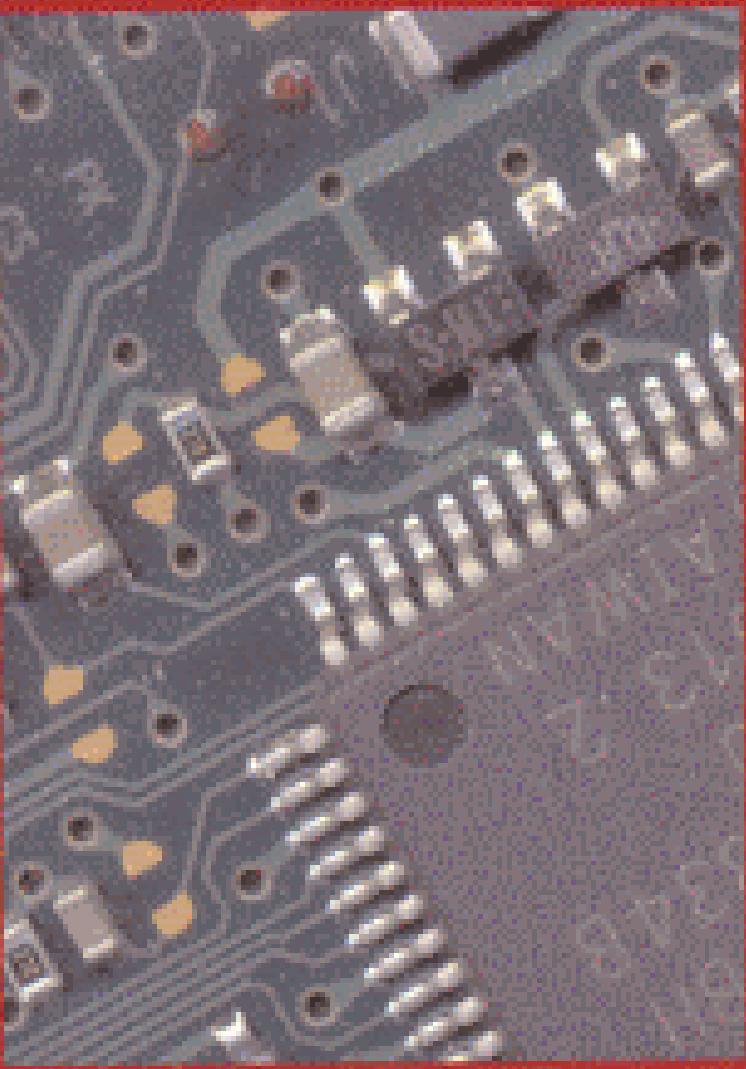


Computer Science

Java Enabled



2nd Edition

INTERNATIONAL
BACCALAUREATE
EXAMINATIONS

International Baccalaureate
Examinations

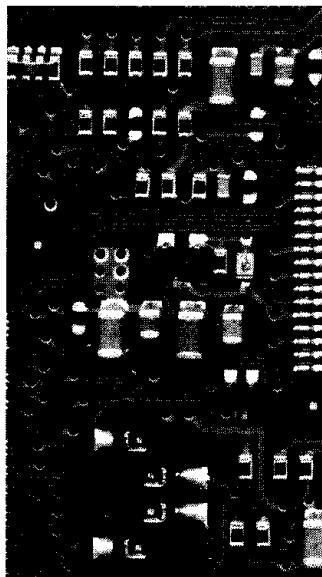
Andrew Meyers
Richard Jones

INTERNATIONAL BACCALAUREATE

COMPUTER SCIENCE

2nd Edition

Java Enabled



Richard Jones:
Cambridge, New Zealand

Andrew Meyenn:
Wesley College, Melbourne,
Australia

Copyright ©IBID Press, Victoria. First published in 2004 by IBID Press, Victoria,



Published by IBID Press, Victoria.

Library Catalogue:

Jones, R & Meyenn, A.

1. Computer, 2. International Baccalaureate. Series Title: International Baccalaureate in Detail

ISBN: 187659041

All rights reserved except under the conditions described in the Copyright Act 1968 of Australia and subsequent amendments. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the publishers. First impression 2002.

While every care has been taken to trace and acknowledge copyright, the publishers tender their apologies for any accidental infringement where copyright has proved untraceable. They would be pleased to come to a suitable arrangement with the rightful owner in each case.

All possible endeavours have been made by the publishers to ensure that the contents of this resource are correct and appropriate. However, the Publishers accept no responsibility for any errors made by the authors. The contents and materials are solely the responsibility of the authors.

This book has been developed independently of the International Baccalaureate Organisation (IBO). The text is in no way connected with, or endorsed by, the IBO.

Cover design by Adcore.

Published by IBID Press, 36 Quail Crescent, Melton 3337, Victoria, Australia

Printed by Shannon Books, Victoria, Australia.

ANDREW'S DEDICATION

Dedicated to my wife Cathie and my daughter's Lisa and Nicola for their love and to my father and mother for their support.

RICHARD'S DEDICATION

Dedicated to the memory of Marion Jones, my mother and my first teacher.

PREFACE TO SECOND EDITION

Richard Jones became interested in computer programming while studying for his M.Sc. in Marine Earth Science at University College, London in the 1970s where there was a requirement for all postgraduate students to take a FORTRAN IV course and produce programs using offline card punches. He currently lives in Karapiro Village, Cambridge, New Zealand. Richard has been teaching IB Computer Science for 13 years and also holds an M.A. Ed. from Bath University, UK and a Postgraduate Certificate in Online Education from the University of Southern Queensland.

Andrew Meyenn began studying computer science in 1971 and began teaching computer science at Dickson College in Canberra, Australia in 1977. In 1982 he moved to the computer industry. He is currently head of the LT. learning area at the Prahran campus of Wesley College, Melbourne, Australia. He has taught the IB computer science course since 1994. He has worked at the University of Melbourne in the Information Systems department. Andrew holds the degrees of M.Ed. and M.Sc. from the University of Melbourne. Andrew is also Deputy Director of the Australian Institute of Computer Ethics.

Both authors are examiners in the IB computer science and Richard has run a number of IB computer science workshops for teachers in his role as Deputy Chief Examiner.

Richard would like to acknowledge the support over the years of Glen Martin, former IB Chief Examiner for Computer Science and Mike Towers, Director of Educational Computing at United World College of South East Asia in Singapore. Thanks are also due to the class of 2002, UWCSEA for feedback on early drafts of the manuscript. My children James and Vicky (who both passed LB. Computer Science) have been most supportive during the development of the book. Since the first days, my wife Hania has encouraged me to take on projects that I might otherwise have let go - thank you for believing in me.

Andrew Meyenn would like to thank the many students and colleges that assisted in the process of compiling the text. He would also like to express his thanks to his family for their support.

The authors were motivated to write the book to assist teachers across the LB. world and to assist in establishing LB. computer science as a growing subject. There are a number of general computer science texts, but until now there has not been one specifically written for the LB. subject.

The book is based on the LB. computer science syllabus. Teachers should be able to follow the syllabus exactly and this will ensure that all aspects of theory are covered. Exercise and practice questions are included that are of a similar nature to those found in the exams. Answers to the exercises are obtainable as a free download from our website: www.ibid.com.au

The algorithm sections are all written in JETS and all algorithms have been checked and tested with computer programs. The authors, however, realise that mistakes may occur and ask that teachers contact them if there are any concerns. The authors will make the program code available for download as a package from the publisher's website <http://www.ibid.com.au>.

Both authors would like to express their gratitude to the editors at **IBID** Press for their support and helpful advice during the writing of this book.

CONTENTS

| | |
|--|-----|
| Introduction | 2 |
| 1.1 Systems Life Cycle | 2 |
| 1.2 Systems Analysis | 9 |
| 1.3 System Design | 13 |
| 1.4 Social Significance and Implications of Computer Systems | 24 |
| 1.5 The Software Life Cycle | 29 |
| 1.6 Software Design | 31 |
| 1.7 Documentation | 36 |
| 2.1 Program Construction in Java | 40 |
| 3.1 Language Translators | 132 |
| 3.2 Computer Architecture | 140 |
| 3.3 Computer Systems | 161 |
| 3.4 Networked Computer Systems | 171 |
| 3.5 Data Representation | 186 |
| 3.6 Errors | 201 |
| 3.7 Utility Software | 206 |
| 4.1 Number Systems and Representations | 210 |
| 4.2 Boolean Logic | 230 |
| 5.1 Terminology | 254 |
| 5.2 Static Data Structures | 262 |
| 5.3 Dynamic Data Structures | 281 |
| 5.4 Objects in Problem Solutions | 308 |
| 5.5 Recursion | 315 |
| 5.6 Algorithm Evaluation | 320 |
| 6.1 CPU Configuration | 330 |
| 6.2 Disk Storage | 334 |
| 6.3 Operating Systems and Utilities | 337 |
| 6.4 Further Network Fundamentals | 338 |
| 6.5 Computer/Peripheral/Communication | 346 |
| 7.1 File Organisaton | 356 |
| 8 The Case Study | 374 |
| 9.1 The Dossier | 392 |

1

Chapter contents

| | |
|--|----|
| Introduction | 2 |
| 1.1 Systems Life Cycle | 2 |
| 1.2 Systems Analysis | 9 |
| 1.3 System Design | 13 |
| 1.4 Social Significance and Implications of Computer Systems | 24 |
| 1.5 The Software Life Cycle | 29 |
| 1.6 Software Design | 31 |
| 1.7 Documentation | 36 |

INTRODUCTION

This topic is concerned with the development of computer systems from analysis through to maintenance and documentation. Much of the focus is on the similar Software Life Cycle which is considered a part of the overall process.

From an IE point of view, the Computer Science Program emphasizes the problem-solving approach over simply coding solutions. The new curriculum has adopted the prototyping approach as being more likely to succeed with students who often prefer a more practical approach to a theoretical or abstract design methodology.

© IBO 2004 1.1 SYSTEMS LIFE CYCLE

This cycle involves the design and implementation of the complete system including such things as software requirements, hardware requirements and any organizational re-structuring that might be needed.

There are obvious similarities to the software development life cycle. For example, the process is also not linear because systems, once developed, stay in place for many years. They have to be adapted to reflect changes in the way they are used. So we again find that maintenance involves making changes to an existing system, which requires analysis leading to a new design and so on.

© IBo 1.1.1 MAJOR STAGES

The subject guide defines the major stages as: analysis, design, implementation, operation and maintenance. Other terms can be used in the description of this process, but the emphasis should be on its cyclical nature.

Analysis: this stage involves collecting and examining data, particularly about the user's requirements and the flow of data through the existing system. A feasibility report may be produced at this stage.

Design: at this stage the software and hardware aspects have to be clearly defined. File design and selection of suitable data structures and algorithms are important. In an object-oriented design, a modeling language such as UML might be used to define the objects and methods and the way that data flows between them. With hardware properly identified, a more detailed feasibility report can be produced at this stage, including a cost-benefit analysis.

Operation: if the decision to proceed with a solution is made then a number of items will be produced. Among these is a plan for the development of a solution.

The planning of a major information system is a very complex process which is assisted by identifying the discrete tasks which need to be completed. Two methods for this, GANTT and PERT charts are discussed in Section 1.3.

Installation: there are several ways of undertaking the process of getting the completed system up and running. These are discussed further in Section 1.2.7. Once the system is in operation, there will be a further review and, perhaps, bugs will be noticed. This will need further analysis causing the cycle to be repeated.

Maintenance: Bugs (errors), or worse, flaws in the initial design have to be fixed while the program is in use. However, fixing a bug is a dangerous process as other bugs are likely to be introduced. The maintenance process has been said to be "two steps forward, one step back".

©IB01.1.2 DATA COLLECTION IN THE ANALYSIS PHASE

2004

There needs to be a clear picture of what the problem is and this is obtained by data collection. Without thorough data collection, the problem cannot be identified correctly, leading to a poor solution.

Data collection identifies:

- who inputs data to the system.
- what form the data is in.
- any validation that is needed.
- what processing is done to produce the required outputs.

This must be carried out in a thorough and systematic way or some vital aspect of the system may be overlooked.

©IB01.1.3 DATA COLLECTION TECHNIQUES

2004

The first phase of analysis is often termed 'fact-finding'. The classic ways to investigate an existing system are to:

- conduct interviews.
- carry out questionnaires.
- search existing documents.
- search the literature for other solutions to the same problem.
- observe people working with the existing system.

Each of these methods has advantages and disadvantages as outlined in the table below.

| Method | Advantages | Disadvantages |
|--------------------------|--|---|
| Interview | More detailed data can be gathered compared to a questionnaire; interesting topics can be pursued. | Time consuming, not easy to 'classify' or 'quantify' data. |
| Questionnaire | Many people can be reached quickly. Results can be analysed with numerical methods. | People may not respond, questions cannot be clarified as they can be in an interview. |
| Study existing documents | The data required to be input and the outputs produced can be identified. | The documents may not tell the whole story without the people that use them. |
| Literature search | May save work if the problem has been solved efficiently by someone else. | The fine detail of the solutions may not be given. |
| Observation | May be able to find out things not apparent from interview or questionnaire - observations not biased. | Time consuming to carry out; the observer can subtly alter the process. |

It should be apparent from this list that more than one method is usually applied to any given problem.

©IB01.1.4 THE REQUIREMENTS SPECIFICATION

2004

This document describes what the customer and developers expect the system to be able to do and how it will be done. This must include all of the costs of building, testing and running the system - both the hardware and software and the expected time to completion.

The requirements specification will include:

- a list of hardware and software tools that will be needed to produce the solution (see also Section 1.2.1).
- descriptions of the functions of hardware and software in the completed system.
- a formal agreement on performance of the system.
- list of personnel and the tasks which will be allocated to them.

Following this, an organisational chart, such as a GANTT or PERT chart can be used to track progress.

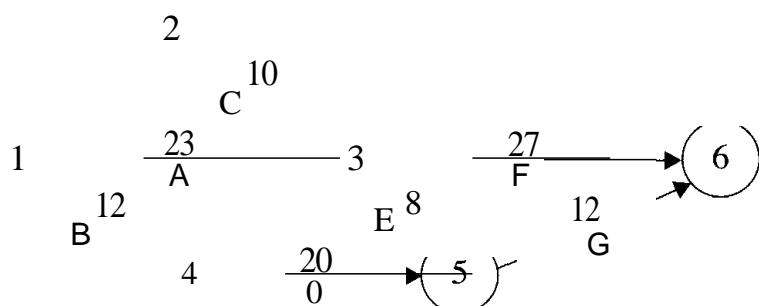
A GANTT chart is merely a list of activities plotted against time whereas a PERT chart shows module dependencies as well.

The GANTT chart is useful for small scale projects such as an IE Computer Science dossier and can help students to keep track of deadlines.

| Activities | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------------|---|---|---|---|---|---|---|---|---|----|
| Problem statement | | | | | | | | | | |
| Problem analysis | | | | | | | | | | |
| Solution | | | | | | | | | | |
| Pseudo-code | | | | | | | | | | |
| Programming | | | | | | | | | | |
| Test and debug | | | | | | | | | | |
| Document | | | | | | | | | | |

Project and Evaluation and Review Technique (PERT) shows events as nodes and activities as lines joining them, with an expected time for each activity, typically in days.

Figure: 1.1



The minimum time to complete this project is 50 days (nodes 1,3 and 6). Any other path is shorter. These nodes are said to lie on the Critical Path. PERT charts and calculation of the critical path are not included as part of the IB Computer Science Programme.

© IBO 2004 **1.1.5 FEASIBILITY REPORT**

When the requirements of a new system have been identified during the analysis phase it is possible to produce a report which estimates the cost, identifies any expected benefits, estimates how long the project will take and outlines any potential difficulties. This is known as a feasibility report and, on the basis of this report, a decision will be taken as to whether to proceed to detailed design.

If this decision is taken, the detailed requirements specification (above) will be produced. The detailed requirements can be used to give an improved estimate of costs, benefits and difficulties and this is also called a feasibility report. So a feasibility report can be produced during analysis, design or both.

The report should also include an analysis of any risks (cost or technical) associated with the project and should review other solutions which may have been produced in the past (these can be found by a literature search). The level of detail in the report will depend upon how much detail has been put into the design. If a detailed design has already been made, prototype screens can be produced to show the user what the final system will look like. Prototyping is discussed further in Section 1.5.

© IBO 2004 **1.1.6 ALTERNATIVE SOLUTIONS**

There are usually a number of different ways in which a given problem can be solved and this may involve non-computer as well as computer systems solutions. One of the primary considerations during development will be the output that the system produces. Output can be presented in many forms as discussed in Chapters 3 and 8. Similarly a range of methods to collect and input data can be used including the use of different interfaces (GUI, CLI) also described in Section 1.3.6.

Other considerations might be to use a stand-alone or networked computer system. The software itself could be one of the following three main types:

- General applications software; this includes word processors, spreadsheets, databases and other 'office' packages which may be integrated. They are the least expensive option but may not have all the features the development requires.
- Specific applications packages may have been written for the business; examples include school administration, video tape rental outlets and medical practices. These packages are more expensive than general applications packages and provide features that these organisations will typically use.
- Tailor made software can be customised to do exactly what the customer requires. They have to be written from scratch and therefore will be the most expensive and time-consuming option.

CLASS ACTIVITY

A family company runs a small store that already has a computerised stock control system using a single **POS** terminal. They are now considering expanding into home delivery. They have considered three systems:

1. A manual system; the customer telephones the store to place an order which is written down on a record card. The order is assembled and delivered with a copy of the card and the customer pays cash or with a cheque.
2. The existing system is used; when a customer telephones an order the goods are entered into the POS terminal as if they were sold over the counter. The receipt is used to assemble the order which is delivered as before.
3. A new system connected to the internet; customers can fill in an online order form and pay by credit card. The goods are assembled and delivered with a copy of the online form.

Discuss the methods of fact-finding that you would consider most appropriate at the analysis stage.

Compare these three systems considering:

- a) The probable costs involved in purchasing hardware and software.
- b) Requirements for extra terminals/network connections.
- c) The time required to develop each solution.
- d) Any possible effects on staff at the shop (see also Chapter 3).

Assemble your findings in a feasibility report for a technical audience.

Create a presentation illustrating your proposed solution which is suitable for the store owner.

Explain why more than one cycle of analysis and design might be needed.



© IBO 2004 **1.1.7 SYSTEM TESTING**

Students are expected to be able to discuss different methods of system testing. The consequences of improper testing depend on what the system is supposed to do but obviously there can be serious implications for an organization if the testing is not done properly. Perhaps it is not too severe if a stock check system for a small shop fails; the users can always resort to counting items by hand. However an air traffic control system presents the other end of the spectrum: failure is not an option!

Systems usually go through several stages of developmental testing. In some development methods users test the software at different stages of its development - a group of programmers may examine an early version to see how it performs (alpha testing). Beta testing is used when the product is nearly ready for release and the developers believe that there are few or no errors. The software is released to users outside the company who can try it out in a range of different environments. Credible companies always let the users know that the product is at the beta stage and may not perform perfectly (see, for example, <http://www.winzip.com>).

There are formal methods of testing which attempt to 'prove' that software actually works using

theoretical or mathematical techniques. Any software above the very simplest programs you will have produced in the first month of your course is too complex to be 'exhaustively' tested (see below).

The process of testing a program involves both functional testing and testing with different types of input data.

Functional testing involves describing systematically what is supposed to happen when buttons are pressed on an event driven interface or menu choices are selected. If an AddRecord choice is made, does the program go the subprogram dealing with adding records? Another way of verifying an algorithm will actually work is to trace (or 'dry run' or 'desk check') it.

Test data is frequently categorised in the following ways. Suppose that there is a simple program that accepts a person's percentage in an exam and gives an output that they have passed if the percentage is greater than or equal to fifty or otherwise prints a 'fail' message. We can test that input with:

Normal Data such as 23 or 56 will check to see if the pass and fail messages are properly delivered. **Data at the Limits** should also be checked, for example 0, 49, 50, 100 are all examples of normal data at the limits as defined by our problem description.

Extreme Data will be outside the normal limits, -1, 104, 122 are examples. The user may not type in such data because they're dumb, it's easy to hit a key accidentally or twice by mistake.

Abnormal Data will be the type of input data we really didn't expect (in this case it could be data that looks like a string and not an integer). A user may enter 'three', which seems unlikely, but they could also hit the spacebar by mistake and enter '3 5', for example.

Thus a test plan for this program might be made up as follows:

| Test data | Type | Expected response | Actual response | Debug? |
|------------|------|---------------------------|-----------------|--------|
| 23 | N | Fail message | | |
| 56 | N | Pass message | | |
| 0 | L | Fail message | | |
| 49 | L | Fail message | | |
| 50 | L | Pass message | | |
| 100 | L | Pass message | | |
| 154 | E | Out of range message | | |
| -90 | E | Out of range message | | |
| thirty two | A | Not valid integer message | | |

Notice that columns have been included to describe what happens when the actual testing is done - the process of detecting, diagnosing and correcting errors in a program is known as 'debugging'.

Notice also that this represents systematic testing of entry into just one field, your final dossier program may have scores of data entry points, all of which may need to be tested, so this is not a trivial task.

© IBO 2004 1.1.8 METHODS OF IMPLEMENTING NEW SYSTEMS

More detail of the social implications of the operation of systems and the development of new systems will be discussed in relation to various case studies presented in Chapter 8. These include training and a comparison of different methods of changing to a new system.

When a new system is introduced it will often mean changes in the way things are done. The existing staff will need adequate training to use the new system. This creates a difficulty for the company since these people still need to carry out their regular duties. The method of changeover to the new system has an impact on opportunities for training.

Parallel running involves running the old and new systems together. This way it is possible to confirm that both systems produce the same results and, should the new system develop any faults, no data or 'up-time' is lost. It also provides an opportunity for users to be trained on the new system and any mistakes they make are not too critical as the old system is still running. On the other hand, there could be twice as much work for the operators to do!

With **phased introduction**, parts of the system can be implemented at different times. After each part is tested and confirmed to work, the next part is introduced. This means that training period is extended and also the new system will be introduced over a longer period of time. A similar approach at, for example, a bank would be to introduce a 'pilot system' at one branch to see if there are any problems before introducing it across the entire organisation.

Phased implementation and parallel running are difficult when the new system is a complete replacement for the old one, and there is little overlap between the two.

In this case a **direct changeover or 'big bang'** may be made. In this case the users need to be trained completely to use the new system before the changeover takes place. Clearly there are risks associated with this type of changeover if the new system does not work correctly.

© IBO 2004 1.1.9 MAINTENANCE

The proper maintenance of a system that has been released is expensive and time consuming. As for testing and error correction, if the system has a modular design the processes involved in making corrections to a system that is already running are made easier. The errors (bugs) are easier to locate and fix. This kind of maintenance is known as 'corrective maintenance'.

Periodic reviews of the system will take place, these go through the same process of fact-finding and analysis that was described earlier. The aim is to decide if improvements to the system can be made and what the cost might be in terms of disruption to the operation of the existing system and in financial terms.

The same methods of fact finding that were employed in the analysis stage can be used in the maintenance phase to evaluate the performance of the new system (i.e. questionnaires, interviews, observation, etc). This evaluation can be presented as a performance review and perhaps will indicate opportunities and costs associated with further improvements to the system.

The types of documentation that are needed for computer systems; system (or technical) documentation and user documentation are described in Section 1.6. As was mentioned earlier on in this section, if technical documentation is poorly done, maintenance costs become very high since programmers need to work very hard to understand the design decisions and other factors involved in choosing the given solution.

EXERCISE 1.1

1. A graphic design company produces gift cards for special occasions. They are considering a change from traditional drawing methods to use of a computer graphics application.
 - a. State three types of software the company could use.
 - b. Explain one advantage and one disadvantage of each type of software you identified in a.
 - c. State three ways in which the new system could replace the old one.
 - d. Explain one advantage and one disadvantage of each way you identified in c.
 - e. Explain how the systems analyst could help the artists understand the ways in which the new system could benefit them.

2. A hospital is introducing two new computer systems. One to handle stock control in the kitchens and another to monitor patients in intensive care using a central workstation receiving data from sensors attached to the patients.
 - a. Discuss the advantages and disadvantages of different methods of changing from the old system to the new system in each of these cases.
 - b. Two data items for the stock control system might be the number of goods in stock and the expiry date of the goods. Suggest suitable data for testing the data entry of these items with reasons for each piece of test data.
 - c. State three items that would be included in a requirements specification for either of these systems.
 - d. Compare the use of a CLI or GUI for the patient monitoring system.



© IBO 2004 **1.2 SYSTEMS ANALYSIS**

The purpose of systems analysis is to gain a clear picture of what the existing system does, what data it uses to accomplish its task(s) and what outputs it produces. Without this information, very little effective programming can be done.

© IBO 2004 **1.2.1 IMPORTANCE OF FORMULATING A PROBLEM PRECISELY**

There needs to be a clear picture of what the problem is because:

- there may be a team of people working on the problem - it has to be communicated to all of them.
- the users of the problem must understand exactly what will be done - what changes it implies for them.
- the costs of the project can be calculated and weighed against the (assumed) benefits - these are written down in a feasibility report.
- the system can be tested properly - you cannot test a system if you don't know what it is supposed to do.

© IBO 2004 **1.2.2 ASPECTS THAT MUST BE CONSIDERED**

The first phase of analysis is often termed 'fact-finding'. The classic ways to investigate an existing system are to:

- conduct interviews.
- carry out questionnaires.
- search existing documents.
- search the literature for other solutions to the same problem.
- observe people working with the existing system.

Each of these methods has advantages and disadvantages which have been discussed in section 1.2.3.

The purpose of these investigations is to determine:

- what parts of the problem (if any) can be solved by computer.
- what other solutions have been tried in the past.
- what the costs might be.
- how much time it might take to complete the proposed system.
- roles and responsibilities of different parties in the development and construction of the new system.

In examination questions, candidates can be asked to discuss these aspects (see the Class Activity in Section 1.1.6 and Exercise 1.1)

© IBO 2004 **1.2.3 OUTCOMES OF INVESTIGATION**

When the existing system has been thoroughly investigated and a range of possible solutions has been identified a decision needs to be made as to which system is most suitable or even whether it is worth making a change. Appropriate hardware and software need to be identified for a particular problem.

If the decision to proceed with a solution is made then a number of items will be produced. Among these is a plan for the development of a solution.

This topic is also discussed in relation to feasibility studies (Section 1.1.5).

The planning of a major information system is a very complex process which is assisted by identifying the discrete tasks which need to be completed. See also Section 1.2.4.

© IBO 2004 **1.2.4 PROBLEMS THAT CAN BE SOLVED USING COMPUTER SYSTEMS**

Many problems can be solved by computer, but not all are cost effective or technically feasible.

This may apply to hardware constraints:

- Is it feasible or cost-effective to have a robot replace books on shelves in a library system?
- Will a local shop be able to afford a mainframe computer to handle stock control?

And the software:

- Does a suitable program or application already exist?
- Should an existing system of application forms be replaced by an onscreen data entry system?

CLASS ACTIVITY

Consider teaching and learning as familiar activities (hopefully!). Some parts of the process have been adapted to take account of developments in information technology whereas some parts have stayed the same. The amount of technology-based education clearly varies from time to time and place to place.

In your organization, identify those activities:

- which have been computerized.
- which have yet to be computerized (but might be).
- which are unlikely to be computerized in the near future.

For each of the above statements try to identify some factors that influence which one they fall into (you might try constructing a grid like the following):

| | activity | for the statement | against the statement |
|-----------------------------|--------------------|--|---|
| have been computerized | attendance records | database of students already exists, network hardware in place. | easier to do by hand, cost of hardware too much, teachers not proficient in IT. |
| | | | |
| | | | |
| might be computerized | tests | multiple choice, gap-fill, easy to computerize, large database of questions possible, automated grading useful. | essay type questions more difficult to do, smart (but misguided) students might hack out the answers. |
| | | | |
| | | | |
| unlikely to be computerized | group discussions | 'face to face' involves more senses in the process (important for kinesthetic learners), limited bandwidth available | might be useful if access to internet was greater, could be a supporting process |
| | | | |

These are examples, not meant to be exhaustive or complete. You might like to form small groups to investigate each one and report back.

©IBO 1.2.5 BASIC CONTROL CONSTRUCTS 2004

Computers systems, in general, are used for systems where the required information can be precisely identified, coded into data, captured, processed and output as new and useful information by suitable hardware and software.

Thus the analysis will produce the information requirements of the new system, it will define:

- The input data required
- The processing needed
- The outputs that will be produced

This 'input-process-output' model of computer systems will occur many times throughout this book. These requirements are often presented using a systems flowchart (see Sections 1.3.1 and 1.3.7).

