement> = <operator> <identifier> | <simple statement>  
 <compound statement> = <identifier> <simple statement> [<simple statement>] IS  
 <identifier>
- Which of the following is a syntactically correct compound statement in that language?
- A P&Q@R
  - B P&QRISP
  - C PISQ@R
  - D P@QISR
- (P 5.2)
14. Which of the following features is not a consideration when designing an input screen display:
- A consistency of design with other screens
  - B input data type
  - C text sizes and styles
  - D the placement of screen elements
- (P 6.3)
15. Which one of these types of documentation from previous projects can help in managing future software development projects?
- A process documentation
  - B system documentation
  - C user documentation
  - D internal documentation
- (P 6.2)





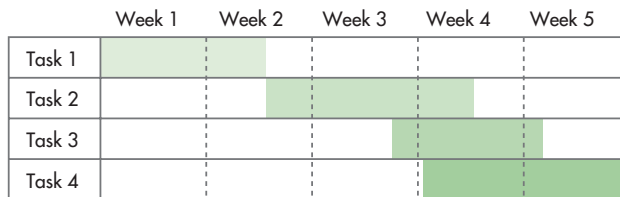
```

C BEGIN find_largest
 set index to 1
 set largest to 0
 WHILE index < array_size
 IF largest > number(index)
 THEN set largest to
 number(index)
 END IF
 set index to index + 1
 END WHILE
 display largest
 END find_largest

D BEGIN find_largest
 set index to 1
 set largest to 0
 REPEAT
 IF largest < number(index)
 THEN set largest to
 number(index)
 END IF
 set index to index + 1
 UNTIL index < array_size
 display largest
 END find_largest
(P 5.2)

```

19. While a computer is waiting for user input it:
- A does nothing
  - B loads in the next program section
  - C cleans out RAM
  - D cycles through the input devices looking for input (P 1.3)
20. Use the following Gantt chart to answer this question. Which one of the following statements is true?
- A Task 2 can be started before task 1 is finished.
  - B Task 3 can be started before task 2 is finished.
  - C Task 3 will be finished before task 2 is finished.
  - D Task 4 will be finished before task 3 is finished. (P 5.1)



21. **Social and ethical issues**
- (a) Name two factors that can lead to operator injury or discomfort. Explain how these injuries and discomfort can be avoided. (P 3.1)
  - (b) (i) Explain the similarities and differences between shareware and public domain software. (P 3.1)
  - (ii) A software developer modifies a public domain software program and then sells it. Explain why the software developer is not acting ethically or legally. (P 3.1)
  - (c) Describe two skills needed by a software developer. (P 6.1)
  - (d) A number of features of the modern computer have made it possible to use a graphical user interface. Explain why these features were not available in early computers. (P 2.2)
22. **Hardware and software**
- (a) Describe the five elements of a computer-based system. (P 1.1)
  - (b) Explain the fetch–execute cycle as it applies to starting a program. (P 1.3)
  - (c) A program is being designed to help children read. Explain why a programmer would use a graphical user interface for this program. (P 6.1)

**23. Software development approaches**

- (a) Describe the prototyping approach to software development. (P 4.2)
- (b) Explain how a user participates in the development of a prototype into a fully working application. (P 6.1)
- (c) Describe a problem that would benefit from a prototyping approach to software development. Explain why prototyping is suitable for the development of the solution. (P 4.3)

**24. Defining the problem and planning the solution**

The following problem is to be used for your answers to this question.

Chris is an estimator for a vertical blind company. He needs a computer program to generate an estimate of the cost of blinds for the windows in a house. The window sizes are measured and the customer chooses the fabric. Chris wants a printed estimate that includes the cost of the blinds and the Goods and Services Tax of 10%.

- (a) Create an IPO chart to describes the operation of the program. (P 4.2)
- (b) For each of the following data items required by the program, choose an appropriate data type.
  - (i) customer name
  - (ii) length of the window
  - (iii) number of windows
  - (iv) whether or not a pelmet is needed (P 1.2)
- (c) Rearrange the steps in the following algorithm so that it reads in the window measurements and calculates the cost of the blinds. The program is to stop when 0 or a negative value is entered for the length of the window.

```

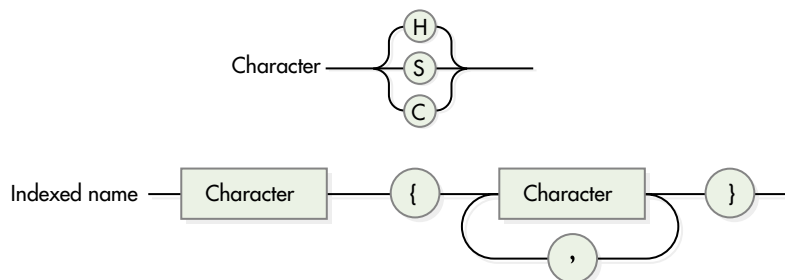
BEGIN
set length to user input
set width to user input
set area to length * width
set cost to 20 + area * 15.75
display cost
set length to user input
set width to user input
REPEAT
WHILE length > 0
END

```

(P 5.2)

**25. Building software solutions**

- (a) The letters of the alphabet are divided into vowels (A, E, I, O and U) and consonants (all other letters). A two-letter word must consist of either a vowel followed by a consonant or a consonant followed by a vowel. Write these syntax rules either in BNF or as a syntax structure diagram. (P 5.2)
- (b) Given the following syntax definition for a programming language, create a syntactically correct indexed name. (P 5.2)



- (c) An algorithm is required to input a series of values. The first value entered is the number of values that will be processed. The algorithm is to calculate the total and the average of the values entered, ignoring the first number. There are two errors in the following algorithm. Describe the errors and rewrite the algorithm to correctly solve the problem. Use the test data values 4, 3, 5, 2, 14 which should output a total of 24 and an average of 6.

```
BEGIN
 set count to zero
 set total to zero
 read howmany
 read value
 REPEAT
 add value to total
 increment count
 UNTIL count = howmany
 set average to total divided by value
 display total
 display average
```

END (P 4.2)

- (d) Describe the purpose of a prompt. Design a prompt that will inform the user that a wrong key has been pressed. (P 6.3)
- (e) Explain why internal documentation is important in a coded program. Illustrate your answer with an example. (P 5.2)
- (f) Explain why a program module should be written so that it can be reused. Use a simple example algorithm to help answer the question. (P 5.2)

## 26. Checking the software solution

- (a) An airline carries First Class, Business Class and Economy Class passengers. Each passenger is given a baggage allowance as described below:

*First Class:* 88 kg carried free, excess baggage being charged at \$5 for each kilogram or part thereof over 88 kg.

*Business Class:* 66 kg carried free, excess baggage being charged at \$7.50 for each kilogram or part thereof over 66 kg.

*Economy Class:* 44 kg carried free, excess baggage being charged at \$10 for each kilogram or part thereof over 44 kg.