©IBO 1.2.6 MODULAR DECOMPOSITION 2004

The design phase will then examine how these tasks are to be carried out and which parts can be solved using a computer. As an example, consider a library system, the computer can handle the tracking of data about clients and books. It can tell you where books belong, where they actually are and so on. The system will not provide for the physical movement of books from check in desk back to shelves – not because it cannot be done, but because it would be expensive to implement and would interfere with peoples' ability to browse the shelves.

See also, Section 1.6.2 for further examples of modular decomposition.

©IBo1.3 SYSTEM DESIGN

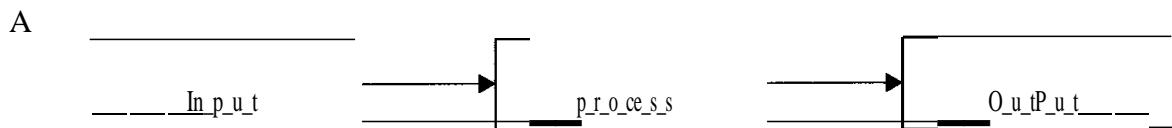
2004

©IBo 1.3.1 PARTS OF A SYSTEM

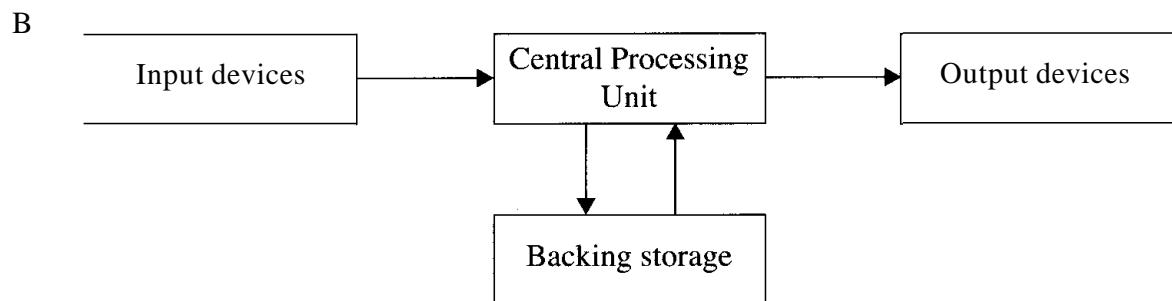
2004

The main parts of any computer system follow the *input-process-output* model of data flow:

Figure: 1.1 A & B

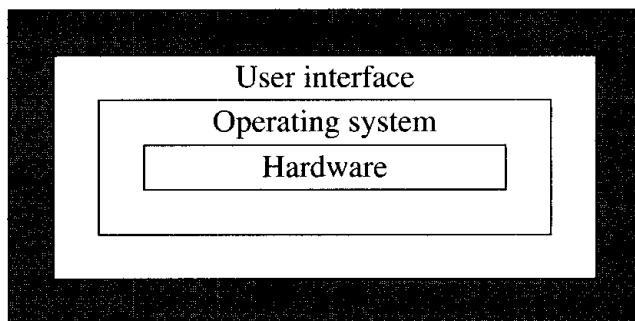


In hardware terms, it is only necessary to add some backing storage since the CPU has only RAM (temporary storage) and ROM (read only storage) in its primary memory:



The user would find it difficult to deal directly with the hardware since all operations at this level are carried out in binary machine code. Therefore, successive layers of software have developed—operating systems (including the user interface) and applications software:

Figure: 1.2



©IBo 1.3.2 DATA IN A COMPUTER SYSTEM

2004

One of the important processes identified in software development (see Section 1.2.2) was analysis and fact-finding. This involves carefully identifying the data which needs to be held and processed by a system.

EXAMPLE

In a bicycle rental system, data can be collected via a manual system, using record cards, for example:

| Bicycle number: | IP201 | | Purchase date: | 18/07/00 | |
|-----------------|------------------|---------------|----------------|---------------|-------------|
| Make: | Elektra | | Value: | \$185.00 | |
| Model: | 18-speed de-luxe | | Hourly charge: | \$4.25 | |
| Date and time | Renter's ID | Date and time | Renter's ID | Date and time | Renter's ID |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

In order to fully describe the system it is necessary to consider what happens under many different circumstances. For the total charge to be computed when a bicycle is returned, the hours of renting need to be calculated and multiplied by the hourly charge.

Often, the data that needs to be held and processed in a system is identified using data flow diagrams. A data flow diagram typically uses the following symbols (although there is not complete consistency in practice):

Figure: 1.3

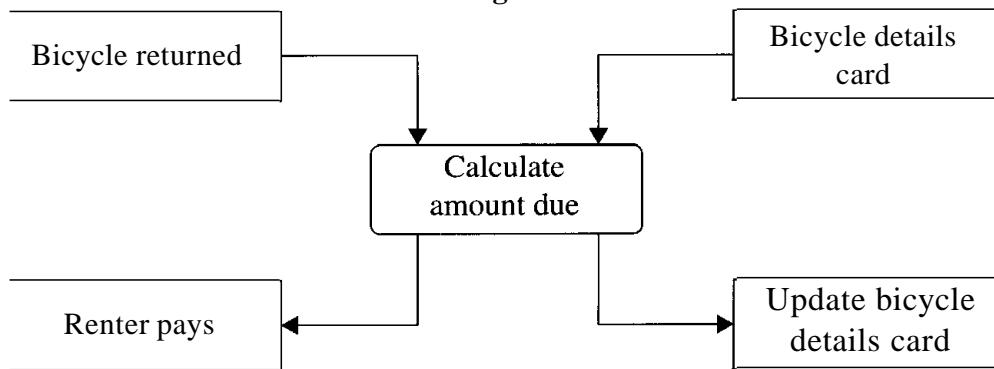
_____ The box with rounded corners (maybe square or rectangular) which represents a process. An example would be the calculation required above.

_____ The box with an open right-hand side representing a data store. This would be the index card for the bicycle.

_____ The closed rectangle is a source or sink (destination) of data. It shows the limits of our diagram, how the data gets into or out of these boxes is not a concern of this diagram.

These boxes are accompanied by arrows to show the direction of data flow. Often the arrows have other associated information such as the data that is on the move or, in other systems, the person responsible for the process.

For a complete picture, consider returning the rented bicycle using the documents and data in the manual system:

Figure: 1.4

No reference is made to hardware used. For example, the calculation may be done with a calculator (or abacus or on the back of an envelope), this does not concern the data flow. However, when a new system is to be created, it is important to know that this process needs to be carried out.

EXERCISE 1.2

1. Construct a data flow diagram showing what happens when a bicycle is rented out.
2. List all the other situations where updating of data may take place in this system.
Even a small system will require several diagrams to describe it completely.
3. Study your existing student registration system. Draw a series of data flow diagrams that illustrate where data is stored, how it is used and when it is updated.
4. Examine the Weather Case Study (IBO 2001). Construct a dataflow diagram showing how a person at a TV station would prepare a weather forecast for presentation.



© IBO 1.3.3 DATA CAPTURE AND PRESENTATION

The data flow diagram shows only data flow without reference to mechanisms of capture and display. There are a great many ways to capture data for use in a computer system. The main devices are described in chapter 2 and their methods can be classified in the following way:

| Input method | Example devices | Example of use |
|-----------------------|---|---|
| Manual data entry. | Keyboard, mouse, joystick, touch screen, touch pad. | Adding client or book records in a library. |
| Direct data entry. | OCR/OMR scanners, MICR reader, barcode scanner. | Lending a book, locating borrower details. |
| Automatic data entry. | Sensors - temperature, sound, pressure, light etc. | Controlling the temperature in the library. |

Software Development

Similarly one can classify output devices in common use:

| Output method | Example devices | Example of use |
|-----------------------------------|---|---|
| Temporary display. | VDU, LCD display, lights. | Showing the price of an item at a pas terminal. |
| Permanent display. | Printers, plotters. | Printing a receipt at a pas terminal. |
| Electrical! mechanical output. | Actuators – relays, switches, converters etc. | Sending credit card details to a bank from a pas terminal. |

There are so many input or output devices that not all of them will fall into a particular classification.

EXERCISE 1.3

Study the Human Evolution Research Case Study (IBa 2002).

1. Identify and describe all of the input and output devices in the case study.
2. Explain why each of these devices is appropriate to a given task.



© IBO 2004 1.3.4 DESIGN OF APPROPRIATE DATA STRUCTURES

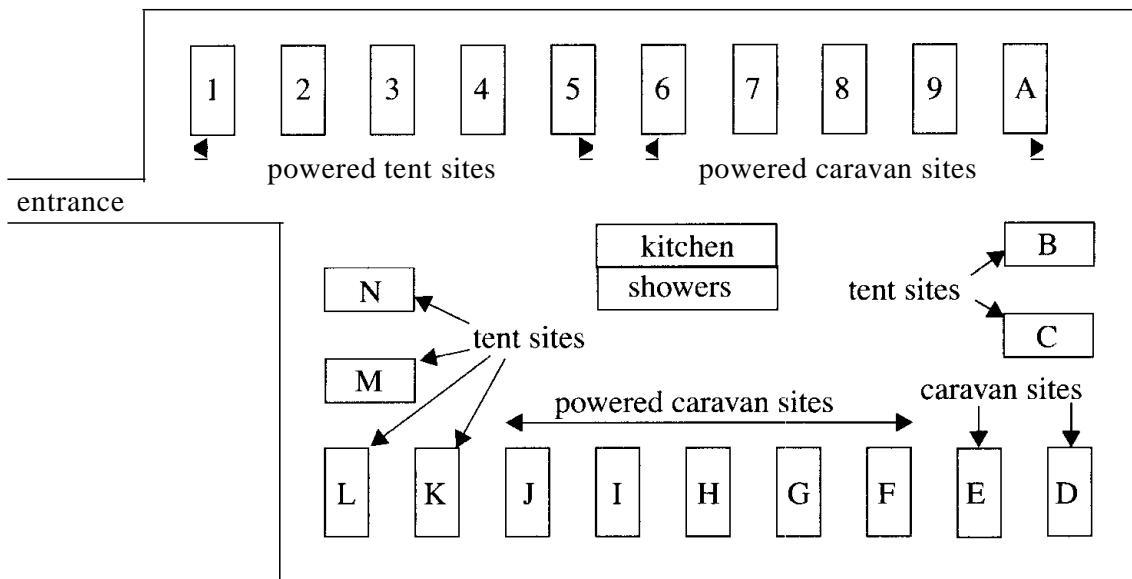
This topic is covered in detail in Chapter 2 where the relevant data structures are described together with examples of when it would be appropriate to use them. Here we present some exercises that can be used once the student is familiar with a range of possible data structures that can be used.

Notice that reference is often made to activities done "in Chapter 2"; this is because this topic would normally be covered after that even though it appears first in the Subject Guide.

In the examination, students are likely to be asked similar, but much simplified questions. The design of appropriate data structures is an important activity for the dossier and students should be able to discuss their choices. It is an excellent idea to keep a written (or web-based) log during the design stage of the dossier including sketches and notes about the problem and possible ways of representing or storing the data for a system.

EXERCISE 1.4

1. A holiday park offers sites for tents and caravans some of which are supplied with power.
To book a space the following system is used:



When a customer arrives the following details are collected and entered onto a form:

- Name.
- Address.
- Vehicle Registration Number.
- Site Allocated.

Information is held about each site in a card index file:

- Site number.
- Powered?
- Type (tent or caravan).
- Daily charge.

Carefully consider the data flow in the following scenarios:

- A new customer arrives.
- A customer leaves.
- A site has its status changed (eg from powered to non-powered).

Draw data flow diagrams to illustrate the above scenarios.

Discuss, including diagrams, the data structures that could be used to hold the data for the system. Remember that the 'discussion' keyword requires you to consider a range of possible data structures and give reasons for selecting the ones you did.



© IBO 1.3.5 HARDWARE COMPONENTS 2004

You have probably already studied a range of input, output and backing store devices from Chapter 3. For each one given there, copy and complete the following table related to the holiday park scenario of the last section. An example is given for you:

| Device | Could be used? | Example of use | Advantages | Disadvantages |
|----------|----------------|--------------------------------|---|---|
| Keyboard | Yes | To enter details of customers. | Easy to enter alphanumeric data such as an address. | Slower than direct entry methods such as a barcode. |

© IBO 1.3.6 USER INTERFACES 2004

Early operating systems operated with typed in commands (requiring command-line interpreters or CLIs) while later ones have developed graphical user interfaces (GUIs). The main features of these interfaces are:

| Command Line Interfaces | Graphical User Interfaces |
|--|--|
| Easier to implement for a programmer, requires less memory to run. Can be run on systems without graphical monitors. | More complex to implement, requires more memory, a pointing device and a graphical monitor. |
| Users need to remember specific commands so new users can find them harder to use. | Icons (small images) help users to remember commands, file types; commands are grouped in menus. |
| Long term users may find it quicker to type in a command at the keyboard than to use a mouse or other pointing device. | New users will find it easier to use because they do not have to remember specific commands. |

Graphical User Interfaces are sometimes described using the term WIMP, variously interpreted as:

- Windows
- Icons
- Menus
- Pointers

- Windows
- Icons
- Mice
- Pull-down Menus

EXERCISE 1.5

1. Explain the advantages and limitations of both GUIs and CLIs for each of these tasks:

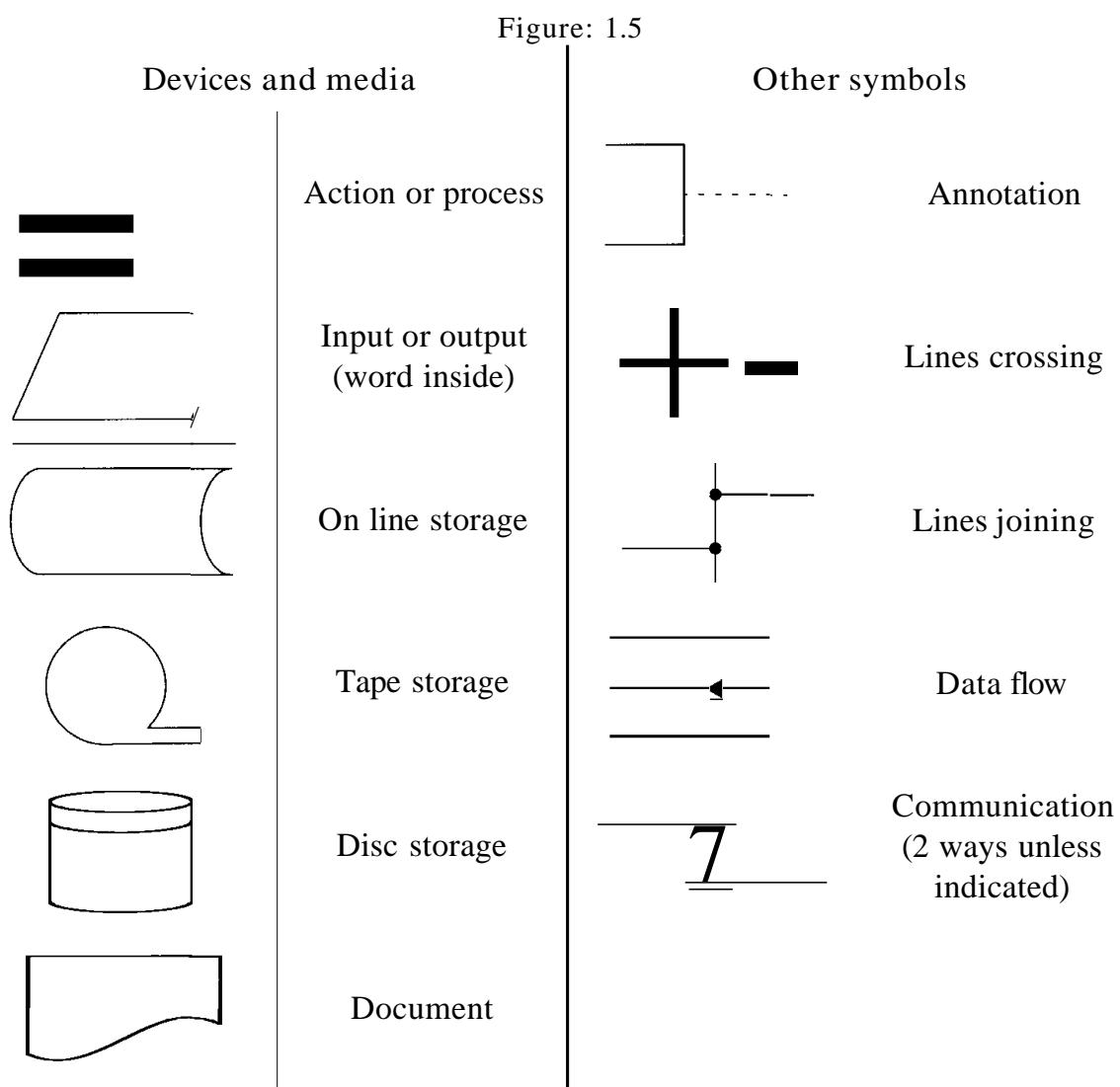
- A researcher preparing images of cloud formations to be posted on a web page.
- A computer technician setting up a task to batch copy a group of files from one device to another.
- A student compiling a directory of Java source files into class files.
- A new user copying a document from hard drive to floppy disc.



© IBO 1.3.7 SYSTEMS FLOWCHARTS
2004

Systems flowcharts are designed to link data flow and processing operations to specific pieces of hardware. They are sometimes known as input-output (systems) flowcharts. They should not be confused with flowcharts used to show the structure of algorithms.

As with data flow diagrams there is a wide variation in symbols used to implement systems flowcharts; below are the ones specified by IE in the Computer Science Subject Guide:

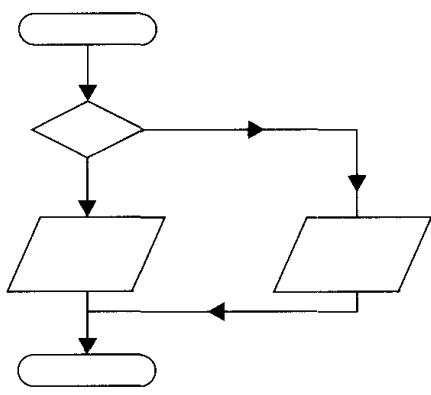


Software Development

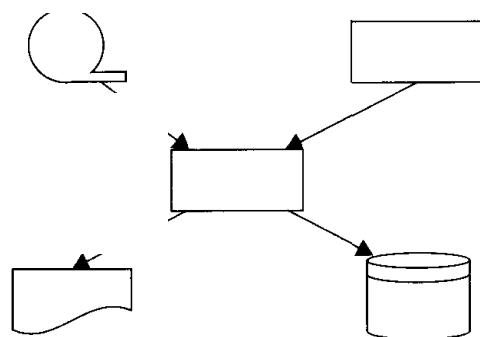
Students generally seem to have great difficulty with systems flow charts; the main problem seems to be that they think in terms of linear algorithm flow charts (I believe there is a conspiracy of subversive maths teachers at work here). The following diagram shows the outline shape of the four different types of chart used in this book:

Figure: 1.6

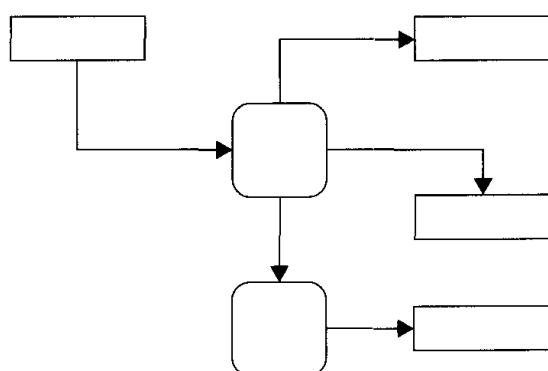
Flowchart



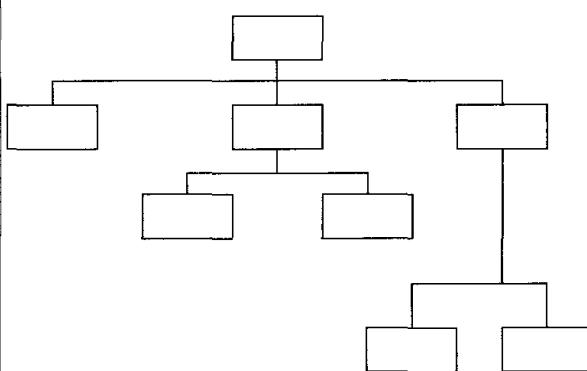
Systems flowchart



Data flow diagram



Module diagram (Structure chart)



Points to note:

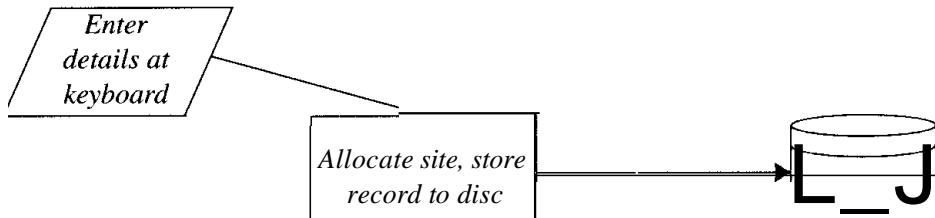
- Flowcharts are used to describe algorithms (although pseudocode is often preferred these days);
 - Systems flowcharts are used to describe input-process-output in computer systems, they are the only charts to refer to hardware devices;
 - Data flow diagrams refer to data objects and processes (people, paper files, computer files, etc);
- Module diagrams are used to split a large problem up into several smaller ones (stepwise refinement). This makes the problem easier to solve and divide up among a programming team.

Simple processes

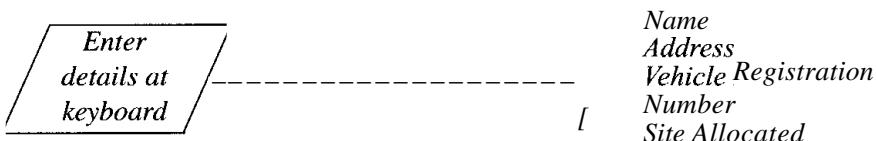
Consider the case of the holiday park described in Section 3.1.5. If this system were to be transferred to the computer, the following tasks (among others) would have to be carried out:

Registering a new customer could involve entering the new customer details at the keyboard, entering the site location and storing these to a database:

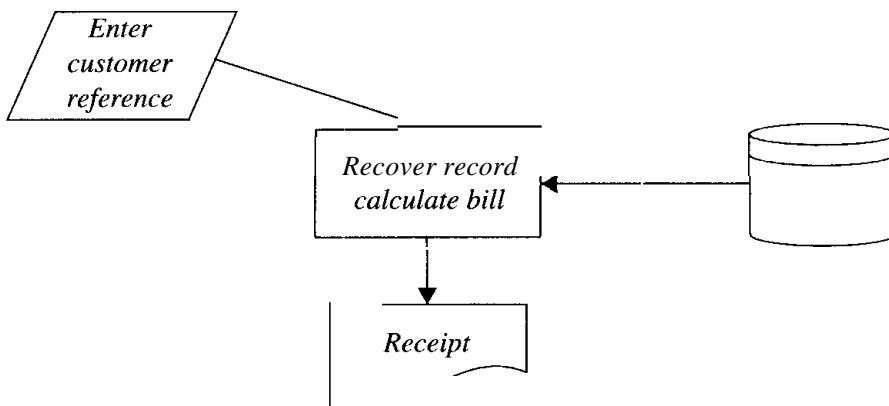
Figure: 1.7 A, B & C



More detail can be added, if required, as an annotation:



The process of checking out a customer would involve calculating the bill and issuing a receipt:



©IBO 1.3.8 CONSTRUCTING SYSTEMS FLOWCHARTS 2004

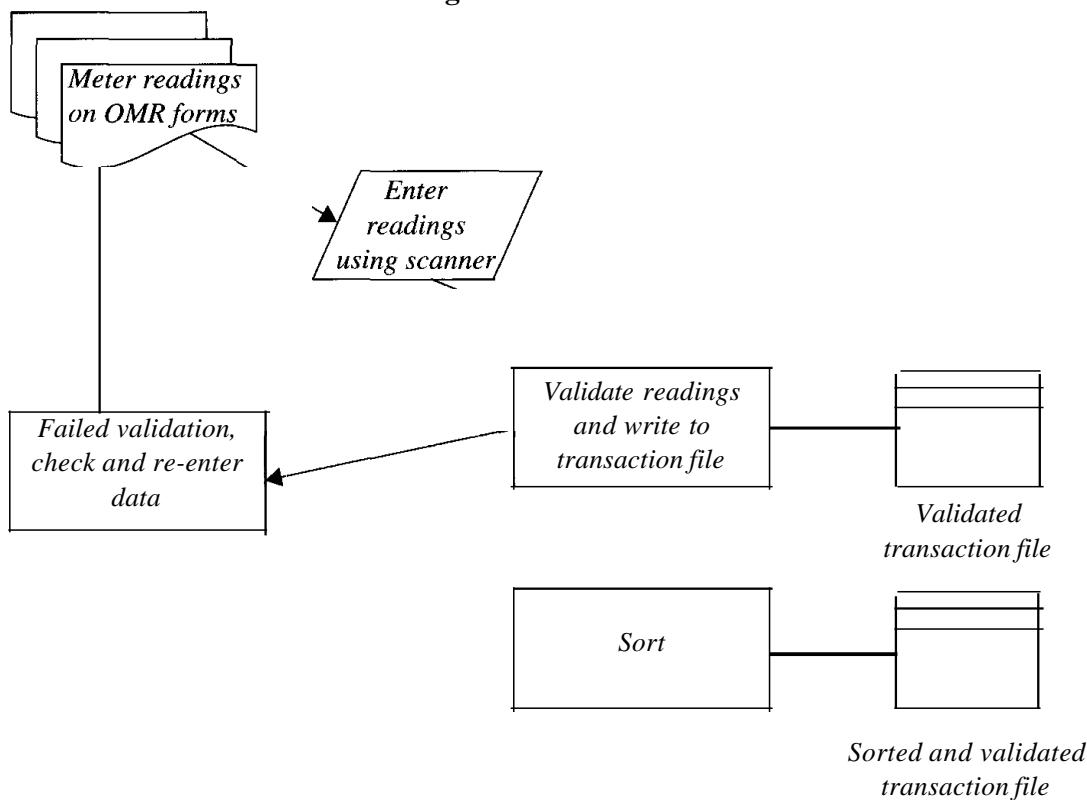
Chapter 3 describes batch, online and real-time systems; here we examine how these types of process are represented in systems flowcharts.

Common batch processing tasks

In batch processing, data is gathered first and then processed in one go. Typical operations update a master file using a sorted transaction file. Therefore, in many batch processes (cheque clearing, electricity billing, payroll processing, batch update of a stock file), paper documents will be collected, validated and sorted. Items rejected by validation may be corrected and re-entered:

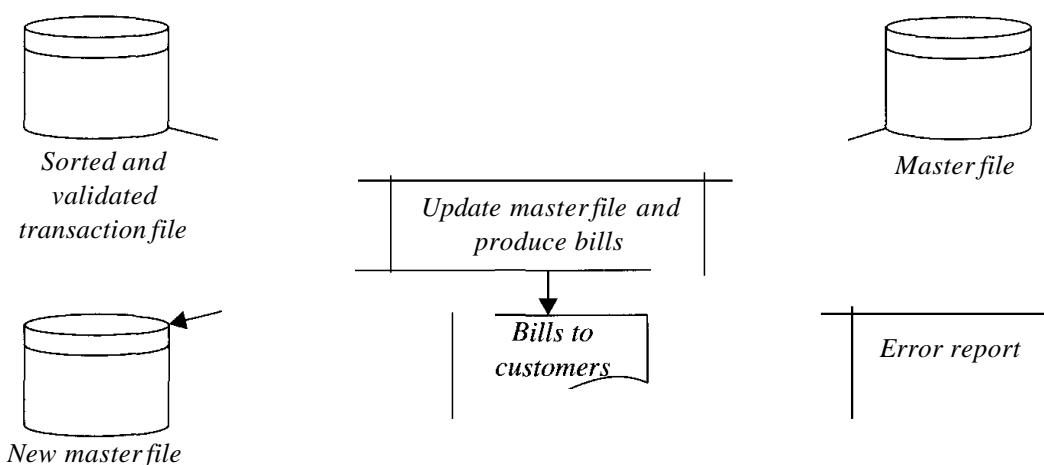
Example for electricity billing:

Figure: 1.8



The sorted transaction file is then used to calculate the amount of electricity used (by subtracting the reading in the master file from the reading on the transaction file) then to produce a bill and to update the master file with the new reading. Of course some errors may still occur. If the meter reading is incorrect, bills can be very large or perhaps negative. These errors are typically recorded on a hard copy print out.

Figure: 1.9



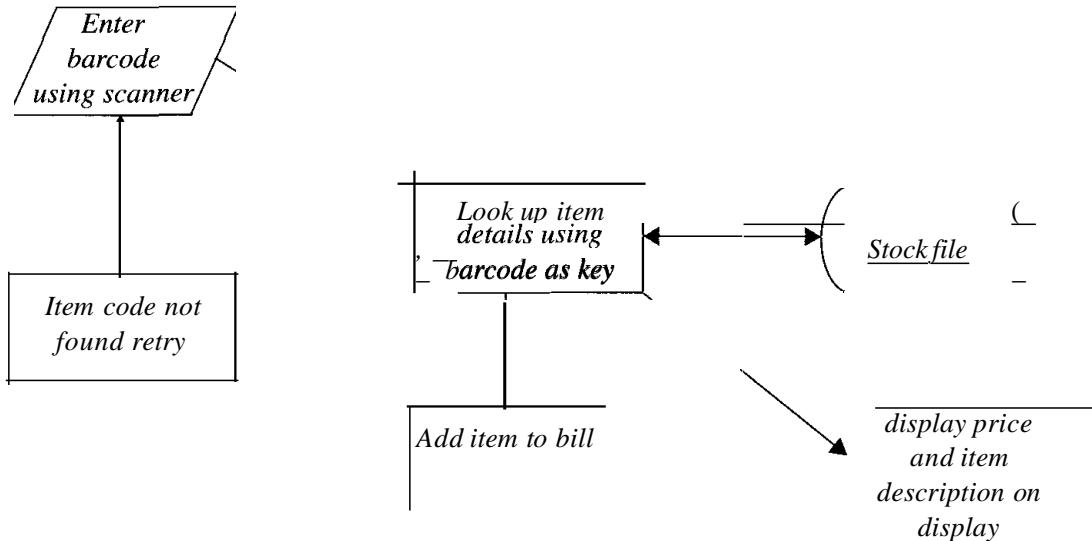
Activity

You will have studied cheque processing and payroll processing systems in Chapter 3. Construct a systems flowchart for each of these systems.

Common online processing tasks

Recall that, in online processing, any transactions are used to update a database immediately. A typical example is supermarket stock control where barcode scanners at a POS terminal read the barcode, look up the item details in a stock database and return the details to the POS terminal where they are printed on a receipt and shown on a display.

Figure: 1.10



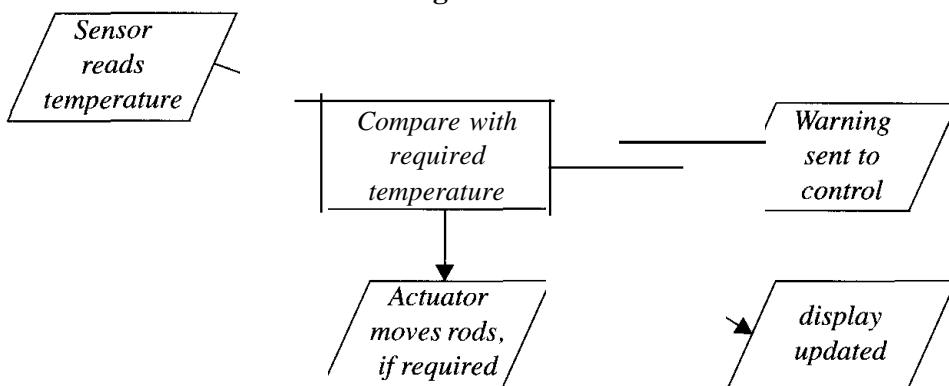
Activity

You will have studied airline reservation and library cataloguing systems in Chapter 3. Construct systems flowcharts for these processes.

Common real time processing tasks

Real time systems are a type of online processing system in which the processing is fast - the input data is processed quickly enough to affect the next output of the system. Usually such systems collect their data through sensors (automatic data entry). A typical example is a system which monitors a nuclear plant's reactor core. In some reactors a set of rods are inserted into the core to damp the nuclear reaction. When this is done, a warning is sent via a communications system to the control room. This situation might be represented as follows:

Figure: 1.11



Activity

You will study air traffic control and patient monitoring systems in Chapter 3. Construct systems flowcharts for these processes.

©IBo1.4 2004 SOCIAL SIGNIFICANCE AND IMPLICATIONS OF COMPUTER SYSTEMS

This is a potentially huge topic to be covered in only 5 hours of teaching time. However, in principal, these topics are only examined in the context of the Case Study. Of course, as computer teachers, we would be failing in our duties if we did not at least alert students to the potential social consequences of the increasing use of information and communications technology.

©IBo 1.4.1 2004 SOCIAL AND ECONOMIC IMPLICATIONS OF INSTALLATION OF NEW SYSTEMS

Section 1.1.8 has already covered some aspects of the installation of new systems and the different ways in which this could be carried out.

The introduction of computers into an organisation has significant consequences for employment. One obvious case is the loss of work caused by the use of computers. Perhaps less obvious are other forms of cost saving such as reducing the amount of time a patient spends in a hospital or a customer spends in a supermarket. This has benefits as both organizations can then serve more patients/customers using the same facilities.

On the negative side, there are losers in this shift to increased efficiency and increased use of sophisticated equipment. If people and societies are unable to adapt to change, or unable to gain training in newer technologies, many jobs will inevitably be lost. The introduction of ATMs and WANs in banking has lead to a decrease in staffing at banks and even a reduction in the number of bank branches.

Opposed to this is the need for people to develop the software and build the hardware needed for new computer systems. However, such people usually have different qualities and training compared with those losing their jobs. This may promote divisions in society between poorly qualified (and less employable) people and better-educated ones. These divisions also extend to communities and countries.

In relation to the specific methods of changeover given in Section 1.1.8, we could summarize the social and economic implications as follows:

| Method | Social Implications | Economic Implications |
|---------------------|--|---|
| Parallel Running | More work for employees or they may have to work longer hours to cope with the extra work. | Staff may have to be paid more or extra staff employed. Mistakes less critical for the business. |
| Phased Introduction | More hours to work in this case also. Different people may be affected in different ways at different times - could cause dissatisfaction. | Costs may still be associated with extra work. Longer period of changeover; difficulties with system may become drawn out - the company's reputation for efficiency may suffer. |

| Method | Social Implications | Economic Implications |
|-------------------|--|--|
| Direct Changeover | Staff may feel under great pressure/stress during this type of changeover. Customers may also dislike changes - at least initially. | As with the other methods there are training costs to be considered. The cost of hardware/software may also have to be paid at one time rather than spread out over time. Always a risk of failure turning away potential customers. |

©IBO 1.4.2 SOCIAL SIGNIFICANCE AND IMPLICATIONS OF THE WIDESPREAD USE OF COMPUTERS

In the context of the appropriate case study the following issues could be considered. A point to remember in examination questions is to always try to see both sides of the question, especially if you are expected to 'Explain the implications' of a particular issue. Simply repeating sentences from the case study will seldom work.

Economic consequences

Competition for goods and services among countries in the increasingly global economy leads to a situation where countries using computer systems to build cars, for example, can offer cars for sale at lower prices than domestic competitors can manage. This, in turn, could lead to further unemployment in the less competitive country. In the case of human evolution research we see that universities with advanced scanning systems can sell their digitised graphic images whereas universities using manual techniques (possibly located in developing countries) cannot.

A further consequence of the development of computer systems as communicating devices has been the move to teleworking and teleconferencing. Where a business deals with information and services there is no need to bring people together to work and to attend meetings as they can do so from home. This has the potential to make huge changes to our towns, cities and lifestyles.

Not exactly a direct consequence, but one industry that has sprung up in certain parts of the world is that of copying software - software piracy. This is of great concern to major corporations and individual programmers alike. Many people put considerable time and effort into the development of software and would like to be paid fairly for their effort. Piracy is simply theft of those ideas, as prospective computer programmers this is an argument you should agree with!

For counter-arguments in which software is developed by a community for the good of all, see <http://www.opensource.org> many excellent applications which now large parts of the internet (Apache, PHP, Mozilla and the operating system Linux) have been developed as (free) community projects.

Political consequences

The OECD (Organisation for Economic Cooperation and Development) has proposed a number of principles which should apply to data held about people:

Collection Limitation Principle. There should be limits to the collection of personal data and any such data should be obtained by lawful and fair means and, where appropriate, with the knowledge or consent of the data subject.

Data Quality Principle. Personal data should be relevant to the purposes for which they are to be used, and, to the extent necessary for those purposes, should be accurate, complete and kept up-to-date.

Software Development

Purpose Specification Principle. The purposes for which personal data are collected should be specified not later than at the time of data collection. The subsequent use should be limited to the fulfilment of those purposes or such others as are not incompatible with those purposes and as are specified on each occasion of change of purpose.

Use Limitation Principle. Personal data should not be disclosed, made available or otherwise used for purposes other than those specified in accordance with Paragraph 9 except:

- a) with the consent of the data subject; or
- b) by the authority of law.

Security Safeguards Principle. Personal data should be protected by reasonable security safeguards against such risks as loss or unauthorised access, destruction, use, modification or disclosure of data.

Openness Principle. There should be a general policy of openness about developments, practices and policies with respect to personal data. Means should be readily available of establishing the existence and nature of personal data, and the main purposes of their use, as well as the identity and usual residence of the data controller.

Individual Participation Principle. An individual should have the right:

- a) to obtain from a data controller, or otherwise, confirmation of whether or not the data controller has data relating to him;
- b) to have communicated to him, data relating to him:
 - within a reasonable time;
 - at a charge, if any, that is not excessive;
 - in a reasonable manner; and
 - in a form that is readily intelligible to him;
- c) to be given reasons if a request made under subparagraphs(a) and (b) is denied, and to be able to challenge such denial; and
- d) to challenge data relating to him and, if the challenge is successful, to have the data erased, rectified, completed or amended.

Accountability Principle. A data controller should be accountable for complying with measures which give effect to the principles stated above.

The increasing amount of personal data held about citizens on governmental and private computer systems has lead to considerable legislation on data protection based on these guidelines.

One of the worries about databases is the potential for linking them together to get a highly detailed profile of an individual. Unscrupulous governments could use such data to spy on their political opponents. Another is the accuracy of data held on computer records can get mixed up and miscarriages of justice have occurred because of this in several countries of the world.

Much of this topic was covered by the 2003 Case Study.

Cultural consequences

One of the effects of the growth of computer systems has been the spread of the English language via software and operating systems. With language comes culture and cultural bias (some would say cultural 'imperialism'). After all, not all cultures read text from left to right or top to bottom of a page.

The computer also offers much in the way of passive entertainment so there are fears that young people spend too much time online or playing computer games. Of course, the internet offers an ideal way for people to find out more about other cultures and to 'meet' people from other countries online in chat rooms.

Environmental consequences

Computers use large amounts of electrical energy, although the main power drain does come from the monitor and two developments have helped reduce this. The first is the use of power settings which allow the monitor to 'sleep' after a short time interval. The second is the increasing affordability of low power consumption LED screens.

Although computers have reduced the amount of paper used in some offices, in general, the use of paper has dramatically increased (look around your own school for evidence of this - which department uses the most paper?). Even when students (and teachers) are careful about printing (we hope) a great deal of paper still ends up in the recycle bin. The environmental cost of paper use is not so much the rainforests or other trees (most computer paper is pulped from fast-growing eucalyptus plantations) but the high energy cost of paper mills and their extensive use of chemicals such as bleaches. Printing also uses ink, and in the case of laser printers, toner cartridges. In schools and colleges there is hope that reasonably priced palmtop computers using wireless networks could replace paper worksheets in the near future. Many institutions in developed nations are already transferring their course notes to the internet or a school intranets (internal networks).

As mentioned above, teleworking and teleconferencing are growing. Since less movement of people is needed, there are environmental benefits (marginal, at present) from reduced automobile or airplane journeys.

CLASS ACTIVITIES

1. Make a list of all the databases where information about you might be found.
2. Investigate your school administration system. What data does it hold about you? How closely does it meet the criteria laid down by the OECD?
3. For each of the case studies, list and discuss the economic, political, cultural and environmental issues involved. When discussing the implications, remember to include all sides of each issue.

This is another activity that can usefully be carried out in small groups.



© IBO 2004 1.4.3 CURRENT TRENDS AND THEIR CONSEQUENCES

Some of the current trends in computing that can be identified are:

- More power in smaller boxes.
- The introduction of microprocessors in many devices.
- The convergence of mobile computing and communications.
- The further development of software (interfaces and agents).

Ever smaller components mean that computer systems become more powerful. The development of the laptop and palmtop personal computer has made it possible for people to work and communicate while on the move. In some cases this has radically changed the way people work. A salesperson for a company typically used to travel the country tasking orders for goods which would then be recorded on paper and taken back to headquarters for processing. Now, if a customer asks about the range of goods, the salesperson can use a laptop computer to connect to the company's mainframe where the inventory (database of goods in stock) is held and say exactly what is available and by when it can be delivered. Current trends have seen companies make some facilities of their own systems available directly to clients' computers so that, for example, the client can themselves browse available inventory and place an online order (so-called B2B systems). One consequence might be a change in emphasis or even loss of job for the sales representative.

Of course, components can only shrink in size so much. Maybe it is possible to make a computer the size of a thumbnail. However, the interface would be somewhat cramped and impracticable however (at the present level of technology, that is!).

We are now used to the idea that many everyday devices are 'computerized' and thus programmable. This applies to older devices like washing machines, watches, cameras and domestic heating/cooling systems. However, some devices in current use would not be possible without the use of microprocessors - pocket calculators, ABS, pacemakers, mobile phones weighing a few ounces and digital cameras are some examples. Some would argue that the use of calculators has lead to a decline in maths skills and the widespread use of mobile phones is more of a cost than a benefit, but it is always worth remembering that the technology itself is neutral. Casual use aside ("I'm on a train"), the mobile phone has brought benefits to people in less accessible communities where the cost of installing and maintaining conventional copper cable would be high.

Current trends would suggest that it will not be too long before most domestic devices are fitted with microprocessors and plugged into networks (wireless or otherwise) so that they can be controlled remotely. This way - the classic scenario has it - the refrigerator calls the dairy products supplier whenever your milk carton approaches an empty state. Cynics again may be more likely to predict a General Protection Fault that causes the fridge to fail corrupting all your goodies. Certainly, for such a scenario to come to pass, we can say that both software and hardware will need to increase in reliability even as they increase in complexity.

The main developments in software will hopefully be a more user friendly interface and software that 'knows us better'. Compare the amount of training required to use today's computers with that of using a mobile telephone. The telephone interface has evolved over a number of years and is now quite 'user friendly'. It seems that the natural language interface and spoken commands may, at last, be getting to the point where they are less trouble than the traditional keyboard and mouse. We have to be a bit careful though as computer programmers thought that this problem would be solved long, long ago.

The idea of the software 'agent' has been attracting much interest. The vision is of a computer program that works with you and understands your computer needs. It tracks your emails and sends other programs out over the net to see if there are any sites matching your known preferences. It keeps track of all the files you have ever created and can tell you where you stored them (wouldn't that be nice!).

Further reading

It is, of course, impossible to cover all the developments and their possible consequences in detail. Teachers and students would do well to read a selection of articles in computer magazines and even general interest magazines such as *The Economist*, *Time* and *Newsweek* which often cover current trends in computer systems. Such articles provide a useful framework for discussion of this topic. In principle, however, in examination papers these topics will only be discussed in relation to the appropriate Case Study.

© IBO
2004

1.5 THE SOFTWARE LIFE CYCLE

This topic is concerned with the way that software applications are developed and the way in which that process forms part of the larger cycle of systems development.

Students are expected to undertake a dossier project which may typically run to somewhere between 60 and 100 pages of code and documentation as part of the course. During this process they are expected to utilize the commonly accepted tools of software development and to gain some understanding of the way in which problem solving in this domain can be approached.

From the IE point of view, the program emphasizes the problem-solving approach over simply coding solutions which might be the focus of other types of Computer Science program at this level.

There are many models of the systems development and the similar software development process. Both these processes have a cyclical nature and the software development cycle can be viewed as a stage of the systems life cycle.

© IBO 1.5.1 OUTLINE THE MAJOR STAGES IN THE SOFTWARE LIFE CYCLE

There is no general agreement on terms so we have shown those mentioned in the program guide and some others in common use:

| Terms in the Subject Guide | Related terminology | Related sections of this chapter |
|-----------------------------------|--|---|
| Analysis | | 1.2, 1.3, 2.5, 2.6 |
| Problem statement | System requirements or software requirements | 1.5.1, 1.2.4 |
| Design | Modular design or detailed design | 1.3, 1.5 |
| Construction | Integration, development | Chapter 2 |
| Testing | System testing, acceptance testing | 1.2.8, 2.2.12 |

| Terms in the Subject Guide | Related terminology | Related sections of this chapter |
|----------------------------|---------------------|----------------------------------|
| Installation | Implementation | 1.2.7, 1.6.2 |
| Maintenance | | 1.2.9,1.6.1 |
| | Obsolescence | 1.2,1.3,2.5,2.6 |

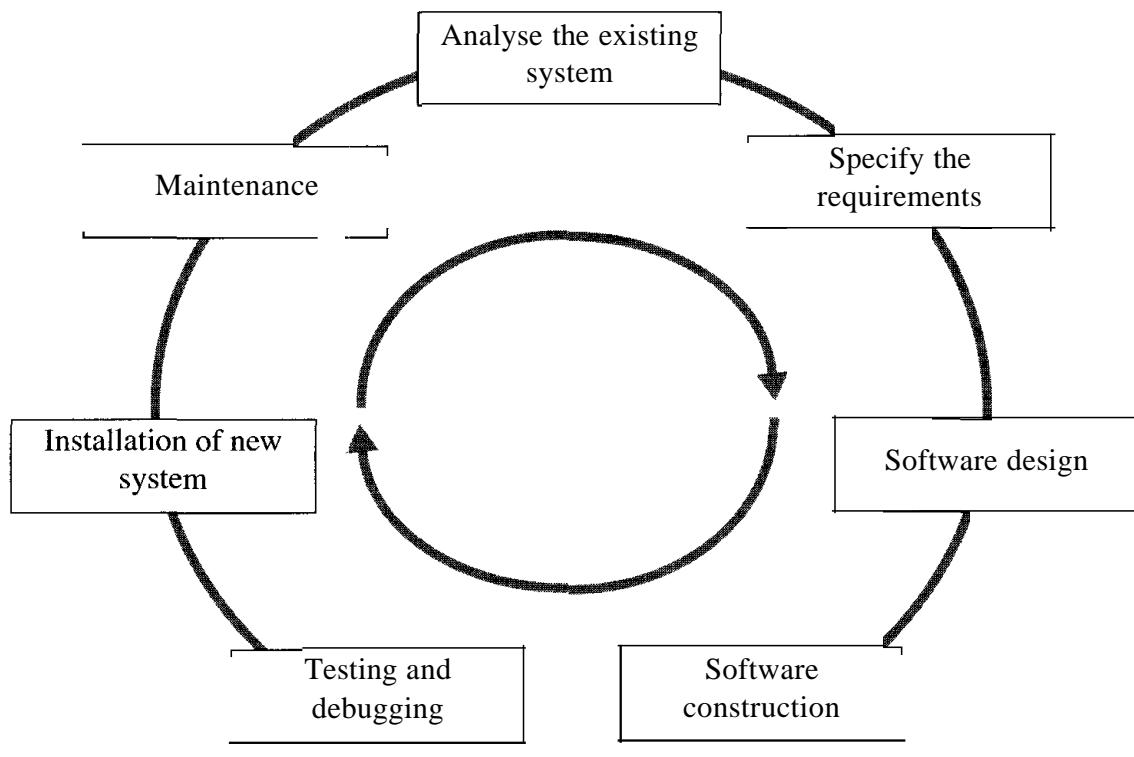
Some of the related sections refer to the systems life cycle rather than the software life cycle, but the same general principles apply.

© IBO 2004 1.5.2 CYCLICAL NATURE

This process is cyclical rather than linear because software systems once developed stay in place for many years. They have to be adapted to reflect changes in the way they are used. As an example, banks have been developing many new systems over the past 20 years, ATM's, telephone and internet banking to mention just three major innovations. Therefore maintenance involves making changes to an existing system which requires analysis leading to specification of requirements and so on.

For this reason the software life cycle is often shown in diagrammatic form.

Figure: 1.12



© IBO 2004 **1.6 SOFTWARE DESIGN**

Software design involves determining the inputs, outputs, data file formats including any data capture methods and output formats. There are very many ways to input data into the computer:

© IBO 2004 **1.6.1 DATA REQUIREMENTS**

Some common input methods:

- Manual entry (keyboard, mouse, numeric keypad, touch screen).
- Direct Data entry (OMR, OCR, MICR, barcodes).
- Automatic data entry (sensors, buttons, switches).

Common output methods include:

- Hardcopy (permanent) output (printed reports, graphic plots)
- Softcopy (temporary) output (monitor screens, graphic displays)
- Physical output (actuators, relays)

Chapter 3 outlines the common devices used to capture input data and display output data.

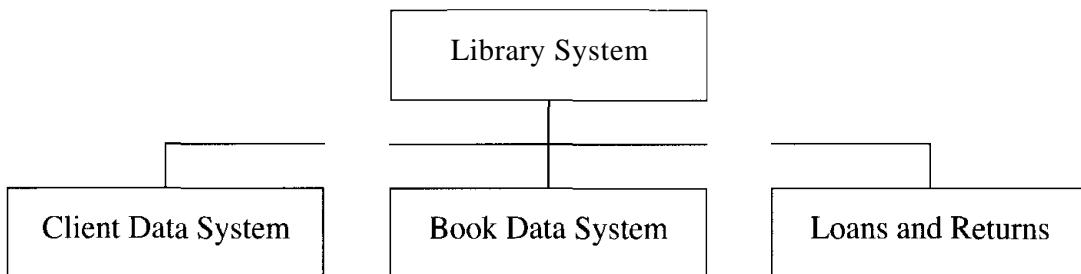
The design of data files is very important; it is not easy to change these once a system has been put into operation (whereas it may be possible to change input from keyboard to touch screen, for example, with few problems).

One of the functions of an operating system is to provide a user interface and, of course, software developers will need to provide one for their own users. The two main 'flavours' are Graphical User Interfaces (GUI's) and Command Line Interfaces (CLI's), see section 1.3.6.

© IBO 2004 **1.6.2 MODULAR DESIGN**

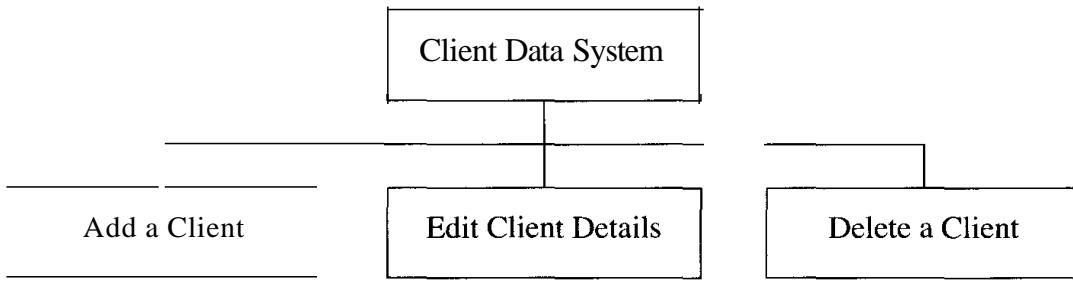
The software parts of a solution need to be designed in detail and a common starting point is the module diagram. This breaks the problem down into smaller components (modules) in a process known as '**top-down design**'. Using our library example again:

Figure: 1.13



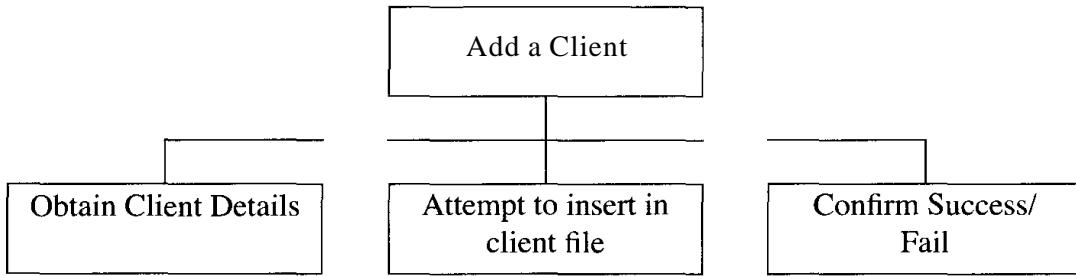
The CLIENT DATA SYSTEM might be required to update the client file, so this module can be further broken down - this is 'stepwise refinement'.

Figure: 1.14



Again we can break down each of these processes by further refinement:

Figure: 1.15



We will usually find, again, at some level our modules are dealing with input, processing and output of discrete data items. When this level is reached and the corresponding data structures (e.g. the data file) have been defined, then pseudocode can be written to give the final level of detail necessary to program the modules.

In the Java language, our modules are called 'Classes' and a 'Class' includes both the programming code and the data it operates on, whereas earlier structured (or procedural) languages like Pascal separated the data types and structures from the code. A procedural language might offer this type of structure:

Program name

global data:

```

data types (integers, reals, characters, Booleans)
data structures (strings, arrays, records, files)
eg client file
  
```

```

procedure main
begin
    procedure add client
    procedure edit client
    procedure delete client
end
  
```

```

procedure add_client
begin
    list of programming statements
  
```

```
these operate on the global data types and
structures, eg store client data in a file
end
```

(a more sophisticated procedure might use parameters and local data)

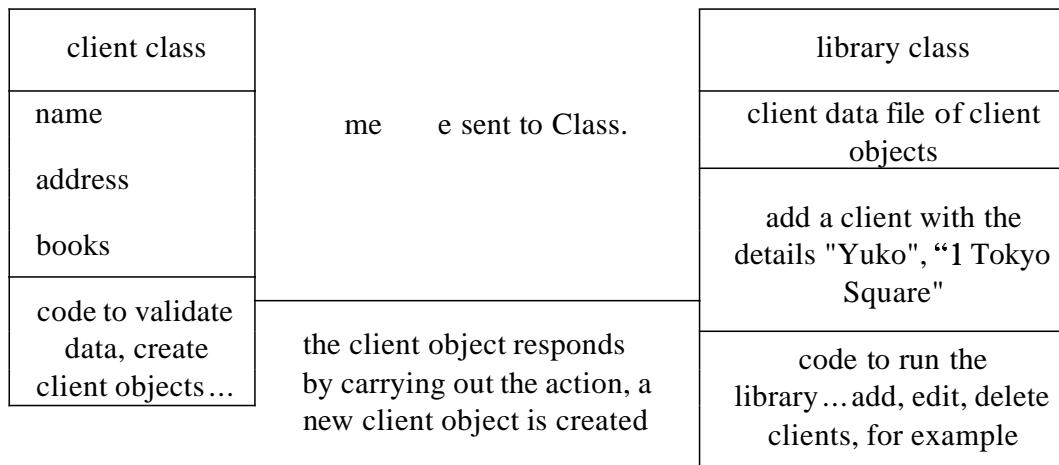
```
procedure edit_client (parameter: client_record)
local data:
    data types (integers, reals, characters, Booleans, eg
    the name of the client, current number of borrowed
    books) data structures (eg the client data file)

begin
    list of programming statements
    these operate on the parameters, the local
    variables and the global data types and
    structures
end
```

The local variables and parameters help to reduce the inter-dependence of modules which can be a problem with large scale projects (consisting of many hundreds or thousands of modules).

Modern languages combine procedures with the data on which they operate (Classes or objects) and don't allow other objects to access their own internal structure. In the library example, a client object could store data about library users. The data to be stored would be held internally, other classes could operate by sending messages or requests to the object.

Figure: 1.16



This topic is far from trivial (!) and is more relevant to higher level students, therefore we address it further in Chapter 5. Students would not be asked to make a comparison of these two methods of programming and this very brief discussion is really aimed at those older teachers (perhaps of the authors' generation) who might be more used to the modular, structured programming approach.

Software Development

For those who wish to learn more, there are excellent, easily-readable books such as:

Understanding Object-Oriented Programming with Java, Budd T, Addison-Wesley, 2000, ISBN 0-201-61273-9

Objects First with Java A Practical Introduction Using BlueJ, Barnes D & M Kolling, 2003, ISBN 0-13-044929-6.

Teachers who wish to use Java for Object-based or even procedural programming can still do so with Java and JETS, particularly at Standard Level.

Developing a solution using modules (or objects), as above, brings many benefits:

- Errors are easier to locate within a faulty module as opposed to looking through a large amount of homogenous code, thus testing and debugging is easier.
 - Work can be split up among many people or teams with each one being assigned a module, thus the overall system is developed in a shorter time.
It is easier for an individual to program a small module and, should the need arise, the work done on a module can be abandoned (the design did not work out or the programmer left the team).
- The system is easier to understand and therefore maintain when it is split into small units.