Design a set of test data pairs that could be used to calculate the amount payable in excess baggage. Justify the inclusion of each pair of values.

You need test only for positive values. (P 5.2)

- (b) The following algorithm is supposed to exchange two values. Use the test data set below to perform a desk check on the algorithm.

Test Data: 3,6

Algorithm:

```
BEGIN
 set number_1 to user input
 set number_2 to user input
 set number_1 to number_2
 set number_2 to number_1
 display number_1, number_2
```

END (P 4.2)

- (c) Describe the roles of the members of a structured walk-through team. Describe the process of a structured walk-through. (P 6.1)
- (d) Explain the differences between top-down and bottom-up testing. (P 5.1)

## 27. Modifying software solutions

- (a) Name and describe two of the reasons that may be the cause of software being modified after its initial installation. (P 3.1)
- (b) Convert the following code into an algorithm expressed either as a flowchart or as pseudocode.

```
MODULE addmarks! **This module adds the marks for each question **
 INPUT: question[1..10]!
 OUTPUT: total_mark!
 VARIABLE
 index is INTEGER!
 BEGIN
 index <- 0!
 total_mark <- 0!
 WHILE index <= 10
 total_mark <- total_mark + question[index]!
 index <- index + 1
 ENDWHILE
 END
```

ENDMODULE! \*\*total\_mark now contains the total score for the test\*\* (P 4.3)

- (c) Explain why a programmer has to fully document a modification of an existing program. (P 6.2)

## 28. Developing software solutions

All questions relate to the following case.

An airline carries First Class, Business Class and Economy Class passengers. Each passenger is given a baggage allowance described below:

*First Class:* 88 kg carried free, excess baggage being charged at \$5 for each kilogram or part thereof over 88 kg.

*Business Class:* 66 kg carried free, excess baggage being charged at \$7.50 for each kilogram or part thereof over 66 kg.

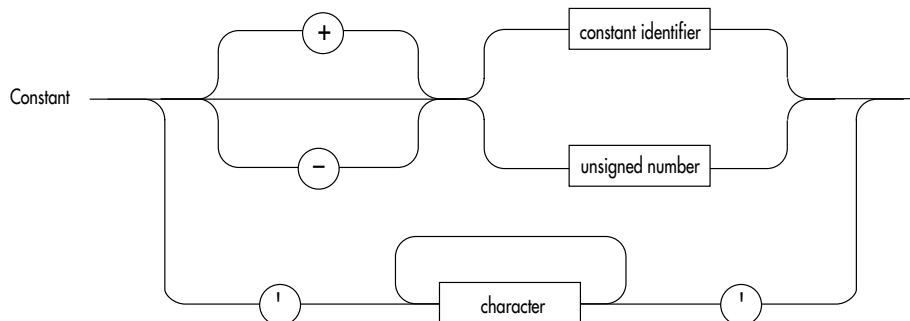
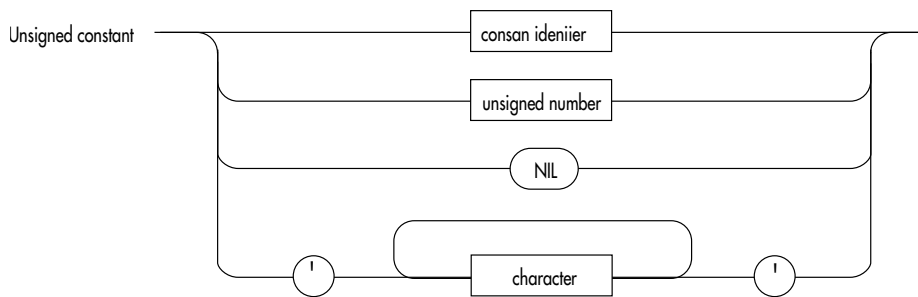
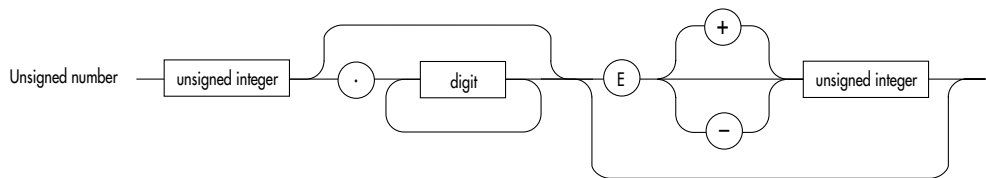
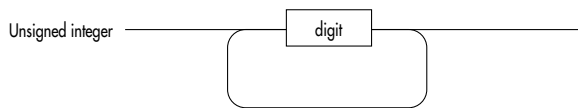
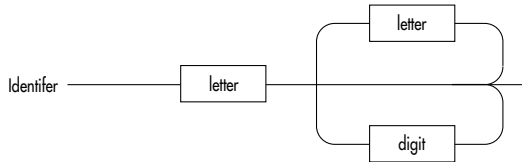
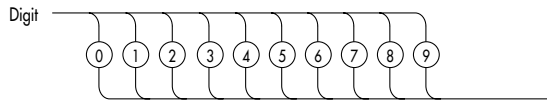
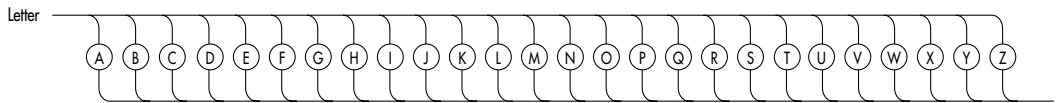
*Economy Class:* 44 kg carried free, excess baggage being charged at \$10 for each kilogram or part thereof over 44 kg.