© IBO 2004 **1.6.3 PROTOTYPING**

The new program in Computer Science requires students to implement a prototype as part of their dossier design.

A prototype is a simple version of a system produced during the design stage.

© IBO 2004 **1.6.4 PROTOTYPING APPROACH TO SYSTEMS DESIGN AND DEVELOPMENT**

The purpose of prototyping is to show the user an interface and to give some indication of how the system is expected to work. The prototype is not a full working version of the software but it does allow the user to propose changes at the design stage.

The prototype could be produced by a different system than the one eventually used for the project. A developer could present an interface using an applications package (presentation software for example) even though the eventual solution might be implemented using Java.

© IBO 2004 **1.6.5 ADVANTAGES OF PROTOTYPING**

One of the key problems for any systems analyst is that the users have no clear idea of what they want from a system (or, at least, are not able to specify their needs precisely and without ambiguity).

By using prototyping at an early stage in the project, the analyst/designer can produce different prototypes showing alternative solutions. The user can then give concrete feedback to the designer to indicate whether they are implementing the solution the user desires.

The earlier that changes can be made to a system the less time is wasted completing the system and therefore less money will be spent. Early changes are not as costly as late changes!

CLASS ACTIVITY

Try this in pairs. Get one partner to describe (by speech alone) a simple object that they want produced (e.g. a birthday card, an application form). On the computer, design and implement the object according to the description they gave. Get the partner to describe how closely the object met their initial expectations. Now reverse roles or switch partners but allow the 'designer' to sketch an initial solution after the verbal description has been given. The sketch corresponds to a prototype and the advantages of this approach should be apparent.

Getting to know what users want is often the hardest part of developing any system (often the users are not sure what is possible). You will probably find this out the hard way when you complete your dossier project.



©IBO 1.6.6 EFFICIENCY OF SOLUTIONS

Computer Scientists usually compare the storage and speed requirements of algorithms using big O notation but only a limited knowledge is required at Higher Level (Chapter 5) and none at Standard Level.

In a given high-level language we can estimate the storage required by a file of records as long as we know the number of bytes used to store each fundamental data type. Consider a typical record structure:

```
newtype STUDENTRECORD record
    FORENAME      string
    SURNAME       string
    BIRTHDATE     string
    SEX           character
    FORM          character
endrecord
```

If we allocate 25 characters for the FORENAME and SURNAME fields then we can calculate the size of one record:

| | |
|-----------|----|
| FORENAME | 25 |
| SURNAME | 25 |
| BIRTHDATE | 8 |
| SEX | 1 |
| FORM | 3 |
| Total: | 62 |

bytes

If there are not more than 800 students in a school then we can calculate that a file of:

$$800 \times 62 = 49600 \text{ bytes or just under } 50 \text{ KB}$$

Therefore we could (easily) store the file on a floppy disc, if need be, for this application. If it is necessary to read the entire file into memory for some operation (say, sorting) then we also know

Software Development

how much RAM will be required to do this. By today's standards, of course, such amounts of memory are tiny.

See Chapter 2 for details of the sizes occupied by Java's data types.

If we need to search this file for a particular record we can calculate that, on average, we will have to search half the length of the file (or 400 records). Therefore we can make some estimate of the time it will take. Searching and sorting are discussed further in Chapter 2.

1.7 DOCUMENTATION

There are two types of documentation that are needed for computer systems; system (or technical) documentation and user documentation.

© IBO 2004 1.7.1 DOCUMENTATION OF THE CYCLE

Documentation accompanies every stage of the cycle so that when changes need to be made the reasons for design decisions, input data collection forms, choice of test data, data structures used and so on can be easily located. Programmers and designers, generally speaking, hate to do this work because it is usually tedious whereas they enjoy the creative aspects more.

It has been known that making a minor change to a running system can involve nearly as much work as developing the system in the first place because of inadequate documentation. In general, if documentation is poorly done, maintenance costs become very high.

CLASS ACTIVITY

This cycle of plan - design - implement - evaluate occurs in many other disciplines. Compare the life cycle given above with those that might be found in other IB areas (such as Design and Technology, Theatre Arts, the Extended Essay) and other disciplines such as instructional design, CAD/CAM and other engineering tasks. Sources might be books, websites and technical journals.



© IBO 2004 1.7.2 SYSTEM DOCUMENTATION

This documentation is intended for programmers who have to understand the program so that it can be maintained - it may have errors or there may be requests for improvements and enhancements. Thus the target audience for this type of documentation is people who need to understand the inner workings of the program (often in a hurry).

System documentation itself is often classified as either internal documentation - these are exemplified by the comments which accompany the source code listings - or external documentation.

As well as commenting on the purpose of each subprogram, the type and purpose of any subprogram parameters and of any local variables, programmers can help make code more understandable by using:

- meaningful identifier names.
- keeping methods small and classes limited in scope.
- a constant indentation scheme to illustrate code structure.

These comprise the elements of '**programming style**' referred to in dossier assessment criterion C1.

External documentation provided for other programmers should include the following items:

- Purpose of the program.
- Data flow in the system.
- Hardware and software requirements of the solution.
- Structure of the program (e.g. via a module diagram).
- Description of the purpose of each module in the diagram.
- The module interface - what parameters it has and any values it returns.
- A list of identifiers (variables) used by each module.
- A description of the module and any subprogram algorithms in pseudocode.
- A description of any data structures used (arrays, records, files etc.).
- The test plan – functional and data entry testing.
- Description of any testing and debugging actually carried out.
- Evaluation of the system.

Your dossier project requires you to submit many of the above items for assessment.

© IBO 2004 **1.7.3 USER DOCUMENTATION**

This is documentation written for the non-technical user of the program. Therefore, terms like 'record', 'array' and 'linked list' are never appropriate in user documentation. This person never sees the internal structure of your program (and couldn't care less, as long as it works). When we are working on a single project as both programmer and user it becomes difficult to separate these aspects - that is why it is worth selecting a real world problem for your dossier which can then be tested on a real non-technical user.

Your user needs to install your program on their system, to know how to use it and to troubleshoot any difficulties. User documentation often includes these as separate sections.

Installation requires the user to ensure that they have the appropriate hardware and software (e.g. operating system) for your program. They may have to change system settings to enable your program to run properly.

Sample input data and expected outputs should be provided to help the user ensure that the program is running as expected. Obviously, screen shots will also help them understand what they might expect to see when running the program.

A well-organised trouble-shooting section should be provided so that users can quickly find out what the problems might be and hopefully fix them before calling you.

Many modern systems of any complexity also include online or onscreen help for the user. These can be simple hypertext files or even interactive tutorials. You are not expected to develop these in your dossier programs.

Software Development

2

• Chapter contents

| | | |
|---------|--|------|
| 2.1 | Program construction in Java | 40 |
| 2.1.1-3 | Data types, data structures and constructs in Java | 41 |
| 2.1.4 | Trace algorithms in Java | 102 |
| 2.1.5 | Evaluate algorithms in Java | 104 |
| 2.1.6 | Construct algorithms in Java | II 2 |
| 2.1.7 | Explain the need for searching and sorting | 114 |
| 2.1.8 | Apply specified searching and sorting algorithms | 4 |
| 2.1.9 | Compare the efficiency of searching and sorting algorithms | 122 |
| 2.1.10 | Discuss the efficiency of searching and sorting algorithms | 122 |
| 2.1.11 | Programming Errors | 129 |

©IBO 2004 **2.1 PROGRAM CONSTRUCTION IN JAVA**

The authors would like to thank Dave Mulkey of Frankfurt International School for his hard work in defining JETS. Chapter 2 leans heavily on the work he contributed to the IB Subject Guide.

It is not necessary for students to know all of Java (a really huge task). The IB Subject Guide defines the appropriate content for us which is known as the IB Java Examination Tool Subset or JETS. It is found in the Appendix to the Subject Guide.

You do not need to write algorithms in perfect Java (or JETS) syntax in the examination, missing a semi-colon off the end of a statement is likely to be forgiven:

```
if (x == 3)
{
    y = y + 4           // there is an error here
    z = Y - x;
}

x = x + 1;
```

However, a slip that leads to ambiguity or incorrect logic will be penalized:

```
if (x = 3)
    y += 4
    z = Y - x;

x = x + 1;
```

In this example it is hard to know if the candidate made a syntactic slip in leaving out the enclosing bracket or a 'genuine' mistake. The error in comparison (= instead of the correct == would probably be OK). Notice that we have used:

```
x = x + 1;
```

as opposed to the more customary:

```
x += 1;
```

or even

```
x++;
```

in this example. This is deliberate since there is less chance of making mistakes or being ambiguous. The examination paper will also use the same conventions to make algorithms clear to students. We will follow the conventions laid down by JETS in this book with two main exceptions:

- i. Code examples on the book's support website may not always follow the JETS convention.
- ii. Where we relate examples to Classes students could use in dossier programs we may use constructs and syntax that are Java language rather than the JETS subset.

©IB02.1.1-3 DATA TYPES, DATA STRUCTURES AND CONSTRUCTS IN JAVA

2004

VARIABLES AND TYPES

Variables are named memory locations where data can be stored, each variable can hold one of the data types defined below. A variable only holds one piece of data at a time.

The basic variables, or primitives as they are known in Java, have the following data types:

- byte
- int
- long
- double
- char
- boolean

Candidates may encounter the other primitives short and float, but they will not be used in examination questions.

A primitive is declared with an identifier name as follows:

```
int number;
char initial;
```

Identifiers always start with a lowercase letter by convention, usually they are all in lowercase unless they are multi-word identifiers in which case they may look like this:

```
double incomeTaxRate;
```

You cannot mix data types, as in:

```
char initial = 23.8; // this is an ERROR!
```

This generates a syntax error in the java compiler. Notice that *//* defines a comment - a piece of text not considered part of an instruction - put there for information.

It is always useful to write comments in your examination answers as you may get credit for this (especially where the exact syntax of your statements is unclear).

You can sometimes allocate a data type to a different variable, e.g.:

```
char initial = 65; // this is not necessarily an error
```

initial has the value of the character 'A' whose UNICODE value is 65. We don't know in this case if it is an error made by the programmer, it is not a syntax error.

It is possible to transform some data types using a cast:

```
int n = (int) Math.round(2.566); // round returns type long
```

long is a type of integer that can deal with a greater range of values than int.

DESCRIPTION AND APPLICATION OF JETS PRIMITIVES

| primitive | size (bytes) | Example data | Description |
|-----------|--------------|----------------|--|
| byte | 1 | 123,-59 | Used for small integers (whole numbers) in the range -128 to +127 |
| int | 4 | -500,450 | Used for whole numbers (negative or positive) in the range -2,147,483,648 to +2,147,483,647 |
| long | 8 | -5000,4500 | Used for whole numbers (negative or positive) in the range -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807. |
| double | 8 | -34.49587,3.0 | Used for numbers with fractional parts in the approximate range -1.79 * 10308 to +1.79 * 10308 |
| char | 2 | 'A', '*' , '9' | Represents a single symbol, a letter, other symbols or a number. Can also represent integer values in the range 0-65535. |
| boolean | 1 | true/false | Represents one of two states only |

The way in which these (and other) types of data are stored in the computer are discussed in Chapter 3.

Uses of these primitives

int

The int primitive type can be used to represent whole numbers between:

-2,147,483,648 and + 2,147,483,647.

-2,147,483,648

etc. -3, -2, -1, 0, 1, 23 etc.

2,147,483,647



The largest int which can be stored depends on the number of bits allocated by a particular system and in Java, 32 bits are allocated. Higher Level students will need to understand how the number of bits is related to the maximum and minimum values but Standard Level students need not.

Integers are thus useful for storing data such as the distance in air miles between Paris and New York, the number of items in stock in a supermarket or the number of students in a class.

byte

This is an 8-bit storage unit, otherwise equivalent to the int which can store values between -128 and +127.

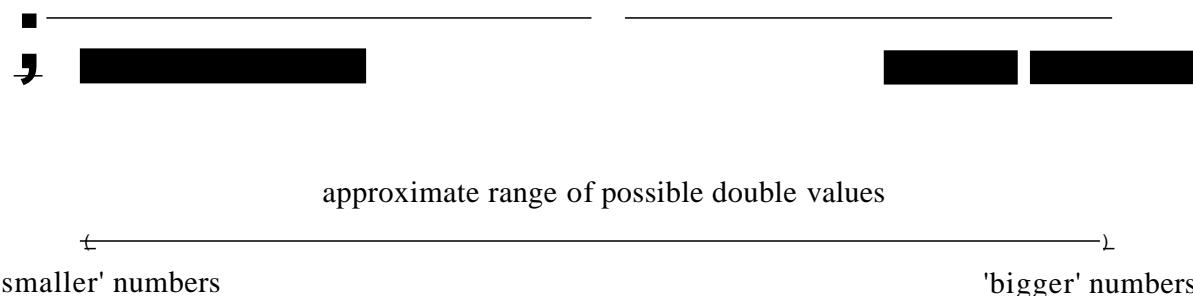
long

This is a 64-bit storage unit, otherwise equivalent to the int but which can store values in a much larger range. It is needed by JETS mainly because certain Math and RandomAccessFile methods return a long value; you will rarely need to use a long (or byte) type in your own programs.

double

These are numbers with fractional parts and they will always require more storage space in computer memory than integers. There are four limits to the range of numbers that can be stored:

| largest magnitude negative number | smallest magnitude negative number | smallest magnitude positive number | largest magnitude positive number |
|---|--|--|---|
| -1.79x10 ⁻³⁰⁸ | -4.94x10 ⁻³²⁴ | +4.94 x10 ⁻³²⁴ | +1.79x10 ³⁰⁸ |



With doubles the computer will be able to store larger and smaller numbers compared to using integers but there will be some loss of precision when calculations are made. Also, there are some values which cannot be represented in this format (consider the gap around zero for example).

Notice we have tried to avoid using the term 'small' without qualification since the largest magnitude negative number is usually considered smaller than the smallest magnitude negative number.

You can specify a double value by using a suffix if you wish:

```
double x = 5.0d;
```

double primitives are useful for storing data which has fractional parts such as currencies, the average height of students in a class or FM radio frequencies.

char primitives

Computer Fundamentals

The first computers were developed to perform complex calculations (such as the exact trajectory of an artillery shell). However, you might not be studying this subject if it had not been recognised that character data could be represented within the computer allowing text to be input, stored, processed and output. Characters are represented by bit patterns as discussed further in Chapter 3.

Character data on its own is fairly limited - more useful is the ability of most programming languages to deal with collections of characters (see Strings). Here we should mention that character data can consist of letters, punctuation marks and other symbols and numbers. Numbers are stored as char primitives when you do not need to perform arithmetical calculations on them (house numbers, for example).

In a Java program you can also use the char type to store numbers between 0 and 65535 as it stores a positive 16-bit integer value (the UNICODE representation of the corresponding symbol).

boolean primitives

In the middle of the 19th century a (largely self-educated) British mathematician called George Boole developed the system of logic which still bears his name. This work lay largely unused until just before the Second World War when Claude Shannon, then a student at MIT, proposed that the system could be used to describe the behaviours and properties of electronic switches. Electronic switches are used by the millions in integrated circuits in modern computers.

The boolean primitive stores one of two states, true or false, corresponding to the on or off state of an electronic switch. In a computer application, the Boolean type could be used to store data such as whether a video tape is rented or not, whether a student is a boarder or not, if a certain item has run out of stock or reached its expiry date etc.

Assignment means giving values to variables; this may be done when the variable is first defined (good practice) or at a later time:

```
char initial = 'r'  
int number;  
number = 23;
```

ASSIGNMENT

Assignment always takes the expression on the right of the 'equals' sign and places the result in the variable on the left:

```
number = number + 23; // number is now 23 greater than it was
```

Notice that the primitive number can hold one and only one value, let's assume that when located in memory we have a picture something like this:

| | location | contents |
|------------|----------|----------|
| int number | 10001 | 4 |

When the statement:

```
number = number + 23; // number is now 23 greater than it was
```

is executed, the right-hand side is evaluated first. Thus 23 is added to the value in memory (4) and the result, 27, is then placed into memory.

| | location | contents |
|-------------|----------|----------|
| int number" | 10001 | 27 |

The value 4 is overwritten and lost forever. This has implications when you want to swap the values in two primitives.

| | location | contents |
|-------------|----------|----------|
| int number2 | 10001 | 13 |
| | 10010 | |
| int number! | 10011 | 27 |

if we try something like:

```
number1    number2;
number2    number1;
```

Both numbers are going to be 13 (I hope you can see this). What we need to do here is use a temporary location to hold one of the values:

| | location | contents |
|-------------|----------|----------|
| int number2 | 10001 | 13 |
| int temp | 10010 | |
| int number! | 10011 | 27 |

Then, this should work:

```
temp = number2;
number2    number1;
number1 = temp;
```

Try tracing through the above statements on the diagram to verify that it does actually work.

EXERCISE 2.1

What is the result of the following assignment statements?

```
temp = number2;
number1    number2;
number2 = temp;
```



BEYOND PRIMITIVES

As well as primitive data types, Java also includes objects and classes. Some of these objects are built in as part of the language while others can be imported via libraries or defined by the user. One of the most important built in classes is the String class:

```
String name = "Binh Nguyen Le";
```

You can tell String is a class and not a primitive because the type begins with a capital letter. While we can think of our primitives as 'containing' a value, it is important to remember that where objects are concerned the identifier is a reference to where the object resides in memory. This is because objects have other data stored with them besides their apparent value. Thus in:

```
String name = "Binh Nguyen Le";
int len = name.length(); // Length of String object
```

`length()` is a method (a piece of Java Code) that returns the current length of the string name.

Primitives have no such extra information thus the following does not work:

```
// not possible!!, int is a primitive:
int ln = len.length();
```

Therefore, without going further into Java at this point, if we imagine memory as a series of storage locations:

| | location | contents |
|---------------|----------|--|
| int len → | 0010001 | 23 |
| | 0010010 | 'B:_nh Nguyen Le' |
| | 0010011 | number of chars in string |
| String name → | 0010100 | reference to code to get length |
| | 0010101 | reference to code to convert characters to lowercase |
| | etc. | etc. |

Then we can see that primitives really consist just of a value whereas objects have a whole lot of associated 'baggage' and that name actually refers to the location of a whole lot of other stuff. Not very technical, I know, but we'll get into some of the detail later.

Notice the difference between String literals and char literals:

```
String literal: "Hello Globe!"
char literal 'H'
```

INPUT/OUTPUT

The purpose of most computer programs is to communicate with users; this is handled by the operating system. Since Java is a cross-platform language, independent of OS, there are fewer specific input/output statements than we might find in other languages.

For input and output the IE has defined a set of very simple statements for use in examination papers. These are:

| method used | description |
|---|---|
| <code>inputString("Type anything")</code> | outputs prompt, accepts input, and returns a String |
| <code>input ("Type anything")</code> | same as <code>inputString</code> |
| <code>input ()</code> | accepts input, and returns a String, no prompt is given |
| <code>inputByte("Type a byte")</code> | outputs prompt, accepts input, and returns a byte |
| <code>inputInt("Type an integer")</code> | outputs prompt, accepts input, and returns an int |
| <code>inputLong("Type a long integer")</code> | outputs prompt, accepts input, and returns a long |
| <code>inputDouble ("Type a decimal")</code> | outputs prompt, accepts input, and returns a double |
| <code>inputBoolean("true or false?")</code> | outputs prompt, accepts input, and returns a boolean |
| <code>inputChar("Yes or no (y/n)? ")</code> | outputs prompt, accepts input, and returns a char |
| <code>output(String)</code> | outputs a String |
| <code>output(int)</code> | outputs an int value |
| <code>output(long)</code> | outputs a long value |
| <code>output(double)</code> | outputs a double value |
| <code>output(boolean)</code> | outputs a boolean value |
| <code>output(char)</code> | outputs a char value |
| <code>output(byte)</code> | outputs a byte value |

The subject guide indicates one way in which these methods can be implemented in practical Java programs.

We shall use them here to generalize and simplify code examples, as in:

```
String name = input("What is your name?");
double height      inputDouble("How tall are you in em?");
```

Computer Fundamentals

```
double heightInches = height / 2.54;  
output("Hello " + name);  
output("Your height in inches is: " + heightInches);
```

Note that it is a feature of Java (rather than JETS) that a number is automatically converted to a string when joined together with the `+` operator (said to be 'concatenated'), that is:

```
"Your height in inches is: " + heightInches
```

produces a String result with the primitive (`heightInches`) converted to a String equivalent.

EXERCISE 2.2

Predict the outcome (type and contents) of the following statements:

```
result = "My name is" + "Michael Moore" + 3;  
height = 2.3 * 2;  
length = length + 20 + 3;
```



Sometimes it is necessary to put non-printing or other special characters into Strings, for example the code for newline or a quote character(`"`). This is done by using the backslash character as an 'escape' code:

```
result = "My name is\n" + "Michael Moore\n" + 3;
```

Will produce a different result when printed. Other common uses of the backslash in a String are:

| string | produces |
|--------|-----------------|
| \n | a new line |
| \t | a tab character |
| \\" | a backslash |
| \" | a quote |

You might need some of these to produce complex Strings such as:

Dhanasun says "Hello" to you\me\everyone

How would you encode this into a String?

ARITHMETIC OPERATORS

JETS only uses the following:

| operator | meaning |
|----------|----------------|
| + | addition |
| - | subtraction |
| * | multiplication |
| / | division |
| % | modulus |

Division of two int values produces an int result:

```
int x = 214;
int y = 7;
int z = x/y;
```

z will have the int value 30. The modulus operator returns the remainder after integer division thus after:

```
int x = 214;
int y = 7;
int z = x%y;
```

z will have the value 4.

PRECEDENCE

For example, does the statement:

```
int x = 23 % 5 + 2
```

mean:

```
int x = (23 % 5) + 2
```

or

```
int x = 23 % (5 + 2) ... ?
```

This is not one of those things you can safely ignore and hope for the best since the answer to the first is 5 and the second 2.

Precedence tells us which operators are done first when there are no brackets. You are surely familiar with an expression like $23 + 5 * 2$ from maths. The multiplication is done before the addition, it has higher precedence.

The order in Java is %, * and / are done before + and -. If a combination of %, * and / appears in the same expression, then they are evaluated from left to right:

```
int x = 23 % 5 * 2
```

Computer Fundamentals

leaves x with the value 6 (and not 3). Of course, as in maths, we can force parts of expressions to be evaluated first by using parentheses (that's brackets to you and me).

```
int x = 23 % (5 * 2)
```

leaves x with the value 3.

EXERCISE 2.3

Evaluate the following statements, stating whether parentheses are necessary:

```
double x = 23.0 / (5.0 * 2.0);
int x = 9 + 6 - 2 * 8;
int x=9%3*(5/7);
double x = -23.0 - (-5.0 + 2.0)
```



MATHEMATICAL FUNCTIONS

Other mathematical functions are specified in the Maths class, they are:

| method | use |
|---------------------|---|
| double, int abs (x) | Returns the absolute value of its argument (if $x < 0$ then returns $x \approx -1$), x may be double or int. |
| double pow (x, y) | Returns x raised to the power y . pow(16.0, .5) returns 4.0 while pow(2.0, 10.0) returns 1024.0, for example. Operates on double values. |
| double sin(x) | Returns the sine of x , where x is in radians. |
| double cos (x) | Returns the cosine of x , where x is in radians. |
| long round(x) | Rounds x to the nearest integer. Returns a long type which you can convert to int, if required, as in: <pre>int i = (int) Math.round(3.45)</pre> (i now has the value 3). |
| double floor(x) | Rounds x to the nearest integer not greater than x . floor(9.5) returns 9.0, floor(-9.5) returns -10.0. Returns a double value which can be converted to int if required, as in: <pre>int f = (int) Math.floor(x);</pre> |

CASTING

Means converting from one data type to another. Placing the primitive type in front of the right hand side of an assignment (as in the last two examples) is known as a cast.

Students are expected to know how to use the cast operator. Generally speaking you can always convert a type that uses less bytes or has a smaller range of values into one that has more bytes or stores a greater range of values:

```
int k = 450;
double x = k; // k gets converted to a double
```

This is an 'implicit' cast, it is done automatically. Implicit type conversion also occurs in mixed mode arithmetic:

```
double x = 20.5 * 5;
```

However, beware:

```
int x = 23;
int y = 5;
double z = x/y;
```

`z` has the value 4.0 because, in assignments, the right hand side is evaluated first and in this case, performs integer division. The following will give the expected result since it then becomes a mixed mode expression (`x` converted to double first):

```
int x = 23;
int y = 5;
double z = (double) x/y;
```

In many cases you use a cast to convert a 'bigger' type to a 'smaller' one, accepting the risk that precision could be lost:

```
int y = 120;
byte x = (byte) y;
```

No loss of data, but:

```
int y = 210;
byte x = (byte) y;
```

A byte only holds values up to +127 so you get an unexpected result. Java does not warn you because you used a cast - the compiler assumes you know what you are doing!

PROGRAM STRUCTURE

Let's look at the structure of a complete program. If you want to try this out you will need some Java software on your computer. For this book (and the IB Computer Science, in general) we have used the Java Development Kit (JDK) from Sun Microsystems and the BlueJ development environment (both free):

| Software | Version | Available for free download from: |
|----------------------|---------|---|
| Java Development Kit | 1.3 | http://java.sun.com |
| BlueJ | 1.2 | http://www.bluej.org |

Computer Fundamentals

As far as we are aware there is no reason why the examples in this book should not run on later versions of the software or, indeed, any other Java language system (please check the IBID Press website for the latest information - <http://www.ibid.com.au>).

If you open a new project in BlueJ and then create a new Class, you can double click the class module shown on the workbench and type in the following (or download the code examples from the IBID Press website URL given above):

```
/*
 * A class which can add 2 numbers together
 *
 * @author Richard
 * @version 060903
 */
public class Add
{
    public static void main(String[] args)
    {
        new Add();
    }
    /**
     * Constructor for objects of class Add
     */
    public Add()
    {
        double number1    = inputDouble("Input the first number: ");
        double number2    = inputDouble("Input the next number: ");

        double total = number1 + number2;
        output("The total is: " + total);

        /**
         * IPIO methods, (c) International Baccalaureate 2004
         * Computer Science Subject Guide, Appendix 2.
         */
    }

    static void output(String info)
    { System.out.println(info);
    }

    static void output(char info)
    { System.out.println(info);
    }

    static void output(byte info)
    { System.out.println(info);
    }

    static void output(int info)
    { System.out.println(info);
    }

    static void output(long info)
    { System.out.println(info);
    }
}
```

```

static void output(double info)
{ System.out.println(info);
}
static void output(boolean info)
{ System.out.println(info);
}
static String input(String prompt)
{ String inputLine = "";
  System.out.print(prompt);
  try
  { inputLine = (new java.io.BufferedReader(
      new java.io.InputStreamReclder (System.in))) readLine ();
  catch (Exception e)
  { String err = e.toString();
    System.out.println(err);
    inputLine = ""
  }
  return inputLine;
}
static String inputString(String prompt)
{ return input(prompt);
}
static String inpu()
{ return input("");
}
static char inputchar(String prompt)
char result =(char)0;
try{ result=input (prompt) .charAt(0);}
catch (Exception e){ result = (char)0;}
return result;
static byte inputByte (String prompt)
{ byte result=0;
  try{ result=Byte. valueof (input. (prompt) trim () ) .bytevalue () ;}
  catch (Exception e){ result = 0;}
return result;
}
static int inputInt(String prompt)
{ int result=0;
  try{ result=Integer .valueOf input(prompt)trim() .intValue() ;}
  catch (Exception e){ result = 0;}
return result;
}
static long inputLong(String prompt)
long result=0;
try{ result=Long.valueOf (input. (prompt) trim()) .longvalue();}
catch (Exception e){ result = 0;}
return result;
}
static double inputDouble(String prompt)
double result=0;
try{ result=Double.valueOf

```

```
        input(prompt).trim()).doubleValue();  
    catch (Exception e){ result = a;}  
    return result;  
}  
static boolean inputBoolean(String prompt)  
{ boolean result=false;  
    try{ result=Boolean.valueOf  
        (input (prompt).trim()).booleanValue()  
    catch (Exception e){ result = false;}  
    return result;
```

Students do not need to know these I/O methods or even understand how they work.

We will not explain all the examples in the book in detail since this is not a Java Book as such, however, we will provide resource materials for teaching Java in the future, please watch the IBID Press website for further details. Our purpose here is to explain and illustrate the assessment statements in the syllabus, much as we did in our First Edition when PURE was the IB 'language' of choice.

However, it is an excellent idea to experiment with the examples given in this chapter and chapter 5 for Higher Level.

Let's just look at a couple of features of the program in a little more detail:

```
/**  
 * A class which can add 2 numbers together  
 *  
 * @author Richard  
 * @version 060903  
 */
```

These comments are used to automatically generate documentation of the Class using a system known as JavaDoc. The @ tags and other information are used to convert the comments into html in a pre-defined format.