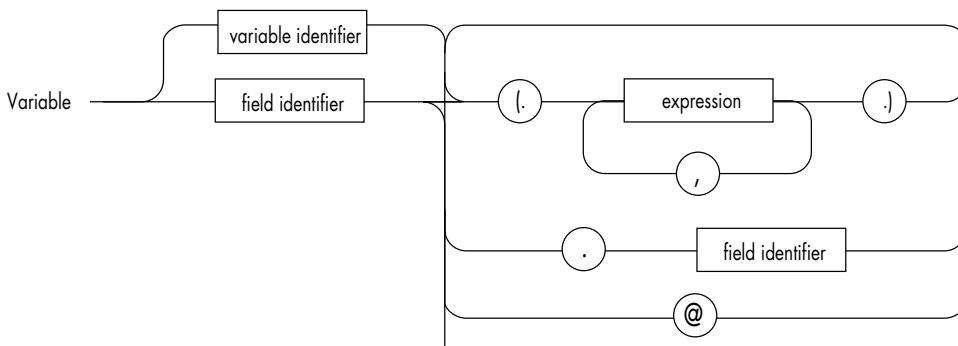
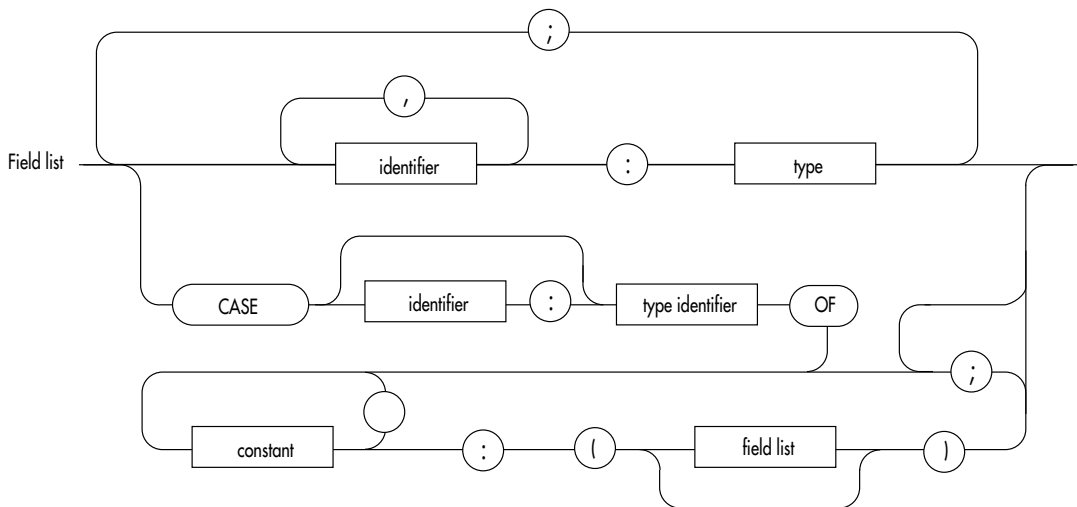
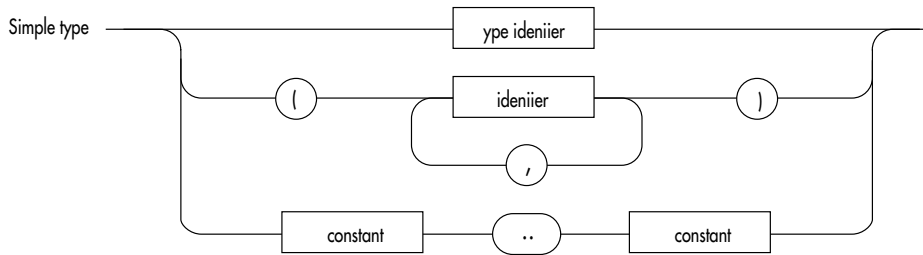
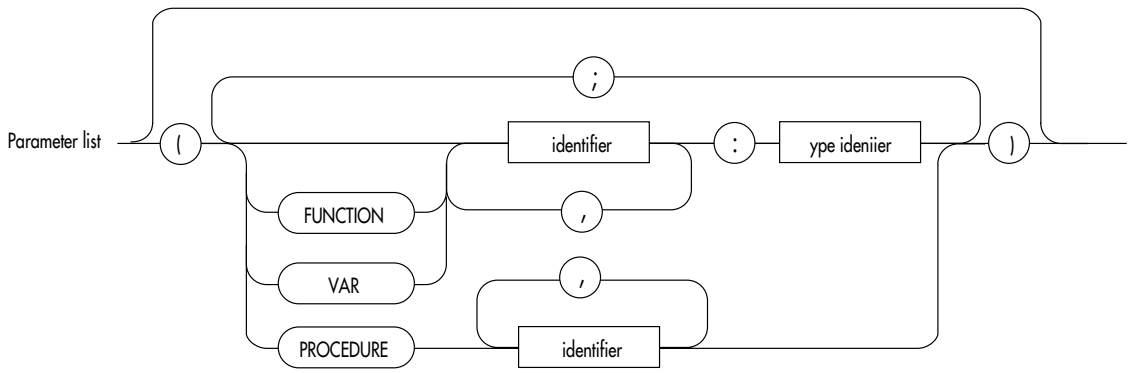
When the baggage is checked in at the airport, a baggage label is created showing the weight of each article, the 3-letter destination code and the passenger's name. Each passenger can check in only one article. A bill for excess baggage is created if the weight of the bag is over the limit for that passenger's ticket.

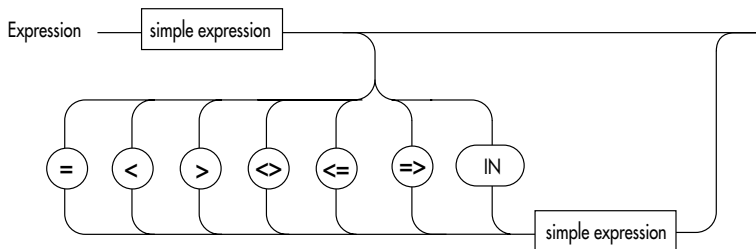
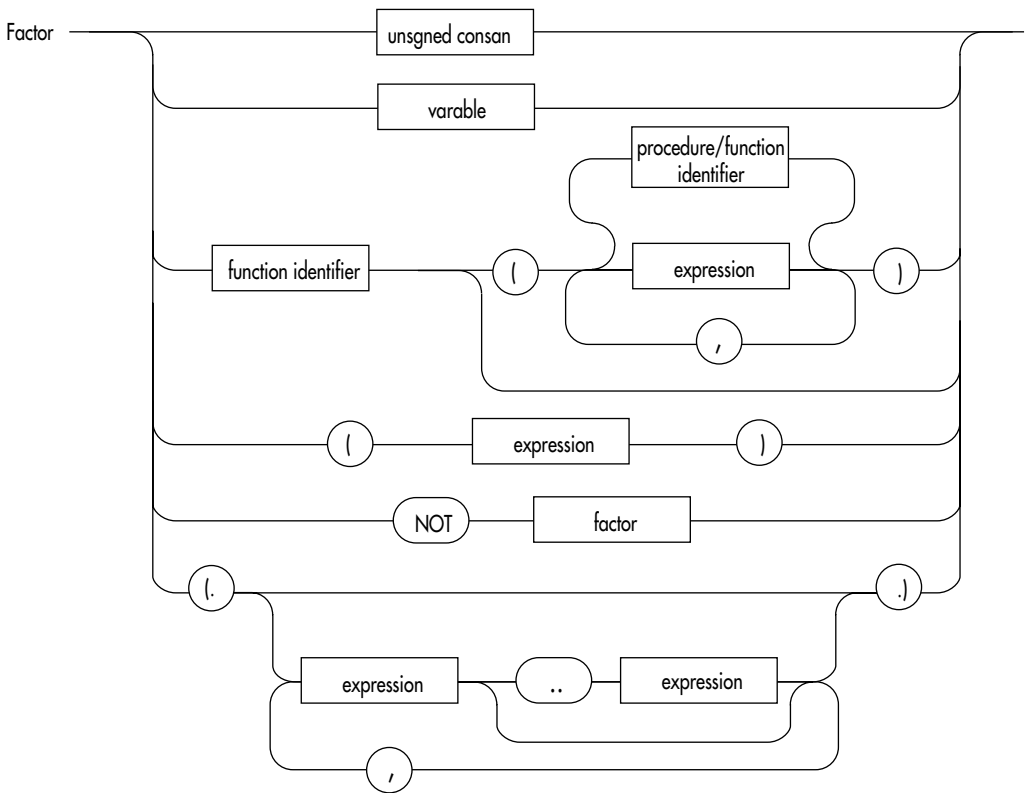
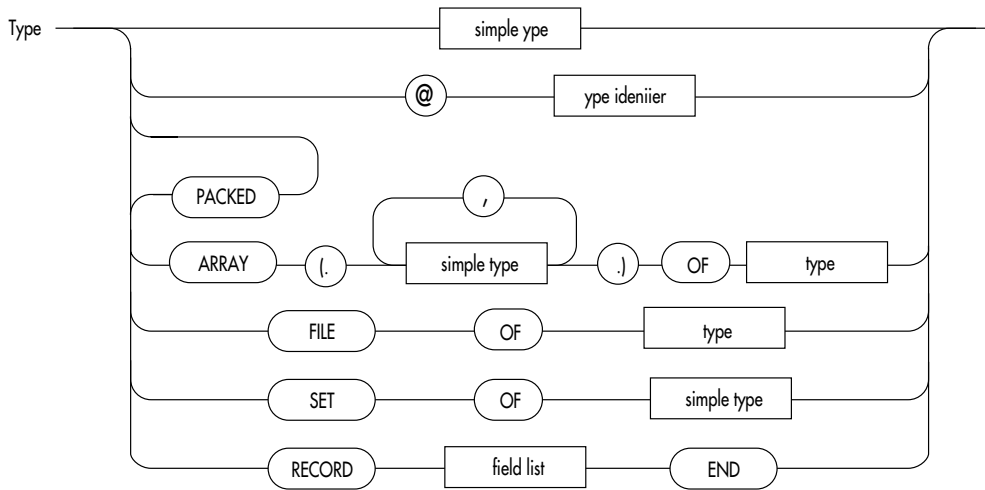
The airline needs to update its baggage handling facility. Management has asked you to design the computer program to process the baggage labels and excess baggage bills.