Further discussion of this topic is outside the scope of the book (see <http://java.sun.com> for further details).

```
public class Add  
{  
    public static void main(String[ ] args)  
    {  
        new Add () ;
```

The main function always has this form, it defines a Class which can be run as a program. This main method calls the constructor:

```
/**  
 * Constructor for objects of class Add  
 */  
public Add()  
{
```

```

double number1 = inputDouble("Input the first number: ");
double number2 = inputDouble("Input the next number: ");
double total = number1 + number2;
output("The total is: " + total);

```

The constructor is a special method that has the same name as the Class and does not contain any other qualifiers except public. Should you forget to add a constructor, Java will automatically supply an empty, no argument one for you in the 'background'. The constructor is called whenever a new instance of the Class is needed:

```
new Add();
```

It is subtle and very hard to trace bug to create a 'normal' method in place of a constructor:

```

public class Add
{
    public static void main(String[] args)
    {
        new Add();
    }
    public void Add()
    {
        double number1 = inputDouble("Input the first number: ");
        double number2 = inputDouble("Input the next number: ");

        double total = number1 + number2;
        output("The total is: " + total);
    }
}

```

This program will compile just like the other (as long as the IBID methods are added too), but nothing will appear to happen (and all because of one, carelessly added keyword!).

Avoid this problem by never using the class name in a method you write (unless it is meant to be a constructor, of course).

SEQUENCE

Where statements or actions in an algorithm simply flow from one to the next, this is known as a sequence of instructions. Often the sequence can be important:

```

public Add()
{
    double number1 = inputDouble("Input the first number: ");
    double number2 = inputDouble("Input the next number: ");
    double total = 0;
    output("The total is: " + total);
    double total = number1 + number2;
}

```

This algorithm does not produce the required result.

Computer Fundamentals

Any program can be represented in a modular design, even one as simple as this.

```
1**
 * A class which can add 2 numbers together
 *
 * @author Richard
 * @version 060903
 *1
public class AddModular
{
// Change I-need to declare variables here-in Class definition
// Class variables, available to every method of the Class
double number1, number2, total;

public static void main(String[] args)
{
    new AddModular();

// Change 2 call the methods from the constructor

1**
 * Constructor for objects of class AddModular
 *1
public AddModular()
{
    obtain();
    calculate();
    display();
}
1**
 * Method to obtain two numbers
 *1
public void obtain()
{
    number1 = inputDouble("Please input your first number: ");
    number2 = inputDouble("Please input your second number: ");
}
1**
 * Method to add two numbers
 *1
public void calculate()
{
    total = number1 + number2;
}
1**
 * Method to display the total
 *1
public void display()
{
    output("The total is: " + total);
```

1**

* IBIO methods, (c) International Baccalaureate 2004

* Computer Science Subject Guide, Appendix 2.

*1

(as before, cut and paste these methods in)

}

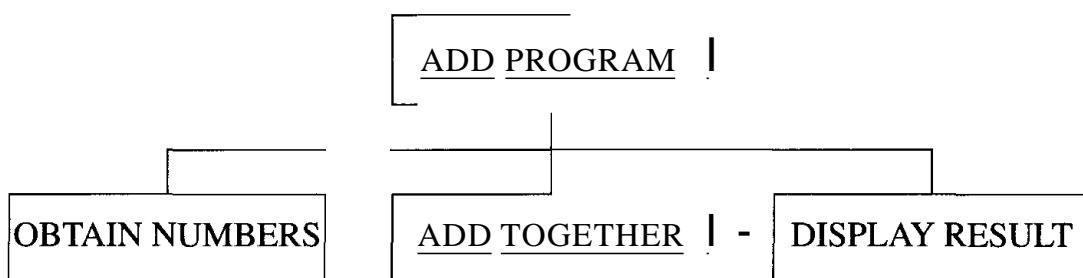
Modularity is a powerful technique for writing programs, although it is possibly overkill here. At least the Class adheres to the good principle that a method should do only one task.

CONSTRUCTING A MODULAR PROGRAM

The IE Computer Science program requires students to analyse and solve problems rather than to write code solutions to simple programming exercises. When students are beginning to learn a language, they are naturally given small scale exercises in keeping with their limited knowledge of the structures and constructs available. However, it is possible to encourage students to think about design issues right from the start.

As an example consider the small problem of adding two numbers together, a typical beginners exercise. Students can be taught to break this down into input-process and output sections:

Figure 1.10



Rudimentary concepts of program testing and evaluation can also be introduced at this stage. As the programming exercises increase in complexity, other stages (analysis and user documentation, for example) can also be brought in. This way, by the time students come to write their dossiers, they have already become accustomed to the software development life cycle.

While it is quite possible to apply these techniques, which are associated with procedural type programming, to Standard Level student projects, it is probably better to introduce HL students to the concepts associated with object oriented programming right from the start. The debate goes on at first year University level as to whether objects should be introduced to students early or late (or, indeed, in the middle). There are textbooks (very many textbooks) which favour one or another of these methods out there. You can visit <http://www.ib-computing.com> for details of books currently favoured by IE teachers.

SCOPE OF IDENTIFIERS

Notice that it was necessary to move the primitive declarations

```
// Class variables, available to every method of the Class
double number1, number2, total;
```

Computer Fundamentals

to the Class definition (opening) section. If they had remained declared in the constructor (public AddModular), they would not have been 'visible' (in scope) for the other methods - obtain, calculate, display.

In general, identifiers and other types are restricted to the program block (including methods) in which they are declared:

```
public void calculate()
{
    double total = number1 + number2;
}
```

The identifier total is only valid within calculate. If another total is used elsewhere, it counts as a different identifier:

```
public void display()
{
    double total;
    output("The total is: " + total);
```

Not the same total. This can lead to errors which are hard to track down so always think carefully about declarations like the one above: "Do I really want a new total or should I be using the Class data member instead?"

PARAMETERS

One way of passing a variable to another method is to use a parameter:

```
public void display(double total)
{
    output("The total is: " + total);
```

Now total has to be 'passed' to the function display, as in:

```
public void calculate()
{
    double total = number1 + number2;
    display(total);
```

or even:

```
public void calculate()
{
    display(number1 + number2); // call to display method
```

Using an expression such as number1 + number2 only applies to primitive identifiers, however. The identifier or expression that appears where the function is 'called' is often termed the 'argument' or sometimes the 'actual parameter'.

The parameter acts like a local identifier whose scope is the method body, therefore, as we shall see later, a parameter can mask or shadow a Class data member:

```

public void display(double total)
{
    // if there is a Class data member called total
    // it's not visible here!
    output("The total is: " + total);
}

```

Methods may also return a value. They do this by replacing the keyword void with the type of value that they return. For example, a function to add two numbers together could be written:

```

public double calculate(double number1, double number2)
{
    return (number1 + number2);
}

```

The call might then look like this:

```
double total = calculate(5.0, 6.8)
```

Which has the effect of placing the value 11.8 in the identifier total. Clearly, this gives us very many different ways of writing the AddModular program.

CLASS ACTIVITY

Re-write the AddModular program using the above techniques. Compare the ones that worked. How many different and valid ways are there to write it. Challenge: can anyone write the entire program as a single line called from the constructor? Is there any advantage in doing this?



EXERCISE 2.4

The variable declarations have been left out of this algorithm, what should they be?

```

public TempCon()
{
    ??? tempType = input("Enter a C or an F: ");
    ??? temp = inputDouble("Enter a temperature: ");
    ??? result;

    if (tempType.equals ("F")) || (tempType.equals ("f"))
        result = temp * 9.0 / 5.0 + 32.0;
    else
        result = (temp - 32.0) * 5.0 / 9.0;

    output("The converted temperature is: " + result);
}

```

Rewrite the algorithm to use three modules corresponding to input, process and output.

METHOD SIGNATURES

The line where the function parameters and return type are defined is known as the method signature.

```
public void calculate();
public double calculate()
public void calculate (double number1, double number2)
public double calculate(double number1, double number2)
```

are all different methods and could all be used within the same program. If you were astute (or even awake) you might have wondered about all those output methods in Class IBIO in our first full program example. The correct one is called, depending on the type of the argument. This process is known as method overloading.

SELECTION

This is the use of different sets of instructions depending upon a given condition (which can have one of just two outcomes – true or false). Selection is also referred to as 'branching'. Consider these examples:

Simple if .. endif

```
if (temp < 5)
    output("It is cold today");
```

If the value of the primitive temp is 4 or less, the message is output, otherwise nothing happens.

Using an else clause

```
if (age> 20)
    output("you are old");
else
    output("you are young");

if (mark < 40) then
    output("you failed");
else
    output("You passed");
```

If the condition evaluates to 'true' the first pass is done, otherwise the statements after **else** are done.

You can use the following comparison conditions:

| operator | meaning |
|----------|---|
| < | less than |
| > | greater than |
| == | equal to |
| != | not equal to |
| <= | less than or equal to |
| >= | greater than or equal to |
| | On its own acts as a not operator, ie turns true to false and false to true |

It is a very common programming error to use = when == was meant and it can be very hard, under some circumstances, to spot such an error.

Sometimes the conditions (the parts in brackets) can be combined with && (and) or || (or) operators.

```
if ((mark < 0) || (mark> 100))
    output("Not a valid mark");

if ((mark < 75) && (mark> 65))
    output ("Your grade was a B");
```

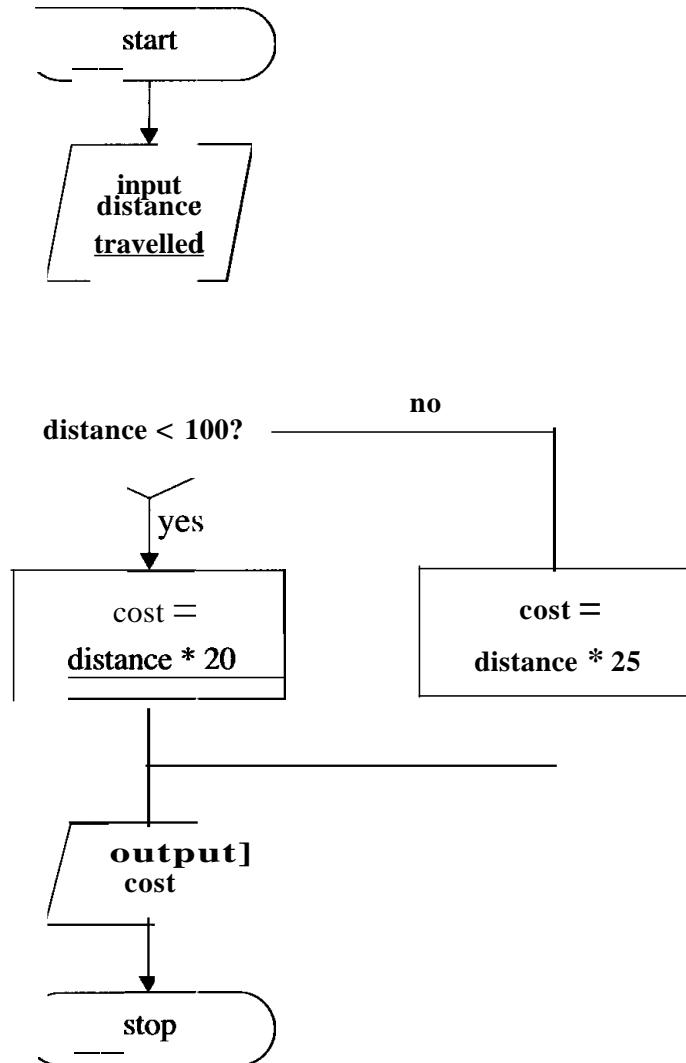
As we saw in the earlier section, arithmetic operators have precedence, and so do logical ones, && is evaluated before ||.

Anywhere a single statement can occur in Java, a compound statement, enclosed in braces (squiggly brackets to you and me) can also occur:

```
if ((mark < 0) || (mark> 100))
{
    output("Not a valid mark");
    output("Please try again");
}
else
    //stuff to process input value
```

EXERCISE 2.5

Re-write the following flowchart algorithm as a java program



NESTING

Nesting occurs when one control structure is placed inside another (if and if else are control structures).

Consider this algorithm which attempts to determine if you can afford to go out tonight:

```

final static int CASH_LIMIT = 50;

String choice = input("Do you feel like going out? (y/n) ");
int cash = inputInt("How much cash do you have? ");

if ((choice.equals("y")) || (choice.equals("Y")))
  && (cash >= CASH_LIMIT) )

  output("You said yes and you have " + cash + " dollars");
  output("Come on then");
  
```

```

else
    output("OK let's see what's on TV");

```

A new point first:

```
final static int CASH_LIMIT = 50;
```

final static identifiers like CASH_LIMIT are useful when you have constant, unchanging values in your programs. They improve readability and allow you to make changes more easily. You can expect to see them used in examination questions.

The above could also have been written

```

final static int CASH_LIMIT = 50;

String choice = input("Do you feel like going out? (y/n) ");
int cash = inputInt("How much cash do you have? ");

if ((choice.equals("y")) || (choice.equals("Y")))
    if(cash >= CASH_LIMIT)
    {
        output("You said yes, you have" + cash + " dollars");
        output("Come on then");

    }
else
    output("OK let's see what's on TV");

```

Braces can be used to improve the readability of nested control structures, even where they are not strictly necessary:

```

if ((choice.equals("y")) || (choice.equals("Y")))
{
    if(cash >= CASH_LIMIT)
    {
        output("You said yes, you have" + cash + " dollars");
        output("Come on then");
    }
else

    output("Oh, that won't be enough");

}
else

    output("OK, let's see what's on tv");

```

Java and JETS code can be written very concisely: always choose to make code more readable than more concise. This is excellent practice for university and industry since people may have to review, read or modify your code and they would like to spend as little of their time as possible in understanding it.

Computer Fundamentals

In addition, the teacher in your classroom is a precious resource (really, you will appreciate this even more when writing your dossier!) - don't waste her/his time on concise but hard to understand code.

EXERCISE 2.6

What is the output of this code:

```
int x = 9;
int y = 0;
int z = 3;
char c = 'x';

if (x == 9)
if (y < 3)
if (c != 'x')
    output("flip");
else
    if (z >= 3)
        output ("flop");
    else
        output("fly");
```

Improve its readability by the judicious use of braces.



MULTIPLE SELECTIONS

When making multiple selections, if statements can become quite complicated. Much more useful than the pattern:

```
if
  if
  else
```

is the pattern

```
if
else if
else if
```

sometimes known as an else if chain. This is very useful when multiple choices need to be made. For example, get the user to enter a number of choices and respond to them (a menu):

```
String place = input(" Where do you want to go today: ");

if (place.equals("m")
    output(" To the movies");

else if (place.equals("c"))
    output(" To the cafe");

else if (place.equals("p"))
    output(" To the park");
```

```

else
    output("I don't know where that is");

```

Java and other languages have multiple selection statements such as switch or case that work in a very similar way but we won't use them here as they are not included within JETS. Of course, you can use them in dossier programs where appropriate to do so.

Notice that the chain stops as soon as a true condition is found.

| | |
|--|--|
| <pre> if true this else if true this else if true this else finally </pre> | <i>II if this is true</i> <i>II do this then skip to the end</i> <i>II same pattern here, if true</i> <i>II do this and skip, otherwise</i> <i>II try this one...</i> <i>II none of the above were true</i> <i>II 'default' action</i> |
|--|--|

next statement

EXERCISE 2.7

At a certain store, they sell blank CD's with the following discounts:

- 10% for 120 or more
- .5% for 50 or more
- 1% for 15 or more
- no discount for 14 or less

Complete the outline:

```

int discount = 0;
int quantity = 0;
int quantity = input("Input the number of discs bought: ");
if (????????????????)
    discount = ??;
else if (????????????????)
    discount = ??;
else if (????????????????)
    discount = ??;
else
    discount = ??;

```

Don't forget to add proper braces to the completed exercise.



Computer Fundamentals

Conditional expressions, for example:

```
if (a > b)
    big = a;
else
    big = b;
```

are so common in Java that a special expression is sometimes used (borrowed from the C language):

```
big = (a > b) ? a : b;
```

The general form is:

```
result = <condition>?<result if true>:<result if false>
```

It is a perfectly valid construct and you may see it elsewhere, however, we will not use it here and you will not find it in JETS (and therefore exam questions). There is nothing to stop you using it in exam responses if you wish. Just make sure you know what you are doing and write clearly.

As noted above, the other Boolean operator that we have is `!`, which can be used to reverse a logical value from true to false (or vice versa):

```
if !(MARK >= 0) && (MARK <= 100)
{
    output("Not a valid mark")
```

REPETITION

Repeating things again and again until some condition is met or while some condition is true.
Basic structure of a loop:

```
while <condition>
{
    do this
}
```

where the `<condition>` is exactly the same as for the if test.

```
int x = 1;
while (x < 10)

    output(x);
    x *= 2;
```

How many times is this loop executed?

The less common but sometimes necessary alternative loop is:

```
int x = 1;
do
```

```

    output( x );
    x * = 2;
}
while (x < 10);

```

The difference between the two loops is that the do while loop is always done at least once (because the test to repeat is at the end) whereas the while loop may never be executed.

'TRACING LOOPS

The following technique can be used to sum values:

```

int sum = 0;
int next = inputInt("Input first number: ");

while (NEXT> 0)      // start control loop

    sum = sum + next   // add it in
    next = inputInt("Input next value: ") // get next value

output("the sum is: " + sum); // display result

```

Notice that this requires a list of numbers to be input, ending with a 0 or negative value which will then terminate the loop.

Tracing an algorithm means to work out what the values of variables will be in each step as a means of understanding how it works. You should do this when you are designing algorithms for your dossier and will need to apply the technique in your final examination.

Trace the algorithm with the values 4, 78, 2, -2, 4, 0, 9

| next | next> 0? | sum | output | |
|------|----------|-----|--------|---------------------------|
| 4 | - | 0 | - | initial conditions |
| 4 | TRUE | 4 | - | first time round the loop |
| 78 | TRUE | 82 | - | |
| 2 | TRUE | 84 | - | |
| -2 | FALSE | 84 | 84 | terminates |

Modify the above algorithm to count the number of numbers entered and also give the arithmetic mean (SUM/COUNT). Trace the algorithm with the values 14,54,7,0,23,12,-1.

The for loop will repeat itself a set number of times by counting a variable up or down:

```
for (int x = 4; x > 0; x = x - 1)
```

You will normally see this written:

```
for (int x = 4; x > 0; x--)
```

but the IBO have indicated that they will not use these increment and decrement operators.

Computer Fundamentals

However, you are going to see them and probably use them in your dossiers as they are very handy:

| operator | meaning |
|------------|--|
| $++x$ | increase x by 1, then use the new value of x in evaluation |
| $x++$ | use the old value of x in evaluation then increase by one |
| $--x$ | decrease x by 1, then use the new value of x in evaluation |
| $x--$ | use the old value of x in evaluation then decrease by one |
| | |
| also note: | operators can be combined with assignment |
| $x += 2$ | $x = x + 2$ |
| $x *= 2$ | $x = x * 2$ |
| etc. | |

The for type of loop is most useful when you know in advance how many times the loop is to be repeated. It can always be replaced with either the do while or the while loop.

That is:

```
for (int x = 4; x > 0; x = x + 1)
    y = z - 2;
```

is exactly equivalent to:

```
x = 4
while (x > 0)

    y = z - 2;
    x = x - 1;
```

CLASS ACTIVITY

Set up a trace table to work out what this program does:

```
/*
 * A class which illustrates use of methods, a mystery method
 *
 * @author Richard
 * @version 060903
```

```

*/
public class Mystery
{
    public static void main(String[] args)
    {
        new Mystery();
    }
    /**
     * Constructor for objects of class Class Mystery
     */
    public Mystery()
    {
        String message = input("Please type a message: ");
        output("The result of mystery is: " + mystery(message));

        public int mystery(String message)
        {
            int count = 0;
            int len   = message.length();
            for(int p = 0; p < len; p++)
            {
                if (message.charAt(p) == 'A')
                {
                    count = count + 1;

                return count;

            /**
             * IBO methods, (c) International Baccalaureate 2004
             * Computer Science Subject Guide, Appendix 2.
             */
            include them here ...
        }
    }
}

```



WORKED EXAMPLE

Write an algorithm which will:

1. input a request for two examination marks, mark1 and mark2
2. multiply mark1 by 0.75
3. multiply mark 2 by 0.25
4. add together a new total from the new values for mark1 and mark2
5. output a message that candidate has passed or failed, based on a pass mark of 45 for the new total

Computer Fundamentals

Tip: When writing JETS it's a good idea to leave blank lines in your solution so you can add bits you forgot! Also doing a first run through in pencil helps.

Notice how the code follows the description quite closely:

```
public class PassFail
{
    public static void main(String[ ] args)
    {
        new PassFail();
    }
    /**
     * Constructor for objects of class PassFail
     */
    public PassFail()
    {
        int mark1    = inputInt("Please input the first mark: ");
        int mark2    = inputInt("Please input the first mark: ");

        double m1    = mark1 * 0.75;
        double m2    = mark2 * 0.25;
        int total    = (int) (mark1 + mark2);

        if (total >= 45)
            output("You passed");
        else
            output("You failed");
    }
    /**
     * IBIQ methods, (c) International Baccalaureate 2004
     * Computer Science Subject Guide, Appendix 2.
     */
    include here ...
}
```

CLASSES, USER-DEFINED METHODS AND OBJECTS

We have already seen examples of user-defined methods when we looked at the topic of parameters. The language also has built-in functions like the one that returns the length of a string:

```
int len = message.length();
```

The method length is built in to most high-level languages as well as into the Java String Class. Most high-level languages have a range of built in subprograms; some of them, such as FORTRAN, are particularly rich in mathematical and scientific functions.

Sometimes the term 'function' is used to distinguish a method that returns a single value and thus has a return type. It could be said that the method length is an integer function returning the length of a string of characters.

Notice that the method length requires an object of the String Class to 'work on'. Some methods can be used directly with the name of the Class:

```
double x = Math.pow( 9.0, 2.0);
```

Math is the name of a class rather than an object. This type of method is declared as static - i.e. there is a method somewhere in the Math class like this:

```
public static double pow( double x, double y)
```

Maybe it's time we looked more closely at objects. Consider our Add Class:

introductory comments

```
/**  
 * A class which can add 2 numbers together  
 *  
 * @author Richard  
 * @version 060903  
 */
```

define a new class

```
public class Add
```

main method required for an application class - Le. a class that does something.

```
public static void main(String[] args)  
{  
    new Add();  
}  
/**  
 * Constructor for object:3 of class Add  
*/
```

This is the constructor method.

```
public Add()
```

It has the name of the class

```
double number1  
    inputDouble("Please input your first number: ");  
  
double number2 =  
    inputDouble("Please input your second number: ");
```

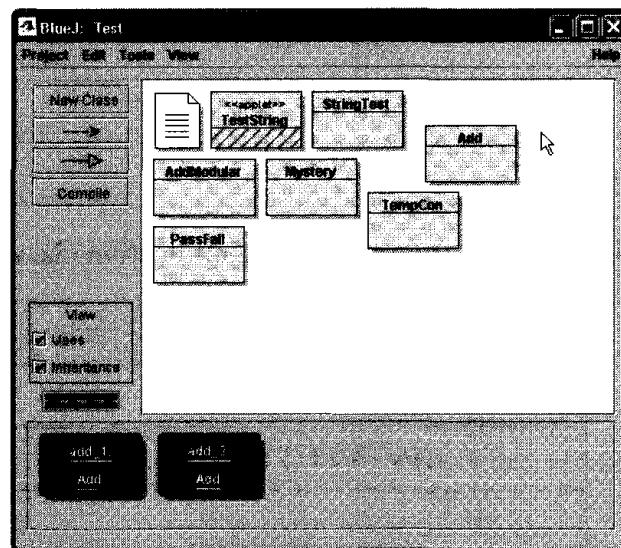
It creates an Add object

```
double total = number1 + number2;  
  
output (The total is: " + total);  
}
```

end of class definition

It is possible to have a Class with no main method as we shall see. A Class is like a blueprint or a plan, it is a definition of how to build something. An object is said to be an instance of a Class. Objects are created via a call to new, followed by the constructor name.

Here is a picture of the BlueJ screen showing some of the classes we have been using to illustrate this chapter:



Notice that there are two add objects (instances) at the bottom of the screen (the workbench). These were produced by running the Add class from the main screen. A Class can be used to create as many objects as needed, just like a blueprint can be used to build many houses.

Examples of objects that don't run are String and Math.

As an illustration of a user-defined object, we are going to create a new data type called Time. This we can declare in our programs just like Strings and primitives:

```
String name = "John";  
Time now = new Time("11", "am")
```

Notice that Strings are a bit special in not having to call new to create a new String object. There are, however, other ways of constructing or creating objects of the String class.

When we have defined the class we will add a method that gives us the difference between two times on the same day to the nearest whole hour. Notice that this Class is of very little use in a

practical program - that's deliberate. We wouldn't want to reduce the fun you will have in developing your own useful Time Class!

Objects need to keep track of their current state, therefore they need identifiers internally; we shall keep the following data about the time:

the hour as a String

- the hour as an int (for calculations)
- the indicator for am and pm as a char (a or p)

If we ask BlueJ to create a new Time Class for us we get the following very nice framework:

```


1** ,
 * Write a description of class Time here.
 *
 * @author (your name)
 * @version (a version number or a date)
 *1
public class Time
{
    // instance variables - replace the example below with your
    ownprivate int x;
1**
 * Constructor for objects of class Time
 *1
public Time()
{
    // initialise instance variables
    x = 0;

1**
 * An example of a method-replace this comment with your own
 *
 * @param y    a sample parameter for a method
 * @return      the sum of x and y
 *1
public int sampleMethod(int y)
{
    // put your code here
    return x + y;
}


```

Teachers (and students) might like to note that they can customize this template (see the BlueJ documentation for details). A customized template for IBIO classes will be found on the IBID Press website.

Computer Fundamentals

The identifiers we need are included in the **Class** definition:

```
1**
 * Class Time is used to store a time as an hour of the day
 *
 * @author Richard
 * @version 070903
 *1
public class Time
{
    II Properties of our Time Class
    private String sHour;      II hour as a String
    private int iHour;         II hour as an int
    private char indicator;   II indicator for am\pm
```

The keyword **private** is used to prevent access from outside the Class. **If** we make them public then they could be changed in a statement like:

```
Time t = new Time();
t.iHour = 11;
```

which we don't want. **If** we allow public access, then another (careless) user of the Time Class could set the **iHour** identifier to an incorrect value (like -34, for example).

The Class itself is public, otherwise it couldn't be used by other Classes. The constructors will also be public for the same reason. We need a constructor for the initialization we presented earlier:

```
Time now = new Time("11", "am")
```

Clearly this takes two String parameters:

```
public Time (String hr, String id)
{
    sHour = hr;
    indicator = id.charAt(0);
```

This is OK but it might be a good idea to check if the user has entered a valid time:

```
public Time (String hr, String id)
{
    II convert the hour string to an int,
    II trim any spaces of id and convert to lower case
    II then check the range of both
    int ih = Integer.parseInt(hr.trim());
    id.toLowerCase().trim();

    if ((ih < 0) || (ih > 11)
        && !((id.equals ("am")) || (id.equals ("pm")))
    {
        II invalid time, set time to midnight
        setTimeMidnight();
```

```

else
{
    // everything OK, set the time values
    sHour = hr;
    iHour = ih;
    indicator = id.charAt(O);
}

```

`Integer` is a special class for dealing with `int` types, it includes the method `parseInt(String)` to convert a `String` to an integer. We also use the helper method `setTimeMidnight` here:

```

private void setTimeMidnight()
{
    sHour = "00";
    iHour = 0;
    indicator = 'a';
}

```

This demonstrates that a method can be private too - this method cannot be accessed from outside the class (no special reason why this one should not be - just wanted to illustrate the point). It is good practice to keep everything private if practical- always operate on a 'need to know basis' as far as external classes are concerned.

We can also supply a 'no argument' constructor:

```

1**
* Constructor for objects of class Time
*1
public Time ()
{
    // initialise instance variables to midnight
    setTimeMidnight();
}

```

So that a new time object can be created with a time of midnight. The method that returns the state of a time object as a `String` needs to be public (why?):

```

1**
* Method to return the time as a String
*
* @return the time as a String
*1
public String toString()
{
    String s_indicator;
    if (indicator == 'a')

        s_indicator = "AM";

    else

```

```
s indicator = "PM";
return (sHour + " " + s indicator);
```

Now we can test that our Time Class actually works:

```
/*
 * A class to test the Time Class
 *
 * @author Richard
 * @version 070903
 */
public class TestTime
{
    public static void main(String[] args)
    {
        new TestTime();
    }
    /**
     * Constructor
     */
    public TestTime()
    {
        String sHour = input("Please input the hour (0-11): ");
        String indicator = input ("Please indicate am or pm (a, p)");
    };
        Time now = new Time (sHour, indicator);
        output("The time is " + now.toString());
    }

    /**
     * IBIO methods, (c) International Baccalaureate 2004
     * Computer Science Subject Guide, Appendix 2.
     */
}

include here ...
)
```

EXERCISE 2.8

Extend the Time Class so that an instance of the class can calculate the difference in hours between itself and another instance:

```
public int timeDifference(Time time2)
```

A program might test the method this way:

```
int timeDiff
Time time1 = new Time("1", "AM");
Time time2 = new Time("8", "PM");
timeDiff = time1.timeDifference(time2);
```

You could use such a class in a dossier for a company that hires out equipment by the hour.



CLASSES AND EXCEPTIONS

This topic is not part of JETS but is included for completeness.

The main idea behind using objects, such as the Time class is to hide the detailed operation of the code from other classes. This is a powerful way of increasing the robustness of software.

However, sometimes things go wrong. Another programmer, misunderstanding the way in which the class works, might try something like this:

```
Time time1 = new Time ("AM", "11");
```

We could simply put an error message within the Time class using our output statement from IEIG. The problem with that is, we don't know if it's being used in a console-based program or not. Also maybe the other programmer doesn't want your error messages cluttering his screen. Therefore, we need a communication mechanism.

Exception objects offer such a mechanism. Exceptions are thrown (constructed) whenever something goes wrong, you might have noticed this if you are running a console-based example and type in an invalid number. For example, typing an invalid number in our TimeTest example results in a NumberFormatException with the message 'invalid integer'. You can see this at the bottom of the code screen if you use BlueJ.

Exceptions can be caught and dealt with. Consider the IEIG inputInt method:

```
static int inputInt(String Prompt)
{
    int result=0;

    try
    {
        result = Integer.parseInt(input(Prompt).trim());
    }
    catch (Exception e)
    {
        result = 0;
    }

    return result;
```

The try block is put around code that can generate an exception. If the exception actually occurs, it can be dealt with in the catch block. More than one exception can be caught if necessary.

If you, as a programmer, don't deal with the exception it is passed up to the next level (in the case of our runnable classes this is dealt with by the System class).

You can also choose to explicitly ignore the exception by having a method throw it (we'll deal with this later).

We can develop our own exceptions by extending Java's Exception Class, for example:

```
/**
 *  TimeClassException.
 *
 *  @author richard
```

Computer Fundamentals

```
* @version 070903
*1
public class TimeClassException extends Exception
{
    public TimeClassException()
    {
        super();
    }
    public TimeClassException(String message)
    {
        super(message);
```

Now we can use this in the Time Class, modify the constructor:

```
public Time (String hr, String id) throws TimeClassException
```

and add this line further down:

```
// invalid time, set time to midnight
setTimeMidnight();
throw new TimeClassException("Error: invalid time");
```

If you try to compile TestTime now, this line will generate an error:

```
Time now = new Time (sHour, indicator);
```

Because the TimeClass Exception has to be dealt with, either caught by the method or thrown:

Add this:

```
public TestTime() throws TimeClassException
```

Now try to compile again. What happens? The main method complains because that is the method that called the TestTime constructor.

Adding this:

```
public static void main(String[] args) throws TimeClassException
```

allows the program to at least compile. However, the TestTime Class can (and should) deal with the problem, maybe like this:

```
try
{
    Time now = new Time (sHour, indicator);
    output("The time is " + now.toString());
}
catch(TimeClassException e)
{
    output(e.getMessage());
```

Although we have used a somewhat roundabout route to achieve the same outcome (posting a message to standard out), the exception mechanism does give more flexibility. It allows the programmer to choose where the exception will be dealt with. Why did we say above that the TestTime Class should deal with the exception rather than any other class, e.g. the System Class?

EXERCISE 2.9

Instead of simply catching the exception and outputting a message, try to modify the TestTime Class so it repeats trying to get a time until a valid time is input by the user.



STATIC METHODS

A couple of times now we have seen methods attached to class names, like this:

```
Integer.parseInt(String)
Math.pow(double, double)
```

This is achieved by use of the keyword static before the member name. As an example, if we wanted to use the **timeDifference ()** method in the Time Class without actually instantiating a Time object, we could declare the method like this:

```
public static int timeDifference(Time time1, Time time2)
```

A program might test the method this way:

```
int timeDiff
Time time1 = new Time ("11", "AM");
Time time2 = new Time("B", "PM");
timeDiff = Time.timeDifference(time1, time2);
```

JETS specifies some standard methods and data members of the String class that students should be familiar with:

| method | return type | description |
|---|-------------|---|
| .equals(String) | boolean | Returns true if the Strings are equal. Do not use the comparison operator (==) with Strings. |
| .substring(int, int) .substring(int) | String | The first returns the string between the start point (first argument) and end point (second) of the String object it operates on. The second version returns the substring from the position to the end of the target String. |
| .length () | int | Returns the length of the String. |
| .indexOf (char) .indexOf (char, int) | int | Returns the position(index) of the first character matching the argument. The second version starts the search at the index specified in the second argument. |

| method | return type | description |
|--------------------|-------------|--|
| .compareTo(String) | int | Compares one String to another for equality, if the argument is less than the target String it returns a negative integer, if greater, a positive integer. If they are equal it returns 0. |
| .toUpperCase () | none | Converts a String object to Upper Case (capital) letters. |
| .toLowerCase | none | Converts a String object to lower case (small) letters. |

Those methods which take a String index value (e.g., substring) throw a `StringIndexOutOfBoundsException` if the index is outside the target String, e.g.:

```
String name = "Hye Rim Nam";
String middle = name.substring(4, 6); // value "Rim"
String error = name.substring(12);    // throws exception
```

Notice that String indices start at 0 (not 1).

Candidates are not expected to memorise these functions for the examination. If an algorithm question requires their use or uses them, a 'recall' description will be given, e.g.:

Recall that `Math.pow(double x, double y)` returns the double value of x raised to the power y .

Two other useful methods which we have already seen are:

| method | return type | description |
|--------------|-------------|---|
| .trim() | none | Removes leading and trailing whitespace (spaces, tabs) from a String. |
| .charAt(int) | char | Returns the character at the position specified in the argument. |

PRIMITIVES VERSUS OBJECT REFERENCES

We referred earlier to the important difference between primitives and objects. The object identifier really contains a reference to an area of memory where the data is stored.

```
Time timel;
```

Creates the reference identifier; no object has been instantiated (created) yet. In order for a new object instance to be created the constructor must be called:

```
Time timel;
timel = new Time();
```

We could create a second reference and copy the address into it:

```
Time time2;
time2 = timel;
```

They both point to (reference) the same area of memory so are, therefore, the same object. If you want to check it out, try this:

```
String sHour = input("Please input the hour (0-11): H);
String indicator = input("Please indicate am or pm (a, p): H);
try
{
    // set up timel with the new time
    Time timel = new Time(sHour, indicator);
    output("The time is " + timel.toString());

    // Create a new time reference (not an Object!)
    Time time2 = timel;

    // Change time2 (and therefore timel)
    time2.setTimeMidnight();
    output("The new time is " + timel.toString());

    // NB I had to change the Time Class to make
    // setTimeMidnight() a public method!
}
catch( /* as before */ )
```

ARRAYS OR LISTS OF VALUES

A array is a group of variables, all of the same type, which share a common name. An example of an array of ints called marks:

| | | | | | | |
|---------|-----|-----|-----|-----|-----|-----|
| marks | 23 | 19 | 56 | 99 | 43 | 82 |
| element | [0] | [1] | [2] | [3] | [4] | [5] |

The array element marks[4] in this example has the value 43. Arrays can hold any of the fundamental data types or can equally well be collections of data structures such as strings or records.

For example a boolean array of equivalent size could be:

| | | | | | | |
|---------|-------|-------|------|------|-------|------|
| status | False | False | True | True | False | True |
| element | [0] | [1] | [2] | [3] | [4] | [5] |

The great advantage of arrays over simple data types is that they can be easily processed inside a loop rather than having separate variable names like mark1,mark2, mark3 etc.

Computer Fundamentals

Arrays can also be multi-dimensional. A two-dimensional array might be useful to hold a set of positions in a board game, for example:

| Board | [0] | [1] | [2] |
|-------|-----|-----|-----|
| [0] | 1 | 1 | 0 |
| [1] | 0 | 0 | 1 |
| [2] | 1 | 1 | 0 |

For a two-dimensional array board[r][c] the indices are in row, column order, thus board[0][1] has the value 1 rather than 0 which is in board[1][0].

Arrays of 3 or more dimensions are also possible but not required for an IB Computer Science course.

One of the most important things to remember about arrays is that all of the elements have to be of the same type.

ARRAY EXAMPLE

A list of names can be placed in an array as follows:

```
String[] names = new String[ 5 ]  
  
for( int i = 0; i < 5; i = i + 1 )  
{  
    names[ i ] = input("please input a name: ");  
  
    output("The second and fourth names input were: " +  
          names[ 2 ] + ", " + names[ 4 ]);
```

Suppose you input the following to this program:

```
please input a name: richard  
please input a name: aruna  
please input a name: minh  
please input a name: sayaka  
please input a name: dorian
```

The output is:

```
The second and fourth names input were: minh, dorian
```

Which might not be quite what you expected. In Java, as in C/C++, arrays are 'zero-based', that is the first element is 0, rather than 1 as it might be in Pascal or PURE.

Therefore names[2] is the third entry:

| | | | | | |
|-------|---------|-------|------|--------|--------|
| names | richard | arona | minh | sayaka | dorian |
| | [0] | [1] | [2] | [3] | [4] |

An array can only hold values of one primitive data type or of one particular class.

EXAMPLE

Devise an algorithm which finds the largest value in a given array (list) of numbers. To help you, here is an algorithm which finds the smallest in a given array of numbers called listA:

listA is assigned the values 3,67, -9,304, -56, 2 as shown in the example:

```
// initialise an int array with values
int[] listA = new int[] {3, 67, -9, 304, -56, 2};
int pos = 1;           // pointer to the array
int smallest = listA[0] // smallest so far

while (pos < 6)

    if (listA[pos] < smallest)

        smallest = listA[pos]
    }
    pos = pos + 1;

output("The smallest number in the list is: " + smallest);
```

Tracing the algorithm inside the loop

| pos | smallest | listA[pos] | listA(pos) < smallest | Comment |
|-----|----------|------------|-----------------------|-----------------------|
| 1 | 3 | 67 | TRUE | small is changed to 3 |
| 2 | 3 | 67 | FALSE | small is unchanged |
| 3 | 3 | -9 | TRUE | small becomes -9 |
| 4 | -9 | 304 | FALSE | small is unchanged |

You should be able to finish off the trace table. When you have completed your algorithm to find the largest value in a given array, draw and complete a similar trace table to confirm that 304 is correctly calculated to be the largest value from the above listA.

EXERCISE 2.10

How could the above algorithms also calculate which position in the array was holding the largest/smallest value? Show where your algorithm needs to be modified to do this.



RECORDS AND CLASSES

A record structure was defined as follows in PURE:

```
newtype STUDENTRECORD record
    FORENAME      string
    SURNAME       string
    BIRTHDATE     string
    SEX           character
    FORM          character
endrecord
```

The record structure is an example of a 'composite data structure' in IB Subject Guide terminology.

Although this term is less apparent in JETS, there is still a dossier mastery aspect related to the use of this type of data structure, so we will consider how it could be implemented in a Java program.

The Classes we have looked at so far have mainly been applications Classes which are run. However, many other types of Class are possible in Java, Classes which don't actually run like applications do. Consider the following VideoTape Class:

```
1**
* A Class to keep details of video tapes
* @author Richard
* @version 200903
*1
public class VideoTape
{
    // instance variables - or data members
    private String title; // title of the video
    private int length; // in minutes
    private boolean lent; // is it lent to someone

    1**
    * No argument Constructor for objects of class VideoTape
    *1
    public VideoTape()
    {
        // initialise instance variables
        this.title = null;
        this.length = 0;
        this.lent = false;
    }
    1**
    * Constructor for objects of class VideoTape
    *1
    public VideoTape(String title, int length, boolean lent)
    {
        setTitle(title);
        setLength(length);
        setLent(lent);
```

```

/*
 * Mutator methods change the objects data fields
 */

/**
 * set the title
 *
 * @param String title the video title
 */
public void setTitle(String title)
{
    this.title = title;
}
/**
 * set the length
 *
 * @param int the video length in minutes
 */
public void setLength(int length)
{
    this.length = length;
}
/**
 * set the status of lent
 *
 * @param boolean lent true if the video is on loan
 */
public void setLent(boolean lent)
{
    this.lent = lent;
}
/*
 * Accessor methods return the objects data fields
 */

/**
 * Return the title
 *
 * @return String the video title
 */
public String getTitle()
{
    return this.title;
}
/**
 * get the length
 *
 * @return int the video length in minutes
 */
public int getLength()

```

```
        return this.length;
    }
    /**
     * get the status of lent
     *
     * @return boolean lent true if the video is on loan
     */
    public boolean isLent()
    {
        return this.lent;
```

Notice that we have used a special keyword to refer to Class data members:

```
public int getLength()
{
    return this.length;
```

This is not strictly necessary but is good practice and avoids any problems where a local identifier or parameter might shadow or mask the Class data member:

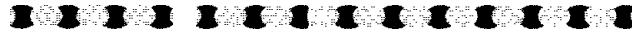
```
public void setTitle(String title)
{
    this.title = title;
```

In the above method, the parameter title has the potential to mask or shadow the Class data member title. This qualifier makes sure the parameter value is assigned to the Class data member.

There is no main method to run this Class - it doesn't actually do anything. However, it is possible to create an array of these objects to store Video Tape data.

EXERCISE 2.11

Add some validation to the VideoTape Class. Get the constructor to throw an exception if invalid data is passed (e.g. the video length is negative).



EXAMPLE OF USING VIDEOTAPE OBJECTS

The private data members of this class are an array of VideoTape objects and the number of videos currently in the collection. The maximum number which can be stored is determined by MAX_VIDEOS, a static final (constant) value. This can be changed to store more.

```
/**
 * A class to store Video Tape data in an array
 *
 * @author Richard
 * @version 200903
```

```
*1
public class VideoStore
{
    static final int MAX VIDEOS = 5;
    private VideoTape[] videoTapeEs = new VideoTape[MAX VIDEOS];
    private int numVideos = 0; // number of videos in collection

    public static void main(String[] args)
    {
        new VideoStore();
    }
}
```

The constructor typifies a menu-driven approach to an application. It loops, getting commands and executing them until a quit command is issued. This is a simple but effective technique for data-based applications.

```
1**
 * Constructor loops until command "quit" is used
 *1
public VideoStore()
{
    char command;
    do

        command = getCommand();
        doCommand(command);

    while (command != 'q');
```

Get command outputs suitable prompts and gets the users command. This is validated and if the command is not recognized, the method loops back and tries again.

```
1**
 * Method to get a valid command from the user
 *
 * @return char, a valid command - a, l or q
 *1
private char getCommand()
{
    char Chi // input command

    // repeat until a valid command is entered
    do
    {
        output("");
        output("Would you like to: ");
        output("(a) add a video tape to the collection");
        output("(l) list the tapEs already in the collection");
        output("(q) or quit thE program");
        output("");
    }
```

```

ch = inputChar("Your choice (a/l/q): ");

if (ch == 0)    II a null char was returned

    output("Please enter a command!");
    ch = 'n';

else

    II check if the value input was a valid command
    if      (ch != 'a') && (ch != 'q') && (ch != '1')

        output("Please enter a valid command (a, l or q) !");
        ch = 'n';

    }

while (ch == 'n');      II repeat outer do loop until
                        II valid command is input

return ch;

```

This method carries out the command passed as an argument (already checked for validity). It uses an else if chain. More commands could be added but then it might be sensible to split the command routines off into their own methods to improve readability and make the solution more modular and therefore more robust.

```

1**
* Method to execute commands (more could be added)
* eg search, lend, delete.
*
* @param command char - the command to be carried out
*1
private void doCommand(char command)
{
    II using an else if chain...

    if (command == 'a')
    {
        II see how many videos are in the collection:
        if (numVideos >= MAX_VIDEOS)

            output("Sorry, no more videos can be stored");
        }
        else

            videoTapes[ numVideos ] = getVideoDetails();
            numVideos = numVideos + 1;
    }
}

else if (command == 'l')

```

```

// loop through array
for(int x = 0; x < numVideos; x = x + 1)
{
    output(" " + x + ": " + videoTapes[ x ].toString());
}
}
else if (command == 'q')

    output("Bye then");

else

    output ("Some internal error in doCommand() ");

```

Here we see a method that returns an object reference rather than a primitive. This requires us to actually create an object within the method body (a new VideoTape object is instantiated).

```

1**
* Method to get details of tape
*
*@return VideoTape - a video tape object
*1
private VideoTape getVideoDetails()
{
    String title = input("Enter the title of your video: ");
    int length = inputInt("Enter the length in minutes: ");
    // We assume it is not yet lent
    return new VideoTape(title, length, false);
}
1**
* IPIO methods, include here
*1

```

EXERCISE 2.12

Before we extend the above example to using data files you will find it useful to simplify the doCommandO method to call helper methods for adding, listing and quitting. True there isn't much to do with quit as yet but there could be in the future.

Further exercises might involve a method to delete a video tape from the array, two techniques you can try are:

The shuffle -locate the record to be deleted, shuffle down records from there to the end of the file. Don't forget to update numVideos! Take care with boundary conditions (like the last record in the array).

The blank out -locate the record to be deleted and mark it in a special way (e.g. use "xxx" for the title). Don't forget to check for an "xxx" location that can be used when adding new tapes rather than adding to the end. Handling numVideos and finding an insertion point becomes a lot more tricky in this case.

Which method is more efficient in time taken? Which is more efficient in use of storage space?



DATA FILES

The IE Subject Guide refers to Random Access and Sequential Files. Java (and therefore JETS) has no tools for reading records from, or writing records to, a file. Rather it deals with character-based streams or streams of bytes that might represent other data types. Either way, it is up to the programmer to extract structure from a stream.

Random access files, based around byte streams, are needed by HL students and will be dealt with later. SL students can simply use sequential file methods to write character-based streams into and out of text files. Alternatively, the serializable interface (not part of JETS) can be used to convert an array of record objects into a stream that can be written to a data file in one go. Even though this is outside the scope of the syllabus it is so simple and powerful that it is an ideal method for SL students who have no interest in more complex methods.

In fact, the different streams available in the `java.io` package are very complex and versatile and whole books have been written on that subject alone. Here we simplify matters (we hope!).

CHARACTER-BASED FILESTREAMS

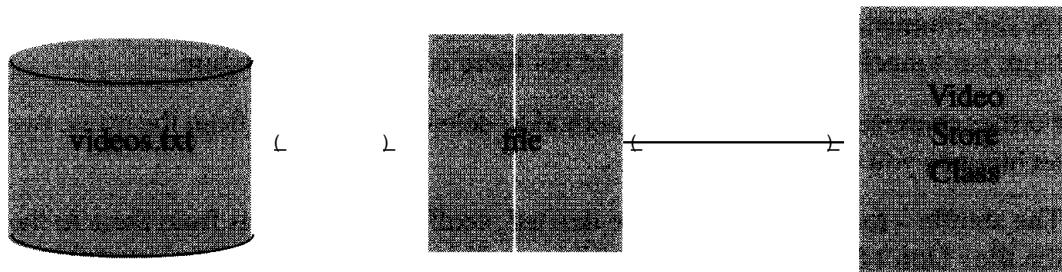
The advantage of these streams in a file handling application is that simple text files can be created and then examined using simple editing tools like Notepad. Students can actually see what gets into the file, very useful for debugging. Java stores characters internally using the 16 bit UNICODE character set but the character stream classes can convert these to 8 bit codes such as ASCII. Normally this process doesn't require any intervention by the programmer.

The character based streams are based on two superclasses:

- Reader
- Writer

Of interest to us, for simple applications are `FileReader` and `FileWriter` Classes which read and write the character-based streams to and from data files.

When using data files, there has to be a link between the program and the local operating system where the file is actually stored:



This statement links the program to a file:

```
FileWriter theFileID = new FileWriter(FILENAME);
```

The `FILENAME` is a String which describes the location of a file on your system.

In Windows you can use something like:

```
static final String FILENAME = "g:\\bluej\\videos.txt";
```

To define the location of the file. Notice that the delimiter for folders (directories) uses the double backslash to 'escape' the escape character (as briefly mentioned in Strings, above).

To format the text as ASCII characters we use the PrintWriter Class which is similar to the output methods we have been using in JETS. The PrintWriter Class can create an object in which the output is directed to a FileWriter object:

```
PrintWriter outFile = new PrintWriter(theFileID);
```

After which items can be written to the file as follows:

```
outFile.println("A string item");
```

One problem with dealing with files and operating systems beyond the control of our program! class is that things can go wrong. Maybe the file cannot be found or created. For example, if it is a floppy disc, maybe there is no space or the disc has been removed.

To handle i/o errors of this type, Java uses the exception handling techniques discussed above. The FileWriter constructor can throw an IOException if there are problems with the physical file our code refers to.

To write a save command for the VideoStore we can use the following techniques:

```
try
{
    // set up data file for writing
    FileWriter theFileID = new FileWriter(FILENAME);
    PrintWriter outFile = new PrintWriter(theFileID);

    // loop through array, writing out records
    for(int x = 0; x < numVideos; x = x + 1)
    {
        outFile.println(videoTapes[ x ].getTitle());
        outFile.println(videoTapes[ x ].getLength());
        outFile.println(videoTapes[ x ].isLent());
    }
    outFile.close();
}
catch (IOException io)
{
    output("Error writing to the data file - "
          + io.getMessage());
```

If we look inside the file videos.txt after a sample run of the program:

```
Your choice (a/l/s/q): a
Please enter the title of your video: Hello Dolly
Please enter the length in minutes: 123
```

Computer Fundamentals

Would you like to:

- (a) add a video tape to the collection
- (l) list the tapes already in the collection
- (s) save the tapes to a data file
- (q) or quit the program

Your choice (a/l/s/q): a

Please enter the title of your video: Goodbye Charlie

Please enter the length in minutes: 211

Would you like to:

- (a) add a video tape to the collection
- (l) list the tapes already in the collection
- (s) save the tapes to a data file
- (q) or quit the program

Your choice (a/l/s/q): 1

0: Hello Dolly - 123 - false

1: Goodbye Charlie - 211 - false

Would you like to:

- (a) add a video tape to the collection
- (l) list the tapes already in the collection
- (s) save the tapes to a data file
- (q) or quit the program

Your choice (a/l/s/q): s

Would you like to:

- (a) add a video tape to the collection
- (l) list the tapes already in the collection
- (s) save the tapes to a data file
- (q) or quit the program

Your choice (a/l/s/q): q

Bye then

We find:

Hello Dolly
123
false
Goodbye Charlie
211
false

Now, the only problem is to get them back into the program on starting again. If there is a Writer

Class then there must be a Reader Class (yes) and a PrintReader Class (no).

To read character data from a file we can use the BufferedReader Class. This provides a `readLine()` method which is analogous to the `println(String)` method of PrintWriter. I can't explain the system of naming the java.io Classes!

The following method, called when the application starts up, checks to see if a data file exists and reads in records from it:

```

private boolean checkForFile()
{
    II See if a file already exists
    try
    {
        FileReader theFileID = new FileReader(FILENAME);
        BufferedReader inFile = new BufferedReader(theFileID);

        String line;      II String read from file
        int length;       II length converted to int
        boolean lenth;    II lenth converted to boolean
        int x = 0;         II counter for number of entries in file

        II The title is in the first line
        II readLine returns null if the eof is reached
        while ( (line = inFile.readLine()) != null
        {
            II get the next two fields and construct a new object
            length = Integer.parseInt(inFile.readLine());
            lenth = Boolean.getBoolean(inFile.readLine());

            videoTapes[ x ] = new VideoTape (line, length, lenth);
            x = x + 1;

            numVideos = x;
            inFile.close();
            return true;
        }
        catch( IOException io)
        {
            output("Error trying to open file" + io.getMessage());
        }
        return false;
    }
}

```

We add the following to the start of the constructor, before calling the `getCommand-doCommand` loop:

```

if (checkForFile())
{
    output("Data file located and loaded with"
          + numVideos + " existing records\n");
}

```

```
else  
    output("No data file found\n");
```

When the program starts it reads in the two records we left in the data file:

Data file located and loaded with 2 existing records

Would you like to:

- (a) add a video tape to the collection
- (l) list the tapes already in the collection
- (s) save the tapes to a data file
- (q) or quit the program

Your choice (a/l/s/q): 1

0: Hello Dolly - 123 - false
1: Goodbye Charlie - 211 - false

The key methods of the Reader and Writer streams in JETS (which can be used in examinations) are:

BufferedReader(FileReader)

| method | description |
|----------|---|
| ready | Tell whether this stream is ready to be read. |
| read | Read a single character. |
| readLine | Read a line of text. |
| close | Close the stream. |

<http://java.sun.com/j2se/1.4.1/docs/api/java/io/BufJeredReader.htm>

PrintWriter(FileWriter)

| method | description |
|---------|--|
| print | Print a string (or other primitive or object as a String) - polymorphic. |
| println | As above but terminate the line with a newline character |
| close | Close the stream. |

These file streams can only be accessed sequentially - no constructs exist for moving directly to a given line. This implies operations such as binary search or sorting are not possible with character

based files. In turn, this implies that SL students need very little experience of file handling in dossiers or for algorithms in examination papers.

EXERCISE Z.13

Implement the save and load file commands for the VideoFileDatabase application. You can either load on entry to the program (as illustrated above) or, for the adventurous, you can try to add an open command which takes the name of a file to be opened (implies you can specify a filename when you save).



If SL students can successfully complete all the exercises in this section, they are ready to start thinking about dossier problems. HL students still need to complete the remainder of Chapter 5.

HIGHER LEVEL FILE HANDLING

HL students need to be able to use the key methods of the RandomAccessFile Class. At the time of writing these were not completely specified in JETS but the following will probably be needed for the dossier, if not for examination questions.

`RandomAccessFile(String, String)`

The constructor takes a filename as the first argument and an access mode as the second.

| method | description |
|-------------------------------------|---|
| <code>seek(long)</code> | Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. N.B. this method takes a long primitive as an argument, not an int. |
| <code>long length ()</code> | Returns the length of this file in bytes. |
| <code>read ()</code> | Reads a byte of data from this file. |
| <code>readFully (byte[])</code> | Reads b.length bytes from this file into the byte array, starting at the current file pointer. |
| <code>int readInt ()</code> | Reads a signed 32-bit integer from this file. |
| <code>double readDouble ()</code> | Reads a double from this file. |
| <code>Boolean readBoolean ()</code> | Reads a boolean from this file. |
| <code>write (byte)</code> | Writes the specified byte to this file. |
| <code>writeBytes(String)</code> | Writes the string to the file as a sequence of bytes. |
| <code>writelnt (int)</code> | Writes an int to the file as four bytes, high byte first. |

| method | description |
|------------------------------|---|
| writeDouble(double) | Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first. |
| writeBoolean(boolean) | Writes a boolean to the file as a one-byte value. |
| close () | Close the stream. |

<http://java.sun.com/j2se/1.4.1/docs/api/java/io/RandomAccessFile.html>

To illustrate these methods we adapt the VideoTape example to store records in a random access file rather than an array. The first consideration is the lack of a record structure in Java – the random access file is an unstructured stream of bytes. In order for us to know, easily, where record components are stored in the file we need to use fixed length objects.

The file maintains a pointer, which can be manipulated via the seekO method. To calculate where one item finishes and the next item starts we need to know the length of our items.

Suppose we fix a title for a video to be 25 characters long, the length is stored as an int (4 bytes) and the lent field is Boolean (1 byte).

The record unit looks like this:

| field | title | length | lent |
|----------|----------------|---------|--------|
| length | up to 25 bytes | 4 bytes | 1 byte |
| 30 bytes | | | |

These records are stored in a data file:

| | | | | | | | |
|-------------|--------|--------|--------|--------|--------|-------|---|
| record: | 0 | ————— | 2 | ————— | 3 | ————— | 4 |
| start point | 0 | 30 | 60 | 90 | 120 | | |
| seek to | 0 * 30 | 1 * 30 | 2 * 30 | 3 * 30 | 4 * 30 | | |

Therefore, as long as the records are all the same length, any component in the file can be found by multiplying its position number by the record length.