- (a) Identify the inputs, processes and outputs and present them as an IPO chart. (P 5.2)
- (b) Choose an appropriate development approach for this problem and justify your choice. (P 4.3)
- (c) List and briefly describe the steps your program development would take. (P 4.2)
- (d) Design an appropriate graphical user interface for the main menu of this program. (P 6.3)

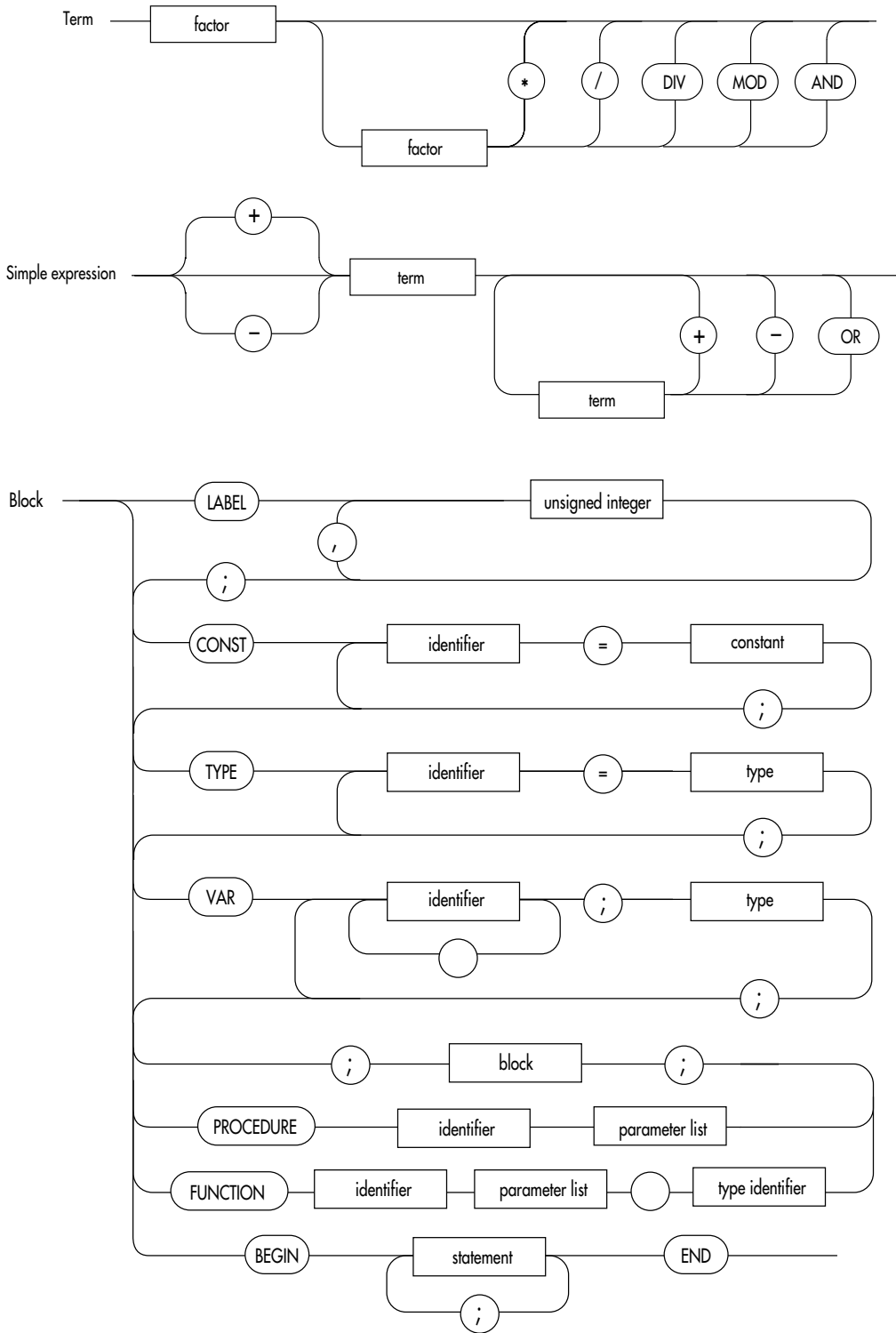
# Appendix 2 Pascal Syntax Structure Diagrams

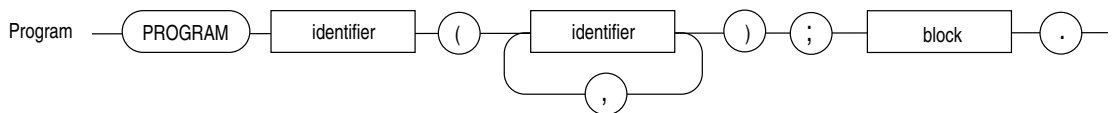
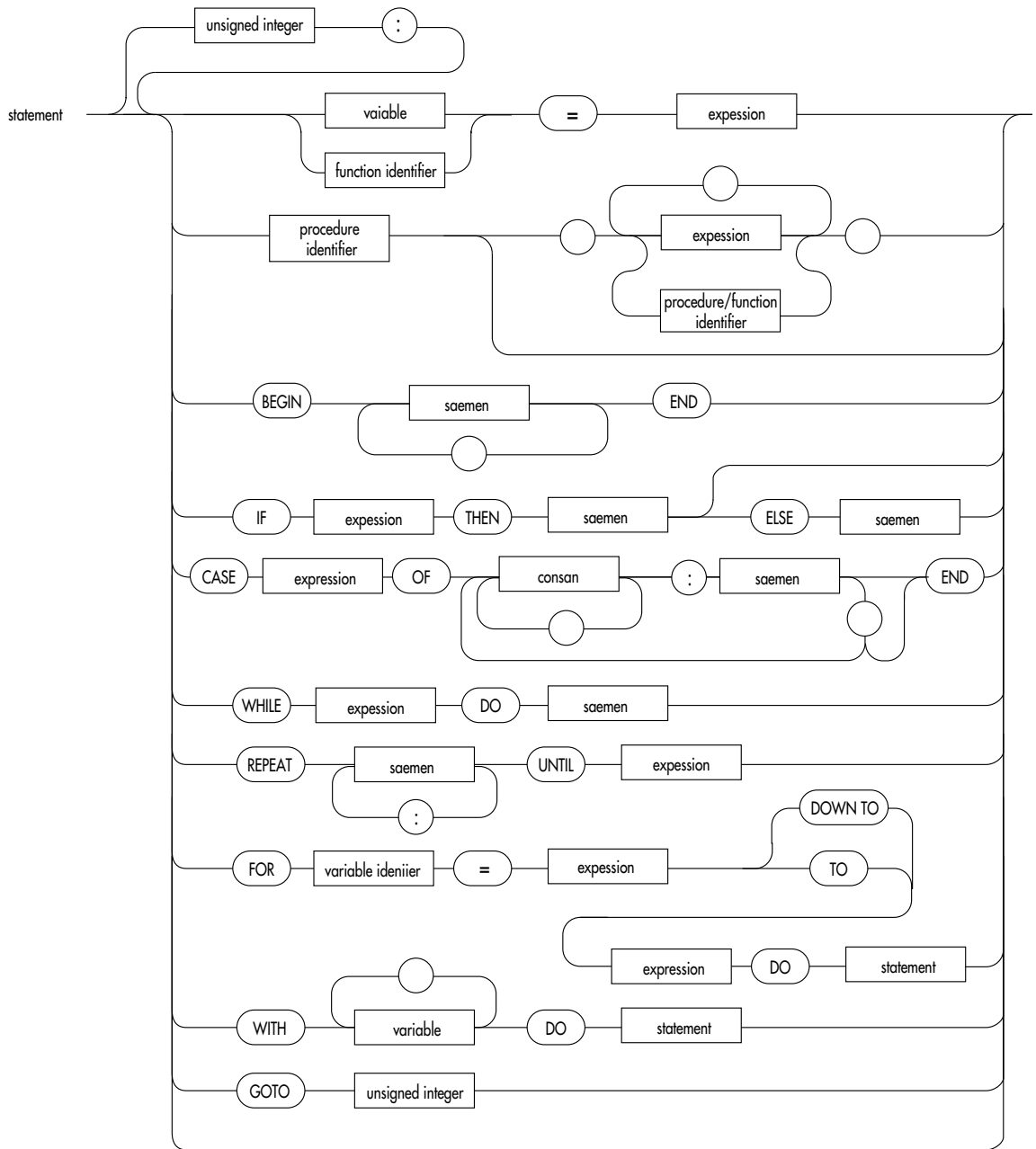












# Glossary

- algorithm** a series of steps which, when performed correctly, will solve a problem in a finite time.
- alt key** a *modifier key* which is used in combination with a second key. It changes the signal of the key it is used with.
- application software** software that performs a specific task.
- array** a *structured data type* containing a number of related data items, each having the same data type.
- ASCII code** a standard method of representing letters, numerals and special characters as unique strings of binary digits.
- binary selection** a *control structure* in which a choice of two paths is presented. The path executed depends on the result of a condition.
- bit** a single binary digit. It is the smallest unit of storage in a digital computer.
- BNF (Backus-Naur form)** a text-based method of stating the rules of a language. See also *EBNF* and *syntax structure diagram*.
- Boolean data type** a data type in which only two possibilities, usually either true or false, are represented by a variable.
- breakpoint** a place 'marked' in a program where execution of the program is suspended so that the values of variables can be examined. Breakpoints are usually used only during debugging of a program.
- byte** an eight-bit string of binary digits.
- cache memory** a type of *primary storage*, usually located between the CPU and RAM, which is used to speed up access to the program and data.
- called** a subprogram is 'called' when control passes to that subprogram from the main program.
- calling module** the module from which a call to another module is made.
- central processing unit (CPU)** a unit that retrieves, decodes, interprets and executes instructions.
- character** the smallest unit of data normally handled by people.
- character data type** a *simple data type* in which only one coded character can be represented by the variable.
- command based interface** a human-computer interface in which the person has to type in commands in order to manipulate data.
- command key** a *modifier key* which is used in combination with a second key. It changes the signal of the key it is used with.
- command-line interface** an alternative term used to describe a *command-based interface*.
- compound statements** a statement in a programming language which combines a number of instructions, each of which is a simple statement.
- constant** a value which cannot be changed during the execution of a program.
- control** one of the logical elements of a computer system. The control element of a computer system coordinates the processes that are carried out.
- control key** a *modifier key* which is used in combination with a second key. It changes the signal of the key it is used with.
- control structures** a term which describes the three basic structures of an algorithm (*sequence, selection and repetition*).
- database management systems** a software system which allows a database to be created, maintained and accessed.
- decrement** to decrease the value of a variable by 1.
- desk check** manually checking the logic of an algorithm by using test data.
- direct access** a method of accessing data in which a record can be accessed without having to access previous records. Also known as *random access*.
- double precision** the use of a greater than normal number of bytes to store a numerical value, allowing a greater degree of accuracy.
- Dvorak keyboard** an alternative keyboard layout which was designed to increase the speed of a typist. It was named after its inventor August Dvorak. See also *Qwerty keyboard*.
- EBCDIC (extended binary coded decimal interchange code)** a binary coding for characters which uses eight bits to represent each character. See also *ASCII*.
- EBNF (extended Backus-Naur form)** a text based method of stating the rules of a language. See also *BNF* and *syntax structure diagrams*.
- end of file (EOF) mark** a pattern of bits which represents the character used to indicate the end of a string of text. It is usually used to show the end of a file stored on an external storage medium such as a magnetic disk.
- ergonomics** the study of the relationship between a person and their working environment.
- escape key** a key that by itself may be used to terminate an action prematurely. The escape key may also be used as a *modifier key* in conjunction with another key.
- evolutionary prototyping** a method of prototyping in which the *prototype* is developed into the final software solution to the problem.
- field** one data item within a *record* data structure.
- file** a block of data which may have been written to a storage device.
- fixed disk** a magnetic storage device in which the disk is permanently housed with its read/write mechanism. Also known as a *hard disk*.
- floating point data type** a simple data type which represents rational numbers.
- floppy disk** a flexible plastic disk housed inside a protective cover. It may be removed from the read/write mechanism.