Unfortunately, the title is a String and it only occupies as many bytes as needed:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| h | e | 1 | 1 | 0 | | d | 0 | 1 | 1 | Y | | | | |
| g | o | 0 | d | b | Y | e | | c | h | a | r | 1 | i | e |

The first String takes 11 characters, the second 15. If we try to store these in a file we won't know where the start and end of each record is.

A solution is to fix the maximum number of characters that can be used in a title and pad out the end of the String with spaces:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|-----|---|---|---|---|----|-----|-----------|-----|-----------|
| h | e | 1 | 1 | 0 | <s> | d | 0 | 1 | 1 | y | <s> | <s> | <s> | <s> |
| g | o | o | d | b | y | e | | c | h | a | r | 1 | i | e |

In Java, we can do this by using a StringBuffer object whose length can be fixed. One approach to file handling is to extend our existing record structure VideoTape to handle reading from and writing to file. The length of the components can also be set in this Class. One advantage of extending the Class is that we do not have to re-write the existing methods:

```
import java.io.*;
/*
 * A Class to keep details of VideoTapes, can read
 * and write records from an open RandomAccessFile
 *
 * @author Richard
 * @version 200903
 */
public class VideoTapeFile extends VideoTape
{
```

II instance variables - or data members

II These are determined by the structure of the VideoTape Class

```
static final int TITLE_BYTES = 25;    II length of video title
static final int LENGTH_BYTES = 4;    II length int 4 bytes
static final int LENT_BYTES = 1;       II lent is boolean - 1 byte
```

II Calculate length of a record - used to seek to correct position

II in the data file.

```
static final int RECORD_BYTES = TITLE_BYTES
                        + LENGTH_BYTES
                        + LENT_BYTES;

/*
 * No argument Constructor, calls VideoTape constructor
 * super = a call to the super class constructor
 */
public VideoTapeFile()
{
    super();
    super.title = fixLength(title, TITLE_BYTES);
}

/*
 * Constructor for objects of class VideoTape
 */
public VideoTapeFile(String title, int length, boolean lent)
{
    super(title, length, lent);
```

Computer Fundamentals

```
super.title = fixLength(title, TITLE_BYTES);
```

This is the method that fixes the length of the title String by using a StringBuffer instance and the setLength() method of that Class.

```
* Method to fix length of title String
*
* @param title - the original String
* @param size - the fixed length size in bytes
* @return a String of exactly size bytes
*/
private String fixLength(String title, int size)
{
    StringBuffer sb = new StringBuffer(title);
    sb.setLength(size);
    return sb.toString();
```

Writing the data file involves moving to the correct position, calculated elsewhere, before this method is called and then writing out the data fields.

```
/**
 * Method to add item to data file
 *
 * @param the file to add to - must be open for writing
 * @param the object to be written into the file
 * @param position to add the record in the file
 * @throws IOException if unable to complete
*/
public static void writeRecord( RandomAccessFile theFile,
                                VideoTapeFile theVideo,
                                long posToAdd )
throws IOException

theFile.seek(posToAdd);
theFile.writeBytes(theVideo.getTitle());
theFile.writeInt(theVideo.getLength());
theFile.writeBoolean(theVideo.isLent());
```

Notice that the method throws an IOException if there is any problem reading from the file. Again, this is dealt with elsewhere. Reading from a file is a bit more involved since the String has to be re-created from a byte array. Since the length was fixed on writing the record, the number of bytes that need to be read is known.

```
/**
 * Method to read item from data file
 *
 * @param the file to add to - must be open
 * @param long the position to start reading in the file
 * @return a VideoTapeFile object (null if not read)
```

```

* @throws IOException if unable to complete the read operation
*/
public static VideoTapeFile readRecord( RandomAccessFile
                                         theFile,
                                         long posToRead )
                                         throws IOException

theFile.seek(posToRead);
// Read a String by filling a byte array then converting
byte[] theTitle = new byte[ TITLE_BYTES]
theFile.readFully(theTitle);
String title = new String(theTitle);
// Remaining fields can be read directly
int length = theFile.readInt();
boolean lent = theFile.readBoolean();
// construct and return a VideoTapeFile object
return new VideoTapeFile(title, length, lent);

```

This method is used by the main database class to calculate the point at which each component starts and finishes.

```

/**
 * Method to return record length
 *
 * @return length of a VideoTapeFile object in bytes
 */
public static int length()
{
    return RECORD_BYTES;
}

```

Having got a class which represents a fixed length record and which can write its own objects into a data file simplifies the logic of the main program:

The main Class contains the file object and the number of videos in the collection as before:

```

import java.io.*;
/**
 * A class to store Video Tape data in a random access data
file
*
* @author Richard
* @version 210903
*/
public class VideoStoreRandom
{
// The filename, number of videos in the file, the file itself
static final String FILENAME = "c:\\bluej\\videos_r.dat";
private int numVideos = 0;
private RandomAccessFile theFile;

```

Computer Fundamentals

```
public static void main(String[] args)
{
    new VideoStoreRandom();
```

The constructor is also the same:

```
1**
 * Constructor loops until command "quit" is used
 *1
public VideoStoreRandom()
{
    if (checkForFile())
    {
        output ("Data file located and loaded with" + numVideos
               + "existing records\n");
    }
    else

        output("No data file found, starting new collection\n");
}
char command;
do
{
    command = getCommand();
    doCommand(command);
}
while (command != 'q');
}
1**
 * Method to check for existing file
 *
 * @return boolean true if there is an existing file
 *1
```

Check for attempts to open the file and calculate the number of components from the total length of the file in bytes and the length of each record. The file length is returned as a long primitive so the result is cast to an int.

```
private boolean checkForFile()
{
    // See if a file already exists
    try
    {
        theFile = new RandomAccessFile(FILENAME, "rw");
        numVideos = (int) theFile.length() / VideoTapeFile.length();
        return true;
    }
    catch( IOException io)
    {
        output("Error trying to open file" + io.getMessage());
    }
```

```
return false;
```

The `getCommand` method is unchanged (except that we no longer have a save option - all operations take place directly to file. So we have left it out of this listing along with `doCommand` that just calls the following methods:

```
)
1**
 * Method to add a video tape
 *1
private void add()
{
    try
    {
        VideoTapeFile theVideo = getVideoDetails();
        long posToAdd = numVideos * VideoTapeFile.length();
        VideoTapeFile.writeRecord(theFile, theVideo, posToAdd);
        nurnVideos = nurnVideos + 1;
    }
    catch(IOException io)
    {
        output("Error adding record" + io.getMessage());
    }
}
```

The error thrown by `VideoTapeFile.writeRecord` is caught here and also in listing the records in the file:

```

1**
 * Method to list video tapes in the collection
 *1
private void list()
{
    try
    {
        // loop through datafile
        for(int x = 0; x < nurnVideos; x = x + 1)
        {
            int posToRead = x * VideoTapeFile.length();
            output(" " + x + ": "
                  + VideoTapeFile.readRecord(theFile, posToRead)
                  .toString());
        }
    }
    catch(IOException io)
    {
        output("Error during read - " + io.getMessage());
    }
}
```

quit also needs to deal with a possible `I/O Exception` when closing the Random Access file -

Computer Fundamentals

although there is not much we can do if an error occurs at this stage of the program:

```
1**
 * quit the program
 *1
private void quit()
{
    try
    {
        theFile.close();
    }
    catch(IOException io)
    {
        output("Some error closing data file - "
              +io.getMessage());
    }
    finally
    {
        output("Bye then");
    }
}
```

The `getVideoDetails` method is also little changed, it returns a `VideoTapeFile` object rather than a `VideoTape` object.

```
1**
 * Method to get details of tape
 *
 * @return the video tape object
 *1
private VideoTapeFile getVideoDetails()
{
    String title = input("Enter the title of your
video: ");
    int length = inputInt("Enter the length in
minutes: ");
    // We assume it is not yet lent
    return new VideoTapeFile(title, length, false);

1**
 * I BIO methods, (c) International Baccalaureate 2004
 * include here...
 *1
```

Notice that the file expands as new records are added at the end. You can delete records by shuffling, Java 2 now has a method to truncate a `RandomAccessFile` so you can actually shorten the file if you need to, but its use is beyond the scope of JETS.

EXERCISE 2.14

The application presented here is a very bare bones dossier program - deliberately so - we really don't want to see our code in a student dossier! As an exercise, students could add methods to navigate through the file (probably better done with a GUI), search the file, edit and delete records.

Further enhancements could involve adding a key field to the record structure and maintaining the file as a sequential random access file – this involves inserting records at the correct position and deleting them by shuffling.



©IBO 2004 2.14 TRACE ALGORITHMS IN JAVA

As an example, we will consider the useful process of maintaining a full index to a data file (or another array). For High Level students this topic is included in Chapter 8 but adventurous Standard Level students might find it a useful dossier technique as well.

A Class has the following data members:

```
public class BookIndex
{
    // instance variables - or data members
    private String bookNum; // book reference number
    private int pos;         // book details location in main file
```

The BookIndex Class has appropriate accessor and mutator (set and get) methods. Another Class declares an array of these objects:

```
BookIndex[] bookIndexes = new BookIndex[ 500] ;
```

This structure is used in a library containing fiction and non-fiction books. There are several shelf units in the library, each labelled with a letter from A to H. Each shelf unit has four shelves in it.

The BOOKNUM field has the following structure:

The first character indicates a fiction or non-fiction book;

The second character indicates the shelf unit letter (A to H);

The third indicates the number of the shelf (1 is at the bottom, 4 is at the top).

The last three numbers indicate a subject code.

An example of a book number is NG2023

The POS field indicates where the rest of the details of the book are located in a direct access data file.

When a new book is added to the library an entry is made at the end of the data file. The book number is then inserted in the correct position in the array.

Computer Fundamentals

Lets assume the array contains the following data:

| element | 0 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---------|--------|--------|--------|--------|--------|--------|--------|-----|
| BOOKNUM | FA1021 | FA1122 | FA1233 | FA2099 | FA2103 | FA2145 | FA2238 | ... |
| POS | 3 | 6 | 2 | 5 | 0 | 4 | 1 | ... |

A new book, with reference number FA2139 is added to the end of the data file at a position contained in an identifier newPos. This book now needs to be added to the array. Consider the following algorithm:

```

private void insert(String bookNum, int newPos)
{
    int i = newPos;           II pointer to the array
    II Shuffle up array entries from top end to entry position.
    II This will find the next smallest to bookNum (or first element)
    while ((i > 0)
        && (bookNum.compareTo(bookIndexes[ i ].getBookNum()) < 0) )

        bookIndexes[ i ] = bookIndexes[ i - 1 ];
        i = i - 1;
    }
    II Check boundary condition at i == 0
    if (i != 0)

        i = i + 1; II insert above smaller entry
    }
    bookIndexes[ i ] = new BookIndex(bookNum, newPos);
}

```

Assuming that the array is as shown and the next vacant location in the file is 7, trace the algorithm given above.

| newpos | bookNum | i | book | <compare condition> |
|--------|---------|---|------------|---------------------|
| | | | Indexes[i] | |
| 7 | FA2139 | 7 | FA2238 | TRUE |
| 7 | | 6 | FA2145 | TRUE |
| 7 | | 5 | FA2103 | FALSE |

At this point the array looks like:

| 0 | 2 | 3 | 4 | 5 | 6 | 7 |
|--------|--------|--------|--------|--------|--------|--------|
| FA1021 | FA1122 | FA1233 | FA2099 | FA2103 | FA2103 | FA2145 |
| 4 | 7 | 3 | 6 | 1 | 1 | 5 |

and the value of i is 4. Therefore we execute:

```

if (i != 0)
{
    i = i + 1; // insert above smaller entry
}
bookIndexes[ i ] = new BookIndex(bookNum, newPos);

```

Thus bookNum FA2139 and the value 7 are inserted into the correct position (5) in the array.

Q1B0:z_1.5 EVALUATE ALGORITHMS IN JAVA

Consider an array NAMES which looks like this:

| | | | | | | |
|-------|-------------|-------------|--------|--------|---------------|--------|
| names | <u>XXXX</u> | <u>XXXX</u> | sayaka | jithan | <u>XX_X_X</u> | andrew |
| | 0 | 1 | 2 | 3 | 4 | 5 |

An algorithm is to be designed which will move entries which are not 'XXXX' to the front of the array, like this:

| | | | | | | |
|-------|--------|--------|--------|-------------|-------------|-------------|
| names | sayaka | jithan | andrew | <u>XXXX</u> | <u>XXXX</u> | <u>XXXX</u> |
| | 0 | | 2 | 3 | 4 | 5 |

The following algorithm attempts to do this:

```

1**
 * The Shuffle Class
 *
 * @author Richard
 * @version 021003
 */
public class Shuffle
{
    static final String MARKER = "XXXX";
    static final int LAST_P = 5;
    String[] names = new String[] { MARKER, MARKER, "sayaka",
                                    "jithan", MARKER, "andrew"

    public static void main(String[] args)
    {
        new Shuffle();
    }
    /**
     * Constructor
     */
    public Shuffle()
    {
        shuffle();
        showArray();
    }
    /**
     * shuffles names not matching special marker to front
     * of array

```

Computer Fundamentals

```
*  
*1  
public void shuffle()  
{  
    int i, j; II pointers to array names  
    i = 0;  
    j = 0;  
  
    II loop until end of array is reached  
  
    while (j <= LAST_P)  
    {  
        II Advance pointers to first XXXX entry  
        if (!names[ i ] .equals (MARKER) )  
  
            i = i + 1;  
            j = j + 1;  
        }  
        else  
        {  
            II advance pointer to first non-XXXX entry  
            while (names[ j ] .equals (MARKER) )  
  
                j = j + 1;  
            }  
            II place valid entry in XXXX position  
            II replace entry with XXXX  
            names[ i ] = names[ j ]  
            names[ j ] = MARKER;  
        }  
        II advance the pointers  
        i = i + 1;  
        j = j + 1;  
    }  
    *1**  
    * list the names array on the screen  
    *1  
public void showArray()  
{  
    for(int x = 0; x < LAST_P; x = x + 1)  
  
        output(x + " " + names[ x ]);  
    }  
    *1**  
    * IBIQ methods, (c) International Baccalaureate 2004  
    * (include here)  
    *1
```

EXERCISE 2.16

- a. Trace this algorithm to demonstrate that it works for the data set given above.
- b. Trace this algorithm to demonstrate that it does not work for this data set:

| | | | | | | |
|-------|------|------|--------|------|---------|------|
| names | XXXX | XXXX | phuong | XXXX | richard | XXXX |
| | 2 | 3 | 4 | 5 | 6 | |

- c. Supply the required correction to the algorithm.
- d. Outline test data needed to ensure that the revised algorithm works for all possible (valid) data sets. One example would be an array with all XXXX entries.



The methods of the String Class that students are expected to know were defined earlier in this chapter, the list is:

| | |
|---------|----------------------|
| boolean | .equals(String) |
| String | .substring(int, int) |
| String | .substring (int) |
| int | .length () |
| int | .indexOf (char) |
| int | .indexOf(char, int) |
| int | .compareTo(String) |
| void | .toUpperCase () |
| void | .toLowerCase |

All of the methods require an instance of the String Class to work with and return the indicated data type (or are void).

COMMON PITFALLS IN USING STRING INSTANCES AND METHODS

Java's (and therefore JETS's) use of String instances is somewhat different to PURE and to other languages, such as C, C++ or Pascal. The String instance is NOT an array of characters but is an 'immutable' object - once instantiated (created) a String object cannot be changed. Therefore you cannot do the things you might expect to be able to do. For example:

```
// unworkable code
String name = "Saddam Hussein";
String character = name[ 0 ] + name[ 11 ] + name[ 11 ] + name[ 13 ] ;
```

Instances of the StringBuffer Class can be used where an array of characters is required, see the Tokenize example below.

You cannot test two Strings for equality using the == operator since the String is an object instance, you merely test their references. Therefore you MUST use the equals method (below).

```
// unexpected result
String name = "Saddam" + " Hussein";
String object = new String("Saddam Hussein");
boolean found = (name == object);
```

```
// expected result
```

```
String name = "Saddam Hussein";
String object = "Weapons of mass destruction";
boolean notFound = (name.equals(object));
```

When a String instance is modified in some way (added to, for example) the old String is thrown away and a new one created with a new reference. The old String instance is eventually removed from memory by 'garbage collection' - a topic covered further in Chapter 5.

CONCATENATION

This means joining two string together so that, for example:

```
String newString = "2" + "2";
```

Places the value "22" in newString. Notice that characters cannot simply be concatenated to new String instances (char is a primitive type):

```
String newString = '2' + '2'; // compilation error
```

However, it is possible to add a character to a String instance:

```
String newString = "2" + '2'; // no compilation error
```

This is the same as implicitly converting other types to strings when using the concatenation operator (+). An example might be to capture a person's first initial and surname and then store them in a different form:

```
String initial;
String surname;
String name;

surname = input("What is your last name?");
initial = input("What is the first initial of your first
                name?");
name = surname + ", " + initial;
```

SUBSTRINGS

A substring is part of a larger string as in:

```
String name = "Henry the Human Fly"
String middle = name.substring(10, 15)
```

Will place the value 'Human' in the object into middle. - notice the space at the end. Substring can also be used with one argument, in which case it returns the letter d from a specified position up to the end of the String, thus:

```
String name = "Henry the Human Fly"
String middle = name.substring(10)
```

Places "Human Fly" into middle.

EXAMPLE OF STRING MANIPULATION

Suppose we wished to 'tokenise' a string such as the one given above. To 'tokenise' means to split it into its components – in this case we would split it at spaces. This can be a very useful function in programs that analyse the syntax of human or computer languages.

Class data members/definitions:

```

II maximum tokens in a string and length of longest single
token
private static final int MAX = 6;
private static final int LONGEST TOKEN    20;
private static final char SPACE = , ;      II token separator

II StringBuffer and StringBuffer array to hold message and data
II Class could be adapted to tokenize String input by user
II to investigate limitations of this example by a suitable
II set of test data

II new array of StringBuffer references
private StringBuffer[] data = new StringBuffer[ MAX]
private StringBuffer message = new StringBuffer("Henry the
Human Fly");

```

A tokenize method:

```

private void tokenize(StringBuffer message)
{
    int pas = 0;                      II position in the String
    int count = 0;                     II element in data array
    int len   message.length();        II length of initial String

    II loop through message until end or no more space for tokens
    while ( (pas < len) && (count < MAX) )
    {
        II Original array just a set of empty object references,
        II so construct new instance for each array element
        data[ count] = new StringBuffer(LONGEST_TOKEN);

        II find end of current token, marked by a space
        while ( (pas < len) && (message.charAt(pos) !=(SPACE)))
        {
            data[ count] .append(message.charAt(pos));
            pas = pas + 1;

            count = count + 1;
            pas = pas + 1;

```

EXERCISE 2.17

- a) What test data would you use to 'dry run' or trace the above algorithm?
- b) Does it work for all items of test data?
- c) What happens if the entered string has more than MAX words in it?
- d) The algorithm does not cater for punctuation - commas or full stops (periods) will be included as part of the word. Outline how it could be modified to take these into account.



OTHER METHODS OF THE STRING CLASS

We have used various methods in previous chapters and the following example shows how the remaining methods might be used in a small application. The user is expected to type in a series of single character commands.

```

1**
 * A class to do illustrate some methods of the String Class
 *
 * @author Richard
 * @version 041003
 *1
public class StringMethods
{
    private String message
        = "Default Text Message for Use with the Class";
    private String commands = "SULHQ";
    private char command = , ';

    public static void main(String[] args)
    {
        new StringMethods();
    }
    1**
     * Constructor
     *1


```

This section of the program gets a message from the user (or uses the built-in message defined as a class data member), then gets and executes commands in a command loop.

```

public StringMethods()
{
    message = getMessage();
    while (command != 'Q')
    {
        command = getCommand();
        doCommand(command);

private String getMessage()
{
    // User types a message, if not, default is used
    String msg = input("Please type in a text message: ");


```

```

if (msg.length() == 0)
{
    msg = message;

return msg;

private char getCommand()
{
    String cmd =
        input("Please input a command
              (H for help on commands): ");
    if (cmd.length() == 0)
    {
        return ' ';
    }
    else

        return cmd.charAt(0);
}

```

Uses indexOf () function to see if command entered by user is in the String object commands

```

private void doCommand(char cmd)
{
    // lowercase letter returns first position of that letter
    // in the String, if any; matches both lower and uppercase
    // letters in target String

    if ((cmd >= 'a') && (cmd <= 'z'))
    {
        String testMessage = message.toLowerCase();
        int pos = testMessage.indexOf(cmd);
        output("That letter is found at position" + pos);

    }
    else

        // check for valid command
        if (commands.indexOf(cmd) < 0)

            output("Unknown command: " + cmd);
        }
        else

            if (cmd == 'S')

                output(message);
            )
            else if (cmd == 'U')
            {
                message = message.toUpperCase();
}

```

```

        else if (cmd == 'L')
            message = message.toLowerCase();
    }
    else if (cmd == 'H')
    {
        output("S - show the string");
        output("U - convert string to Uppercase");
        output("L - convert string to Lowercase");
        output("H - show this list");
        output("Q - end the program");
    }
    else if (cmd == 'Q')
    {
        output("bye then");
    }
}
/***
 * IBI0 methods, (c) International Baccalaureate 2004
 * (include here).
 */

```

EXERCISE 2.18

- a) Extend the command list to count the words in the sentence, perhaps by modifying the tokenize method given earlier. You might prefer to investigate Java's String SplitO method for this exercise.
- b) Extend the command list to include creating a title case option (that is, converting each individual word so that it starts with an uppercase letter).
- c) The search for letter could be extended (or a new one introduced) that counts the number of times the letter occurs in the String. One version of indexOf takes two int arguments, the second being a start offset for the search.
- d) An edit command, which replaces the existing message with a new message (if valid) could easily be introduced via a call to getMessageO.



This type of command line interface is typical of some older editors used in Operating Systems before the widespread use of GUI's. An interesting dossier project based around this structure could be a line-based text editor. A number command brings up a specified line which can then be transformed in some way. The lines could be stored in a String or StringBuffer array. For **HL** candidates a similar Class could be used to edit batch command files, user registration data or setup files for a larger system.

©IB02.1.6 CONSTRUCT ALGORITHMS IN JAVA 2004

The following array has scores entered into it. The end of valid entries is marked with the special value -999. Where previous entries have been deleted the score has the special value -99.

Element 3 demonstrates this:

| | | | | | | |
|--------|----|----|-----|-----|-----|-------|
| scores | 23 | 19 | -99 | 9_9 | 4_3 | -99_9 |
| | 2 | 3 | | 4 | 5 | 6 |

To find the position where a new value should be inserted into this array the following informal algorithm is used:

```

Start at the first element
while an empty location is not found
    OR the end of the array is not reached
        move on to the next element

return the position of an empty location
    OR of the end of data marker (-999).

```

Construct the Java method returning an int value representing the insertion position.

Start with a correct signature:

```

public int findInsertionPoint()
{
    II finds the insertion point in an array scores
    II acceptable insertion positions have values -99 and -999
    II parameters were not specified so assume global variables

    pas = 0;    II set pointer to first element
    II loop through and search
    while ( (scores[ pas ] != -99) && (scores[ pas ] != -999) )

        pos = pos + 1;

    return pos;
}

```

EXERCISE 2.15

The above algorithm does not work if neither -99 nor -999 are encountered within the array. Add the statements necessary to detect this and return a suitable int value (or throw an exception if you prefer).



Consider the following arrays:

| | | | | | | |
|--------|-------|----|----|-----|-----|-----|
| scores | 1_9_9 | 19 | 56 | 9_9 | 4_3 | 8_2 |
| | 0 | 1 | 2 | 3 | 4 | 5 |

| | | | | | | |
|-------|--------|---------|--------|--------|--------|--------|
| names | phuong | richard | sayaka | jithan | aronal | andrew |
| O | I | Z | 3 | 4 | | 5 |

The following algorithm is designed to delete any name from the NAME array for which the corresponding entry in the SCORE scores is less than a given mark by replacing the name with the String "XXXX".

```
public void delete(int[] scores, int passMark, String[] names)
{
    for(int x = 0; x < LAST_P; x = x + 1)
    {
        if (scores[x] < passMark)
        {
            names[x] = DELETED;
        }
    }
}
```

The algorithm is called from the constructor as follows:

```
public PassMark()
{
    delete( scores, 50, names );
    showArrays();
```

Explain why the names array is changed in the constructor PassMark.

Trace the algorithm to show the contents of the NAMES array after it has completed execution.

©IBo2.1.7 EXPLAIN THE NEED FOR SEARCHING AND SORTING

Computer systems are constructed to solve data-processing problems (electricity billing, payroll processing, air traffic control to name but three). In such systems, updating of databases is an equally common operation. New items need to be added, old items deleted and existing items need to have details changed. Therefore searching for an item is a very common process.

Searching for an item is nearly always made easier when data items are held in some specific order and so methods of sorting data items have been extensively studied by programmers.

©IBo2.1.8 APPLY SPECIFIED SEARCHING AND SORTING ALGORITHMS

Students need to know linear and binary search algorithms and selection and bubble sort algorithms in some detail. They can be asked to compare them for efficiency and applicability and be able to say something about their space and time requirements as set out in the following two sections.

Students could be expected to recall the named algorithms from scratch - any suitable variation would be appropriate in this case.

Recursive versions of the binary search algorithm are not required in the common core.

Students may also be introduced, in the examination, to searches and sorts of similar complexity to those specified in the syllabus (eg the insertion sort method or its several variants could be described/developed but the quicksort would never be set).

SEQUENTIAL (LINEAR) SEARCH

This is as simple as the name suggests. In an array, or similar list of data items, each element is examined in turn to see if it is the required item. The search ends when the item is found or the end of the list is reached. The list need not be in any special order.

The following Class illustrates the linear search and the basic framework will be used for all searching and sorting examples:

```

public class SearchesSorts
{
    // basic data array
    private static final int MAX = 998;
    private static final int SIZE = 16;
    private int[] data = new int[ SIZE ];

    /**
     * Constructor
     */
    public SearchesSorts()
    {
        fillData(data);
        showData(data);
        int wanted = inputInt("Please input the item to search
for: ");
        linearSearch(data, wanted);
    }
    /**
     * This method fills the array data with random ints
     *
     * @param the array to be filled
     */
    void fillData(int data[])
    {
        // Puts random values from 1 to MAX into the array
        for (int i = 0; i < SIZE; i = i + 1)
        {
            data[i] = (int) (Math.random() * MAX) + 1;
        }
    }
    /**
     * This method shows the array contents on the screen
     *
     * @param the array to be shown
     */
    void showData(int data[])
    {
        StringBuffer line = new StringBuffer(SIZE*S);
        for (int i = 0; i < SIZE; i = i + 1)
    }
}

```

Computer Fundamentals

```
        line.append(data[ i] + " "I;  
    }  
    output (new String(line));  
}
```

The actual linear search algorithm:

```
1**  
* This method carries out a linear (sequential) search by  
* examining each item in the array in turn  
*  
* @param the array to be searched  
* @param the wanted item  
*1  
void linearSearch(int data[ ], int wanted)  
{  
    boolean found = false;      // flag to indicate end of search  
    int pos     0;              // current position in array  
  
    while ( (pos < SIZE) && !found )  
    {  
        found = (data[ pos] == wanted);  
        pos = pos + 1;  
    }  
    if (found)  
    {  
        // NB: gives position in the list rather than array  
        // element  
        // (which would be (pas-I))!  
  
        output("Found the item at position: " + pos);  
    }  
    else  
  
        output("the item" + wanted + " was not found.");
```

BINARY SEARCH

This method of searching a list is often compared to looking through a dictionary or telephone book. Suppose you are searching for the entry 'cat'. If you open the book near the middle you may find items starting with 'k'. Few people will then look further on in the book but will open the lower section somewhere near the middle, finding a letter such as 'f'. Again they open the book further towards the start and encounter 'b'. This process is followed until the item is located (or found not to be present). Formally expressed as an algorithm this might appear as follows:

```
while the list is bigger than 1 item and wanted item is not  
found  
  
    calculate mid point of list  
    if value at mid is wanted value then
```

```

    found it
else
    if wanted is in lower half
        move end point to one less than midpoint
    else
        move start point to one more than midpoint
endwhile

```

Follow the process on this list of numbers:

| O | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 13 | 15 | |
|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 5 | 7 | 8 | 12 | 24 | 25 | 34 | 42 | 43 | 56 | 67 | 68 | 69 | 74 | 80 | 95 |

Say the wanted item is 43.

First, find the mid point of the list (15+0) 7 (using integer division)

The value at 7 is 42; it is not the wanted item.