**function** a predefined set of operations which returns a value.

**function keys** one or more keys which may be programmed to perform a particular task. Function keys are often found above the character keys on a keyboard.

**GIGO (Garbage In Garbage Out)** an acronym which states that if the input data is not correct, then the output will not be useful.

**graphical user interface (GUI)** a human-computer interface which employs icons and menus to assist the user to navigate through the choices of a program.

**guarded loop** a loop in which the decision is placed at the start of the loop. In pseudocode, it is the WHILE.....ENDWHILE structure.

**hard disk** see *fixed disk*.

**hexadecimal** a counting system based on sixteen. The characters 0 to 9 and A to F are used to represent the sixteen digits needed for the hexadecimal system.

**identifier** a name given to a *constant, variable, function* or *subroutine* of a program.

**increment** to increase the value of a variable by 1.

**index** the value which represents the position of a data item in an array.

**input** the process of transferring data into a computer system from the outside by means of a peripheral device.

**input/output table** a table of test data that lists the test data items and the expected outputs.

**integer data type** a data type used to represent positive and negative whole numbers.

**internal documentation** documentation included in the source code, consisting of *intrinsic documentation* and remarks (also called comments).

**intrinsic documentation** documentation 'built into' the source code. The main type of intrinsic documentation is the appropriate choice of *identifiers*. The use of indentation to show that program modules may also count as intrinsic documentation as it makes the logic of the program clearer.

**IPO (input processing output) chart** an IPO chart tabulates the inputs, processes and outputs required for a system.

**iteration** looping through a process a number of times. See also *repetition*.

**least significant bit** the bit in a binary string that has the smallest value. It is usually the bit at the extreme right of a byte or group of bytes.

**linear search** a search which progresses, one element at a time, from the first indexed element of an array towards the last.

**loop** an alternate term for a *repetition* or *iteration*.

**metalanguage** a method of describing the syntax of a language.

**MIDI (musical instrument digital interface)** a standard for the connection of a musical instrument to a computer using an interface.

**modifier key** a key on a keyboard which changes the 'meaning' of a key. See *alt key* and *control key*.

**module** a part of a program, such as a subprogram or function, which performs a specific task. A module will pass data to and/or accept data from other parts of the program.

**most significant bit** the bit in a binary string that has the greatest value. It is usually the left-most bit of a byte.

**multiway selection** a control structure in which a choice is made from a number of alternatives. The choice is based on the value of an expression.

**nibble** a four-bit binary string.

**non-terminal symbols** elements of a language that are defined elsewhere in the language description. See also *terminal symbols*.

**numerical keypad** a keypad containing only keys which represent the digits 0 to 9.

**nybble** an alternate spelling for *nibble*.

**octal** a counting system based on eight; only the digits 0 to 7 are used to represent numbers. See also *binary selection* and *hexadecimal*.

**operating system** the software that manages the resources of a computer.

**output** the process of transferring data from a computer system to the outside by means of a peripheral device.

**overflow** a condition where an operation is too large for the storage allocated. In this case, some data bits are 'lost', leading to a wrong value being calculated.

**pixel** the smallest element that can be displayed on a screen display.

**post-test repetition** an *iteration* in which the termination test is after the body of the loop. In pseudocode, a post-test repetition is identified as a REPEAT.....UNTIL construction.

**pre-test repetition** an *iteration* in which the termination test is before the body of the loop. In pseudocode, a pre-test repetition is identified as a WHILE.....ENDWHILE construction.

**primary storage** storage that is directly accessible to the CPU.

**process** an action (when the word process is used as a noun) or to perform a set of instructions (when the word is used as a verb).

**prototype** a working model of an application which is used to gather information. A prototype may be developed into the final application or it may be 'thrown away'.

**Qwerty keyboard** a keyboard design by Charles Sholes; its name being taken from the first six letters of the top row.

**random access** a method of accessing data where a record can be accessed without having to access any of the previous records. Also known as *direct access*.

**random access memory (RAM)** *primary storage* which can be written to as well as read. See also *read-only memory*.

- raster** the horizontal scanning line of electron beams used in a cathode ray tube to build up the image.
- read-only memory (ROM)** primary storage which can only be read. See also *random access memory*.
- record** a collection of related facts. A record is stored as one or more *fields*.
- recursion** a definition which incorporates itself.
- relational database** a database in which records in two or more different files are linked by a *field*.
- repetition** an algorithm structure in which a sequence of steps may be executed a number of times.
- RSI (repetitive strain injury)** an injury which is caused by the constant repetition of a task.
- screen buffer** an area of *primary storage* which is used to store the data which represents the image on a screen.
- secondary storage** storage which is not directly accessible by the CPU.
- selection** an algorithm control structure which presents two or more options, the choice of which depends upon the result of a test.
- sentinel value** a value used to 'mark' the end of a data list.
- sequence** an algorithm control structure which consists of a number of steps one after the other.
- sequential file** a file structure in which, to reach one record, each of the preceding records has to be passed over.
- simple data type** a data type that may be applied to one data element.
- single precision** the standard representation of a simple numerical data type within a particular programming language.
- standard constructs** the general term which describes the basic elements of an algorithm: *sequence, selection and repetition*.
- statement** a single step as written in a programming language.
- stepwise refinement** a process in which a problem is broken down into smaller parts until it can be easily solved.
- storage** a device or medium that can be used to hold data.
- string data type** a *simple data type* consisting of a number of *characters*.
- structured data type** a data type which is used to represent a number of related data elements as one data item.
- stub** a small *module* representing a part of the program which is still to be written.
- subroutine** a part of a program which performs a specific task.
- syntax** the set of rules that govern the way in which the elements of a language can be combined to form a *statement*.
- syntax graphs** a pictorial method of illustrating the syntax of a language. Also known as a *syntax structure diagram*.
- syntax structure diagram** a pictorial method of illustrating the syntax of a language. Also known as a *syntax graph*.
- system software** the files and resources needed by a computer system in order to allow it to run properly. System software includes the *operating system* and *utility software*.
- terminal symbols** individual characters, or strings of characters, which are used in the definition of a syntax structure.
- test data** data elements designed to test the operation of an *algorithm* or program.
- top-down design** a design approach in which a problem is broken down into a number of smaller, easier to solve, problems.
- tracing** the process of following the execution path of a running program in order to identify the source of an error.
- truncation** the 'loss' of accuracy caused by a limit to the way that results of operations can be stored.
- two's complement** a method of using a fixed length binary string of digits to represent both positive and negative integers.
- unguarded loop** a loop whose body must be executed at least once each time it is reached. Also known as a *post-test repetition*.
- user interface** the link between the user and the computer program. The most common user interfaces use screens, keyboards and mice.
- utility software** programs that perform management tasks such as formatting disks, duplicating files, virus protection, etc.
- validation** a check made by the computer that data is within allowable limits for processing.
- variable** a name used within the code of a program to reference a stored data element.
- variable declaration** a statement in a programming language that indicates the type of data that a variable will be used to store. It is used by the language translator to set aside an appropriate amount of storage for the data item.
- verification** the manual process of checking that data items have been entered correctly by comparing the entered data with the source data.
- virtual memory** a technique which uses secondary storage in the place of primary storage so that the computer appears to have more main memory than it really does.
- WIMP (windows, icons, mouse and pointer)** another term for the *graphical user interface*.
- word length** the maximum number of bits that can be processed at one time by a CPU.
- WYSIWYG (what you see is what you get)** a display which shows on screen exactly what will be output as hard copy.

# Index

- abstract data types 104
- abstraction/refinement 95–7
- access elements of an array 144–6
- accessibility of technical language 292
- algorithm description 207
- algorithms 112–13, 291
- checking 140–6
- creating for source code when they are not available 262–5
- description 115–16
- standard constructs 118–40
- alignment 198–200
- Antonelli, Kay McNulty Mauchly 247
- application programs 49, 58
- arithmetic errors 232
- arithmetic logic unit 40
- array processing 144–5
- arrays 109, 172
- ASCII code 104, 105
  
- Babbage, Charles 29
- Backus, John 153
- Backus-Naur Form (BNF) 154, 158
- checking 160
- Backus Normal Form (BNF) 154
- balloon text 212
- barcode wand 34
- binary number system 98
- conversions to and from 98–100
- binary selection 121–3, 166–7
- syntax 166–7
- Boolean data type 104
- borders 198, 199
- bottom-up testing 234
- boundary values 222
- breakpoints 178
- building the solution 74–5, 282–4
- Byron, Augusta Ada 3
  
- cache memory 43
- calling module 191
- CASE tools 289–90
- central processing unit (CPU) 30, 40
- CGI (Common Gateway Interface) 282
- chair 8–9
- character data type 104
- character readers 34–5
- check boxes 202
- checking algorithms 140–6
- checking the solution 75–6, 284–5
- choice boxes 202
- choice elements 201–2
- clock speed 58
- coding 154–74
- coding errors, types of 174–80
  
- combining code and modules from different sources 190–2
- command-based interface 55
- command line interface 33
- comments 214, 257
- compact disks 46
- comparison errors 232–3
- compilation 54
- compile-time errors 174–6
- computer system, elements 60–1
- consultation with users 193–4
- control logic errors 233
- control structures 118–40
- identified from supplied documentation 267–8
- syntax 165–71
- control unit 40
- copyright 16–17, 240, 292
- counted loop 132, 168–9
- currency 108
  
- data dictionaries 225–7, 291
- data representation 98–102
- data structure errors 233
- data types 103–11
- syntax 171–2
- data validation 188–9, 222
- database management systems 110, 280
- date 108
- debugging output statements 185–6
- decomposition 95, 96
- defining the problem 90–3, 274–6
- design evaluation 236–9
- design process 208
- desk 8
- desk checking 140–1, 229, 239
- developer's documentation 208
- developer's perspective 194–5
- dialogue boxes 202
- difficulties, response to 289
- digital cameras 35
- digitising tablet 34
- direct access 44
- documentation 207–16, 267–8
- for developers 208
- for users 208–12
- to identify control structures 267–8
- types of 207–8
- dot-matrix printers 38
  
- EBCDIC code 104, 106
- end-user development 83–4
- equality 177
- ergonomic software design issues 12–14
- ergonomically designed furniture 8–9
  
- ergonomically placed furniture 9–11
- ergonomics 4–14, 240, 293
- error-correction techniques 174–87
- error types 232–3
- ethical perspectives 241, 255
- evaluation of a project 289
- evaluation of design 236–9
- comparing different solutions to the same problem 236–8
- methods 238–9
- evaluation of implemented solution 239–40
- checking solution against original design specifications 240
- social and ethical perspectives 241
- user feedback 240
- event-driven programming 52–3
- Extended Backus-Naur Form (EBNF) 154, 158–9
- checking 160
- external documentation 207
  
- fetch-execute cycle 57–8
- fields 109
- fifth-generation languages 51–2
- files 110–11
- first-generation languages 50–1, 154
- flags 187
- flexibility 13
- floating point data type 106–7
- floppy disks 45
- flowcharts 116
- formatting 45
- fourth-generation languages 51
- functional description 209
- functions 191
  
- Gantt charts 287, 291
- glossary 309–11
- government regulations 251
- graphical user interface (GUI) 33, 55, 201
- graphics, pictures and text 206
- guarded loop 130–1
  
- hard disks 45
- hardware 30–48
- elements 30, 31
- relationship with software 57–62
- help screens 211–12
- hexadecimal system 100–2
- high-level languages 50, 154
- Hopper, Grace 89
- HTML (HyperText Markup Language) 282

- icons 203, 204
- impact printers 37
- inclusivity 22–4, 240
- incremental compilation 55
- indoor climate 5
- ink-jet printers 38
- input 30, 32, 261
- input data 194
- input devices 32–6
- input/output errors 233
- input/output table 141–2
- installation guide 210
- integer data type 104
- intellectual property 15–22
- internal documentation 207, 213–16
- interpretation 54–5, 260–7
- intrinsic documentation 214, 256
- introductory manual 209
- IPO chart 90–3, 276
- iteration 130–2
- syntax 168–71
  
- joystick 34
- justification 198–200
  
- keyboard 10–11, 32–3
- Knuth, Donald 221
  
- laser printers 38
- learnability 13
- libraries of code 188–92
- licence agreements 15–17, 20–1, 255
- light pen 34
- lighting 4–5
- liquid crystal displays (LCDs) 37
- logbooks 287, 291–2
- logic errors 178
- loop 130–2
- low-level languages 50, 154
- lowercase text 197
  
- machine language 51
- magnetic disks 44–6
- magnetic tape 44
- maintenance coding, reasons for 248–54
- changed organisational focus 250–1
- changes in data to be processed 250
- changes in government requirements 251
- introduction of new hardware/software 250
- poorly implemented code 251–3
- upgrading user interface 249–50
- manuals 291
- menus and menu bars 203–4
- message structure 196–200
- metalanguages 154
- microcomputer 30
- microprocessor 40
- MIDI (musical instrument digital interface) 35
  
- milestones 289
- modifying the solution 77, 285
- monitor 36
- motherboard 40
- mouse 11, 33
- multiway selection 123–5
- syntax 176–8
  
- navigational elements 202–4
- noise 5
- non-impact printers 37
  
- object code 54
- observation 193–4
- octal number system 100
- online help 211–12
- operating systems 48–9
- characteristics 55
- optical character readers 35
- organisational focus changes 250–1
- origin of software design ideas 17–20
- original algorithms, reading to identify inputs, processes and outputs 260–2
- original documentation, reading to understand the code 260
- output 30, 36, 195, 261
- output devices 36–9
  
- palettes 203
- paper-based documentation 209–11
- parameters 191
- Pascal syntax diagrams 304–8
- peer checking 238
- peripheral devices 30
- pixels 37, 55
- Pizza Express, case study 274–93
- building the solution 282–4
- checking the solution 284–5
- defining the problem 274–7
- modifying the solution 285
- planning the solution 279–82
- plagiarism 255
- planning the solution 72–3, 279–82
- plotters 38
- pointing devices 33–4
- post-test loop 131–2, 170–1
- pre-test loop 130–1, 169–70
- primary storage 30, 42–3
- printers 37–8
- process and control 40–1
- process documentation 208, 260
- processing 30, 195, 261
- processing records from a sequential file 145–6
- product documentation 208, 260
- program listing 213
- programming languages coding 154–74
- generations of 50–2
- project documentation 290–2
- project implementation 274
  
- building the solution 282–4
- checking the solution 284–5
- defining the problem 274–6
- modifying the solution 285
- planning the solution 279–82
- project management techniques 286–90
- prompts 205
- prototyping approach 78–81
- pseudocode 115
  
- questionnaires 193
  
- radio buttons 201–2
- random access memory (RAM) 43
- rapid application development (RAD) 81–2
- read-only memory (ROM) 42
- records 109–10, 172
- reference cards 210–11
- registers 41
- regular backups 289
- regular reporting 289
- relational databases 110
- remarks 214, 257
- repetition 130–2
- repetitive strain injury 4
- resolution 37
- resource allocation table 288
- response time, of software 12–13
- reusable code 188–9
- robustness 14
- run-time errors 176–8
  
- sample examination paper 297–303
- scanners 35
- screen 9–10, 36–7
- screen design 195–6
- screen elements 201–6
- second-generation languages 51, 154
- secondary storage 30, 44–6
- selection 121–5
- syntax 166–8
- sentinel value 145
- sequence 118–19
- syntax 165–6
- sequential access 44
- sequential file 110, 172
- record processing 145–6
- sequential programming 52
- simple data types 104–8
- social perspectives 241, 255, 292–3
- software 48–57
- relationship with hardware 57–62
- software development approaches 70–84
- software development cycle 275
- software licence agreements 15–17, 255
- reasons for 20–1
- software lifecycle 70
- software specifications 207

- sound devices 35
- source code 54, 207
  - documentation within the code 156–7
- maintainability 256–9
- presentation of a coded solution 257–9
- sources of code, and conditions that apply 21–2
- spaghetti code 263
- special key combinations 203
- standard algorithms 144–6
- standard constructs 118–40
- iteration 130–40
- selection 121–30
- sequence 118–21
- statement-coverage testing 231–2
- statements 154
- storage 42–6
- string data type 107–8
- structured algorithms 112–40
- structured approach 70–7
  - building the solution 74–5
  - checking the solution 75–6
  - defining the problem 71–2
  - modifying the solution 77
  - planning the solution 72–3
  - structured data types 104, 108–11
- structured walk-through 238–9
- stubs 180–4
- stumbling blocks 289
- subgoals, identification of 287
- subprograms 191, 214–16
- surveys 193
- swap two data items 144
- symbol 171
- symbolic assembly languages 51
- syntax 154
  - control structures 165–71
  - syntax description examples 160–2
  - syntax errors 174–6
  - syntax representation 154–65
    - Backus-Naur Form and Extended Backus-Naur Form 158–62
  - syntax structure diagrams 154, 155
  - checking 157–8
  - construction 156–7
  - interpretation 155–6
  - system administrator 208
  - system administrator’s manual 210
  - system documentation 208, 291
  - system reference manual 210
  - system software 48–9
- task allocation table 286
- task identification 286
- technical language, accessibility 292
- test data 140, 141, 207, 222–36
  - for testing algorithms and coded solutions 229–35
  - requirements 222–5
  - test data dictionaries 225–7, 291
  - testing strategy 229
- text colour 198
- text legibility 197
- text spacing 197
- third-generation languages 51
- toggling 202
- toolbars 203
- top-down design 95–7
- top-down testing 234–5
- touch screen 34
- touch-typing 7
- tracing 178
- trackball 34
- translation methods 54–5
- trouble-shooting guides 210
- Turing, Alan Mathison 273
- tutorial assistant 212
- unguarded loop 131–2
- uppercase text 197
- user documentation 208–12
- user feedback 240
- user friendliness 13–14, 240, 292
- user interface 13–14, 193
- upgrade 249–50
- user interface development 193–207
- consultation with users 193–4
- user’s and developer’s perspectives 194–5
- user manual 210
- utility software 48, 49
- validation (data) 188–9, 222
- validation (design) 236
- variable declaration 171
- verification 236
- video input devices 35
- virtual memory 42
- visual display devices 36–7
- walk-through team 238–9
- white-box testing 231
- Windows 203–4
- work routine 6–7
- Zuse, Konrad 69

## Acknowledgments

The author and publisher would like to thank the following for granting permission to reproduce the copyright material in this book.

Australian Picture/Corbis Bettmann, pp. 221; Kay McNulty Mauchly Antonelli, p. 247; Malcolm Cross, pp. 5, 15, 21, 31, 33, 44, 45, 47, 193, 251 (top); Epson, p. 35; IBM, p. 153; LMT/Lockhead, Martin, Tenex/Malcolm Cross, p. 195; Logitech, p. 11; Mary Evans Picture Library, pp. 3, 29; Microsoft, p. 10; Northside Productions, pp. 40 (top), 53; Swe-TECH, p. 38; Video Bytes, p. 34; Volgren, p. 71; Horst Zuse, p. 69, Wacom, p. 34 (bottom).

Every effort has been made to trace and acknowledge copyright. The authors and publisher would welcome any information from people who feel they own copyright to material in this book.