It is less than the wanted item so make position 8 the new start point of the list to be searched:

| 8 | 9 | 10 | 11 | 12 | 13 | 13 | IS |
|----|----|----|----|----|----|----|----|
| 43 | 56 | 67 | 68 | 69 | 74 | 80 | 95 |

The new mid point is I, where 68 is greater than the wanted item, discard the top elements:

| 8 | 9 | 10 |
|----|----|----|
| 43 | 56 | 67 |

The new mid point is 9, discard 9 and 10 and the only value left is 43. It is found in 4 iterations, the worst possible case in this particular list. If we had used linear search the worst possible case would be 16 - the length of the list, so that is quite a time saving.

BINARY SEARCH ALGORITHM

We use the same framework as above, only using a new method, **fillSorted()**, to make sure our random values are arranged in ascending order:

```

1**
* This method fills the array data with random ints
*
* @param the array to be filled
*1
void fillSorted(int data[] )

    // Puts ordered random values into the array
    data[0] = (int) (Math.random() * INTERVAL) + 1;
    for (int i = 1; i < SIZE; i = i + 1)

```

Computer Fundamentals

```
data[i] = data[i - 1] + (int) (Math.random() * INTERVAL) + 1;
```

One algorithm for binary search is:

```
/*
 * This method carries out a binary search by examining the
 * mid element in the list and then discarding half.
 *
 * @param the array to be searched
 * @param the wanted item
 */
void binarySearch(int data[], int wanted)
{
    boolean found = false; // flag to indicate end of search
    int start = 0; // first element of part to be searched
    int end = (SIZE - 1); // last element of part to be searched

    // loop until found or no more elements to search
    while ((end >= start) && !found)
    {
        // find mid point of current list
        int mid = (start + end) / 2;
        found = (data[mid] == wanted);
        if (found)
        {
            output("Found the item at element: " + mid);
        }
        else

            // discard half of the array and try again
            if (data[mid] < wanted)

                start = mid + 1;
            }
        else

            end = mid - 1;

    }
    if (!found)
    {
        output("the item" + wanted + " was not found.");
    }
}
```

For HL students this search can also be handled recursively (see Chapter 5).

SORTING

One disadvantage of binary search is that the list of values to be searched must be in order whereas linear search will work with un-ordered lists. There are very many different algorithms for sorting but the common core only requires you to be able to recall the simpler methods - the bubble and selection sorts.

BUBBLE SORT

This sort starts with the first pair of numbers (0, 1) which are compared and swapped if necessary. The next pair (1, 2) are compared until the last pair have been compared. Since the highest value in the list has now 'bubbled' to the top, the whole process is repeated with the first $n-1$ elements of the list. This continues until all items are in place.

First pass

| O | 2 | 3 | 4 | 5 | 6 | 7 | comment |
|----|----|----|----|----|----|----|---|
| 45 | 32 | 2 | 25 | 87 | 34 | 12 | 63 compare 45 & 32 |
| 32 | 45 | 2 | 25 | 87 | 34 | 12 | 63 swap them, 45 > 32, compare 45 & 2 |
| 32 | 2 | 45 | 25 | 87 | 34 | 12 | 63 compare 45 and 25 and swap them |
| 32 | 2 | 25 | 45 | 87 | 34 | 12 | 63 compare 45 and 87, no swap 45 < 87 |
| 32 | 2 | 25 | 45 | 87 | 34 | 12 | 63 compare 87 and 34... |
| 32 | 2 | 25 | 45 | 34 | 87 | 12 | 63 |
| 32 | 2 | 25 | 45 | 34 | 12 | 87 | 63 |
| 32 | 2 | 25 | 45 | 34 | 12 | 63 | 87 Highest value now at top. Sort 1 to 7 next |

A variation is to check if any swaps were necessary on a complete pass of the unsorted portion of the list, if no swaps were needed, the list must have been sorted and the sort can terminate immediately.

This is a very compact bubble sort algorithm:

```
/**
```

```

* This method sorts the array data
*
* @param the array to be filled
*1
private void bubbleSort(int[] data)
{
    output("sorting... ");

    II top is the boundary between the sorted and unsorted portion
    for (int top = (SIZE - 1); top > 0; top = top - 1)
    {
        for (int upper = 1; upper <= top; upper = upper + 1)
        {
            II upper and lower are the neighbours being compared
            int lower = upper - 1;
            if (data[upper] < data[lower])
            {
                I swap
                int temp = data[upper];
                data[upper] = data[lower];
                data[lower] = temp;
            }
        }
    }
}

```

EXERCISE Z.19

- a) Re-write the algorithm to use two nested while loops.
- b) Add a terminating condition to the inner loop to test if any swaps were needed. It might look a little bit like this:

```

while (TOP > 1) and SWAPPED do
    SWAPPED <- false      II flag to check if swap took place
    II start at first 2 values in unsorted part
    while UPPER <= TOP   II check if at top of array yet
        if DATA[UPPER] < DATA[LOWER] then
            II swap the values
            II set the swapped flag
        endif
        II move up the array and compare
        II the next pair of values
    enddo
enddo

```



SELECTION SORT

The term selection sort has been used by different authors to describe different sorting methods. However, in the old IB teacher support material a version with the following informal algorithm is given:

'Sort in descending order by finding the largest element in the array and then swapping it into position 1, then finding the next largest and swapping it into position 2 and so on.'

There does not seem to be any reason for this to change in the new subject guide and support material. Of course the array can equally well be sorted into ascending order as described below.

- The list is split into two parts, the sorted part and the unsorted part (initially all of the elements).
- Each element of the unsorted part is examined in turn to locate the smallest and this is swapped (exchanged) with the first element of the unsorted string.
- The boundary of the sorted string is then incremented.
- The process continues until all elements have been placed in the correct position.

| | | | | | | | | |
|----|----|---|----|----|----|----|----|--|
| 45 | 32 | 2 | 25 | 87 | 34 | 12 | 63 | scan to find 2 is the smallest, then swap 45 & 2 |
|----|----|---|----|----|----|----|----|--|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|---|
| 2 | 32 | 45 | 25 | 87 | 34 | 12 | 63 | scan unsorted to find next smallest, 12, then swap 32 & 12 |
|---|----|----|----|----|----|----|----|---|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--------------|
| 2 | 12 | 45 | 25 | 87 | 34 | 32 | 63 | swap 45 & 25 |
|---|----|----|----|----|----|----|----|--------------|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--------------|
| 2 | 12 | 25 | 45 | 87 | 34 | 32 | 63 | and so on... |
|---|----|----|----|----|----|----|----|--------------|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--|
| 2 | 12 | 25 | 32 | 87 | 34 | 45 | 63 | |
|---|----|----|----|----|----|----|----|--|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--|
| 2 | 12 | 25 | 32 | 34 | 87 | 45 | 63 | |
|---|----|----|----|----|----|----|----|--|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--|
| 2 | 12 | 25 | 32 | 34 | 45 | 87 | 63 | |
|---|----|----|----|----|----|----|----|--|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--|
| 2 | 12 | 25 | 32 | 34 | 45 | 63 | 87 | |
|---|----|----|----|----|----|----|----|--|

This sort is sometimes known as the Exchange or Interchange sort. An algorithm for this sort is given below:

```
/**
 * This method sorts the array data
 *
 * @param the array to be sorted
 */
private void selectionSort(int[] data)
{
```

```

        output("selection sorting... ");
    II pos is the boundary between sorted and unsorted parts
    II of the array
    for (int pos = 0; pos < SIZE; pos = pos + 1)
    {
        II find the position of the smallest value in the unsorted
        part
        II then swap it with the value at the current position
        int smallPos = minValueAt(data, pos, SIZE - 1);
        int temp = data[smallPos];
        data[smallPos] = data[pos];
        data[pos] = temp;
    }
}

```

For clarity a 'helper method' has been used to locate the position of the smallest value in the unsorted part of the array:

```

1**
 * This method finds the position of the smallest value
 * in the array data between start and end
 *
 * @param the array to be examined
 * @param the start element
 * @param the end element
 */
private int minValueAt(int[] data, int start, int end)
{
    int minSoFar = data[start]      II minimum value found so far
    int minPos = start;           II position of minSoFar in array

    for(int pos = start + 1; pos <= end; pos = pos + 1)
    {
        if (data[pos] < minSoFar)
        {
            II found a new minimum
            minSoFar = data[pos];
            minPos = pos;
        }
    }

    return minPos;
}


```

~~Q1B0:Z1.9~~ **COMPARE THE EFFICIENCY OF SEARCHING AND SORTING ALGORITHMS**

The efficiency of algorithms is discussed in terms of their execution time and their space requirements. Some comparisons using the searching and sorting algorithms already presented are given in the following section.

© IBO **2.1.10 DISCUSS THE EFFICIENCY OF SEARCHING AND SORTING ALGORITHMS**
2004

The efficiency of a linear or sequential search algorithm depends on the number of items in the array, in the worst case the entire array has to be searched. Therefore there is a linear relationship between the array size and the search time.

For binary search, the array is halved in size at each search step, doubling the size of the array leads to a single step increase in search time. We can summarize this information as follows:

| Array Size | Linear search steps | Binary search steps |
|------------|---------------------|---------------------|
| 2 | 2 | 1 |
| 4 | 4 | 2 |
| 8 | 8 | 3 |
| 16 | 16 | 4 |
| 32 | 32 | 5 |
| 64 | 64 | 6 |
| 128 | 128 | 7 |

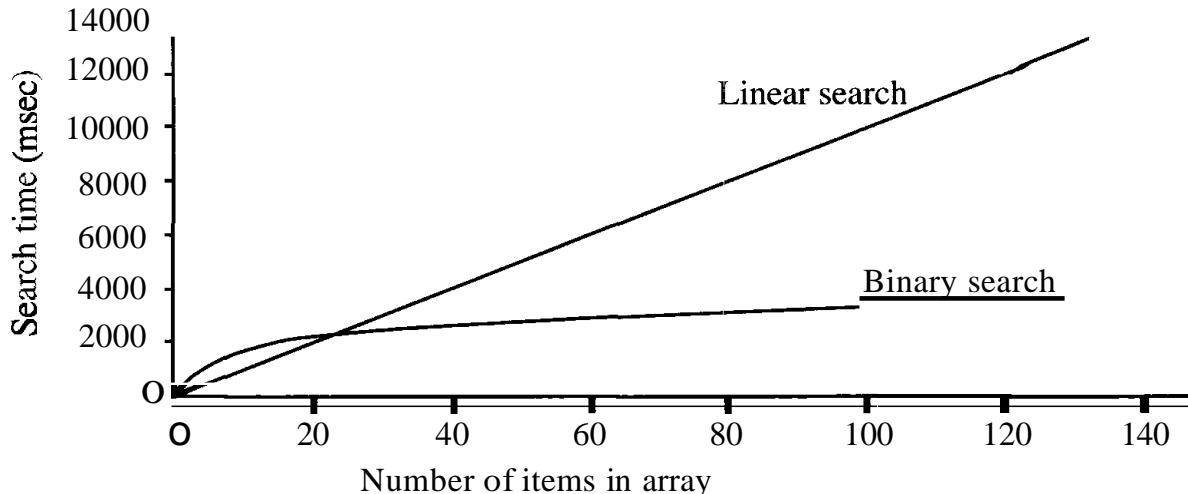
Each binary search step takes a bit longer than each linear search step because the algorithm is more complex. If we allocate (completely arbitrarily) a time of 100 mSec to each linear search step for a particular machine and 500 mSec to each binary search step we can construct a graph of array size against time for a hypothetical (and very slow!) computer:

| Array Size | Linear search time | Binary search time |
|------------|--------------------|--------------------|
| 2 | 200 | 500 |
| 4 | 400 | 1000 |
| 8 | 800 | 1500 |
| 16 | 1600 | 2000 |
| 32 | 3200 | 2500 |
| 64 | 6400 | 3000 |
| 128 | 12800 | 3500 |

Using a spreadsheet to plot this result shows that the binary search has an inherent advantage over the linear search, especially as N gets large. The linear search is faster for small values of N.

For **HL** students, algorithm efficiency can be described using big O notation (Chapter 5) but **SL** students need only a general awareness of the differences in efficiency of these two algorithms.

Figure: 1.1 Comparisou ofLinear and Binary search



In general, with n items in the list, n is the worst-case number of iterations for linear search whereas for binary search it is $\log_2 n$.

To summarize the characteristics of these two search methods:

| Linear Search | Binary Search |
|--|--|
| Is relatively inefficient since, in the worst case, all items must be searched. | In the worst case only $\log_2 n$ items must be examined. |
| Works equally well with sorted or unsorted lists. | Only works with a sorted list. |
| If the list is small, it may be faster than binary search because the algorithm is a simple one. | May not be suitable for small lists since the algorithm is more complex than that for linear search. |

The efficiency of sorting algorithms may depend upon how the original list is ordered. Usually the worst case is a reversely ordered list. In both the bubble and selection algorithms there are two nested loops. One runs from 0 to $N-1$ and the inner loop from the current position to $N-1$. A gross simplification might represent this process as:

```

for X = 0 to N-1 do      // for the whole array
    for Y = X to N-1 do  // sort next unsorted item
        some steps          // using sort specific method
    endfor
    some other steps       // depends on sort method
endfor

```

In the worst case, the loop within a loop structure will cause the sort to take a time that is proportional to N^2 .

The best case would be when the list is already sorted. In this case the inner loop (some steps) would still execute unless there is some condition which will stop it. In the case of the bubble sort, the swap flag can cause this loop to terminate early and increase the efficiency of execution so that only one pass is made through the list. This would make the efficiency of a bubble sort proportional to N instead of N^2 for this particular case. This short circuit mechanism does not work for the selection sort.

EXERCISE 2.20

Consider a two-dimensional array (matrix) of dimensions c and r . The matrix must be searched for a given value and is sorted within each row, but not necessarily by row. Each value in a matrix cell is unique.

A picture makes things clearer (as you should remember when discussing data structures in your dossier):

| | O | 1 | 2 | \dots | c |
|---|---------|---------|---------|---------|----|
| O | 1 | 7 | 11 | \dots | 23 |
| | 4 | 5 | 9 | \dots | 77 |
| 2 | 3 | 6 | 15 | \dots | 46 |
| | \dots | \dots | \dots | \dots | |
| r | 17 | 45 | 65 | | 79 |

- a) Compare two different strategies you could use to implement this search algorithm.
- b) As a class activity, or for the first to finish O), compare and, if possible, classify all the different strategies that have been tried by different members of the class. Discuss the efficiency in terms of r and c .
- c) Do the relative dimensions of r and c have any effect on the efficiency of your chosen solutions? Hint: consider extreme cases like 100×2 and 2×100 matrices.
- d) For the adventurous; design and implement an algorithm that will sort this matrix into order by row and column.



FURTHER EXAMPLE ALGORITHMS AND ACTIVITIES

An informal description of the insertion sort is:

- The list is split into two parts, the sorted part (initially the first element) and the unsorted part.
- Each element of the unsorted part is examined in turn and placed in the correct position in the sorted string, which then increases by one.
- Any elements greater than the inserted element have to be shuffled up one place in the sorted string.
- The process continues until all elements have been examined.

Computer Fundamentals

Many people use a similar technique to arrange playing cards in their hands. Here is the process with some example data:

| | | | | | | | | |
|----|----|---|----|----|----|----|----|--------------------------------------|
| 45 | 32 | 2 | 25 | 87 | 34 | 12 | 63 | 1 sorted, take first of unsorted, 32 |
|----|----|---|----|----|----|----|----|--------------------------------------|

| | | | | | | | | |
|----|----|---|----|----|----|----|----|-------------------------------------|
| 32 | 45 | 2 | 25 | 87 | 34 | 12 | 63 | shuffle 45 to make space, insert 32 |
|----|----|---|----|----|----|----|----|-------------------------------------|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|-----------------------------------|
| 2 | 32 | 45 | 25 | 87 | 34 | 12 | 63 | shuffle 32,45 to make space for 2 |
|---|----|----|----|----|----|----|----|-----------------------------------|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|-------------------------------|
| 2 | 25 | 32 | 45 | 87 | 34 | 12 | 63 | 25 gets inserted in its place |
|---|----|----|----|----|----|----|----|-------------------------------|

(2 doesn't have to move)

| | | | | | | | | |
|---|----|----|----|----|----|----|----|------------------|
| 2 | 25 | 32 | 45 | 87 | 34 | 12 | 63 | 87 is already OK |
|---|----|----|----|----|----|----|----|------------------|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|-------------------------------------|
| 2 | 25 | 32 | 34 | 45 | 87 | 12 | 63 | shuffle 87, 45 to make space for 32 |
|---|----|----|----|----|----|----|----|-------------------------------------|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--|
| 2 | 12 | 25 | 32 | 34 | 45 | 87 | 63 | |
|---|----|----|----|----|----|----|----|--|

| | | | | | | | | |
|---|----|----|----|----|----|----|----|--|
| 2 | 12 | 25 | 32 | 34 | 45 | 63 | 87 | |
|---|----|----|----|----|----|----|----|--|

Here is an algorithm for the insertion sort:

```

1**
 * This method sorts the array data
 *
 * @param the array to be sorted
 */
private void insertionSort(int[] data)
{
    output("insertion sorting...");
    int cp = 0; // current position - value in unsorted list
                // being examined
    int pt;     // pointer to number in sorted part
                // being compared with cp
    int tm;     // temporary store for number to be inserted

```

```

while (cp != (SIZE - 1))  Iwhile not at end of array
{
    II set up pointers
    cp = cp + 1;  II increment current position to first in
                   unsorted
    pt = cp;       II pointer into sorted part - starts at top
    tm = data[cp]; // temp store for element to insert

    II while not at start, and next value is still too big,
    II shuffle up current element in sorted part by 1
    while ( (pt > 0) && (data[pt-1] > tm) )
    {
        data[pt] = data[pt - 1];
        pt = pt - 1;
    }
    II insert the temp value into the sorted part
    data[pt] = tm;
}

```

Copy and complete the following trace table:

| DATA | | | | | | | | | | | | |
|------|----|----|--------------|------|----------------|----|---|----|----|----|---|--|
| cp | pt | tm | cp! = SIZE-1 | pt>0 | data[pt-1] >tm | 0 | 1 | 2 | 3 | 4 | 5 | |
| 0 | | | TRUE | | | 23 | 7 | 34 | 36 | 13 | 9 | |
| 1 | 1 | 7 | | | TRUE | | | | | | | |

Compare the efficiency of the insertion sort using these two data sets:

set 1

| 0 | 2 | 3 | 4 | 5 |
|---|---|----|---|----|
| 2 | 7 | 14 | 6 | 18 |

set 2

| 0 | | | | | |
|----|----|-------------|---|----|---|
| 23 | 17 | <u>1=14</u> | 9 | 13 | 8 |

CLASS ACTIVITY

Search the Internet for sites which have interactive: sorting demonstrations.



EXERCISE 2.21

A two dimensional array of objects is used to hold birthday information on friends and relatives. The Class BirthdayData has the following structure:

```
/*
 * A Class to keep details of birthday data
 * @author Arnold J Rimmer
 * @version 25123000000
 */
public class BirthdayData
{
    private String name;
    private char sex;
    private boolean getsPresent;

    /**
     * No argument Constructor
     */
    public BirthdayData()
    {
        // initialise instance variables
        name = null;
        sex = ' ';
        getsPresent = false;
    }
    /**
     * Constructor
     */
    public BirthdayData(String name, char sex, boolean
getsPresent)
    {
        setName(name);
        setSex(sex);
        setGetsPresent(getsPresent);

        /*
         * you may assume the usual accessor and mutator methods
         * are present, along with IBID methods
         */
    }
}
```

In another Class, an array of these objects is declared:

```
BirthdayData[] birthdays = new BirthdayData[ 12 ] [ 31 ]
```

| | 0 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|----|----|
| 0 | | | | | | | | | | | |
| 2 | | | | | | | | | | | |

| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | | | |
| | | | | | | | | | | | |

| | | | | | | | | | | | |
|----|--|--|--|--|---|--|--|--|--|--|--|
| 29 | | | | | X | | | | | | |
| 30 | | | | | | | | | | | |

The entry marked with an X represents May 30th and the data in the object instance might be:

| | |
|--------------------|-------------------------------|
| X.getName() | returns "30th Avril Ie Jeune" |
| X.getSex() | returns 'F' |
| X.getGetsPresent() | returns true |

- a) State the number of elements contained in this array.
 - b) State the initial values in each array element.
 - c) Outline one possible problem with this data structure.
 - d) If each name is allocated 25 bytes, character and Boolean types each occupy 1 byte calculate the total memory needed to store the array data.
 - e) How many array elements will never contain any data (hint: assume a year has a maximum of 366 days).
 - f) In most cases only a few of the total available cells will contain any data. Outline a more efficient way to store this information.
 - g) Construct the algorithm which accepts a day and month number and returns a string containing the persons name, sex and whether they should receive a present, e.g.:
- The call getDetails(30, 5) returns the string "Avril Ie Jeune, female, yes"
 - The call getDetails(13, 4) returns the string "No birthday data"
- (Remember that array element numbers do not correspond exactly to day and month numbers!)

© IBO 2004 **2.1.11 PROGRAMMING ERRORS**

Errors can occur with hardware and software as well as the data entry errors described in Chapter 3. Computers are sensitive to electrical and mechanical fields which can cause hardware to stop working properly and also alter the contents of magnetic media. Software errors are usually put into one of three categories:

Logic errors: The coding of the program has been incorrect in sequencing or choice of conditions, such as:

```
// algorithm to add up 9 numbers
// example of logic error - contains 2 logical errors
int count = 0;
int numbers = 9;
int sum;

while (count <= numbers)

    sum = 0;
    int number = inputInt("Next number: ");
    sum = sum + number;
    count = count + 1;
}
output("The sum is: " + sum);
```

Runtime errors: There are many things that can go wrong as a program runs, some examples are:

- division by zero
- truncation, underflow and overflow errors
- file not found
- printer not ready
- illegal memory access, etc.

When using double numbers as we described above in section 2.1.1, there is an area where very small numbers cannot be represented if they are too close to zero. The exact way in this works is of interest only to HL students. However, if a number close to the lower bound is divided by a number greater than 1, then a condition known as underflow can occur. Similarly, if the result of a calculation exceeds the largest number that can be stored, overflow can occur. These events will also generate runtime errors in many computer languages.

Syntax errors: These are errors in the syntax of the program language - such as mis-spelling a keyword. These are caught at the compilation stage or while a program is being interpreted.

```
integer sum;

while (true)
    sum = sum + 5
```

Would generate two syntax errors.

The process of testing a program involves both functional testing and testing with different types of input data and is discussed in chapter 1.

3

Chapter contents

- 3.1 Language Translators
- 3.2 Computer architecture
- 3.3 Computer Systems
- 3.4 Networked Computer Systems
- 3.5 Data representation
- 3.6 Errors
- 3.7 Utility Software

INTRODUCTION

This chapter covers the hardware and software related to computer systems and how they interact. It directly covers the theory specified in topic 3 of the common core of the IB computer science syllabus. There are a number of exercises and activities included at appropriate places within the text. Many of the exercises include questions that are of the type used in the exam and therefore use the keywords related to the objectives. It is important that students ensure that they are aware of the significance of the objective number that appears beside each assessment statement (AS) in the syllabus. For example, objective I means that students will be asked to define, draw or state something specific about the assessment item. Students are referred to the syllabus outline for the full list of objective definitions.

This chapter forms the basis of the theory of the standard level course and higher level course. Students will have varying levels of background and some may require further knowledge. For instance, a student may require additional background in understanding I/O devices than is provided in this text. Whilst there are many texts that students can refer to for additional detail or background the Internet provides a range of freely available sites that enable students to get access to definitions and further technical material that is relevant to the course and in some cases their extended essay.

Some of these are listed below.

Useful resource sites

- <http://webopedia.internet.com> (very easy to use and has good links)
- <http://www.ipl.org/ref/RR/static/comOO.OO.OO.html> (complex but fun!)
- <http://www.howstuffworks.com> (follow the computers link)
- <http://www.whatis.com> (simple and easy to use)
- <http://www.compinfo-center.com/tpdict-t.htm> (complex but informative)

© IBO 2004 **3.1 LANGUAGE TRANSLATORS**

© IBO 2004 **3.1.1 DEFINE SYNTAX AND SEMANTICS**

The term **syntax** refers to the rules that govern how statements in a computer programming language must be constructed. Common syntax errors include:

- incorrectly spelling a keyword
eg `clas` for `class` or `inp` for `int`
- using the wrong brackets
eg `System.out.println["this is the wrong bracket"]`
- leaving off the matching bracket
eg `if((x==y) && (p == q)`
- leaving off the end statement symbol, which in Java is the semi-colon

The term semantics refers to the meaning conveyed by a collection of statements.

It is possible to write a syntactically correct statement that conforms to the rules of grammar but which does not make sense in the context in which the statement is used. In other words the meaning covered by the statement is incorrect or it has no meaning. For example, using an input statement when an output statement is required.

A computer can detect syntax errors e.g. a software compiler can check for syntax errors and a spell checker in a word processing package can check for spelling errors and simple mistakes in grammar. However, computers cannot yet determine the purpose of computer programs or the semantic meaning of the contents of a word processed document.

©IB031_2 DESCRIBE THE FUNCTION OF HIGH-LEVEL LANGUAGE TRANSLATORS

What are high level languages?

The hardware cannot do a lot without the software, the programs and data that it operates on. There are two main types of computer language for writing computer programs:

- Low-level language
- High-level language

The hardware runs only in native machine code, usually represented by the 1's and 0's of binary code. The electrical circuits and magnetic storage devices that make up the computer can only recognise these two states. Inside the hardware's random access memory a computer program is represented in binary form as machine code instructions and if we were able to look inside the memory the instructions would look like this:

```
0001010011100110
0001001100001111
1010001110101100
1001010000111011
```

This is tedious for humans to work with and they tend to make lots of mistakes when trying to do so. The first step for computer engineers was to make codes which could stand for binary instructions. Thus ADD might correspond to the code for adding two numbers together, LDA might correspond to the code for fetching a number from memory (LoADING the Accumulator). Thus a low level language might express the above as:

```
LDA 2309
ADD 2310
MLT #002
STO 2325
```

This language is also known as Assembly Language. Each line of an assembler program corresponds to a single line of machine code, thus there is a one-to-one correspondence between assembler code instructions and their equivalent machine code instructions.

Of course, this is still not very easy to handle so the next step was the development of High Level Languages (HLL's). One HLL statement usually translates into many machine code statements.

Computer Fundamentals

An example of a HLL is Java and the following code segment shows a program to display the 2 times table up to 12.

```
int t = 2;  
int j;  
for (j = 1; j < 13; j++)  
  
    System.out.println(" " + j + " x " + t + " " + j * t);
```

Which, believe it or not, is easier to understand when you are trying to develop computer software (programs) rather than developing the software in either assembler or machine code.

Some features of HLL's are:

- They are portable, i.e. they can be run on different machines rather than being machine-specific like assembly language.
- They are English-like and more easy to understand.

Different languages have been developed for different tasks.

EXAMPLES OF HIGH LEVEL LANGUAGES

There are said to be 2000 different high level computer languages, so these are just a few examples from the mainstream:

Early HLL's were FORTRAN (FORmula TRAnslation language) which was intended for mathematical and scientific programming and COBOL (COmmon Business-Oriented Language) for developing business and data-processing applications.

ALGOL (ALGOrithmic Language) was an early (1950's) attempt to introduce the concept of structured programming and eventually led to the development of C and so on to Pascal which are highly-structured block languages. The structure is (supposed to be) used by programmers to make the workings of the program clear to other programmers who may have to work in the same team or perform software maintenance.

BASIC (Beginners All-purpose Symbolic Instruction Code) was introduced for microcomputers and, although poorly structured and often misused, gave many, many people an easy introduction to computer programming.

Some other notable languages are: FORTH, LISP, Scheme, Haskell and Prolog. Some of these languages are not based around procedural statements like those we are familiar with, but logical assertions or functional elements. All of these languages are available (or have been at some time) in versions for microcomputers and there are many free resources, if you wish to experiment.

More recently Object Orientated Languages, notably SMALLTALK, C++ and Java have taken the concepts of structured programming one stage further. Only Higher Level students need to know the concepts associated with Object Oriented programming and these are covered in Chapter 5.

COMPARISON OF HIGH LEVEL LANGUAGES AND LOW LEVEL LANGUAGES

The table below provides a brief comparison between high and low level languages. It should be noted that program developed in a HLL are converted to LLL equivalents prior to execution. We consider this process in the next section.

| | High Level | Low Level |
|---|---|---|
| 1 | One instruction = many machine code instructions. | One instruction = one machine code instruction. |
| 2 | Portable, task-oriented | Machine specific, machine-oriented |
| 3 | More English-like | Less easy to write and debug. |

WHAT IS A COMPILER?

A compiler is a translation program that converts source code into an equivalent object code format. This is the process of converting a program written in a HLL to an equivalent LLL format so that it can be executed. Compiled object code can then be executed or run by the computer. The source code is created by the programmer using either a simple text editor or an IDE (Integrated Development Environment).

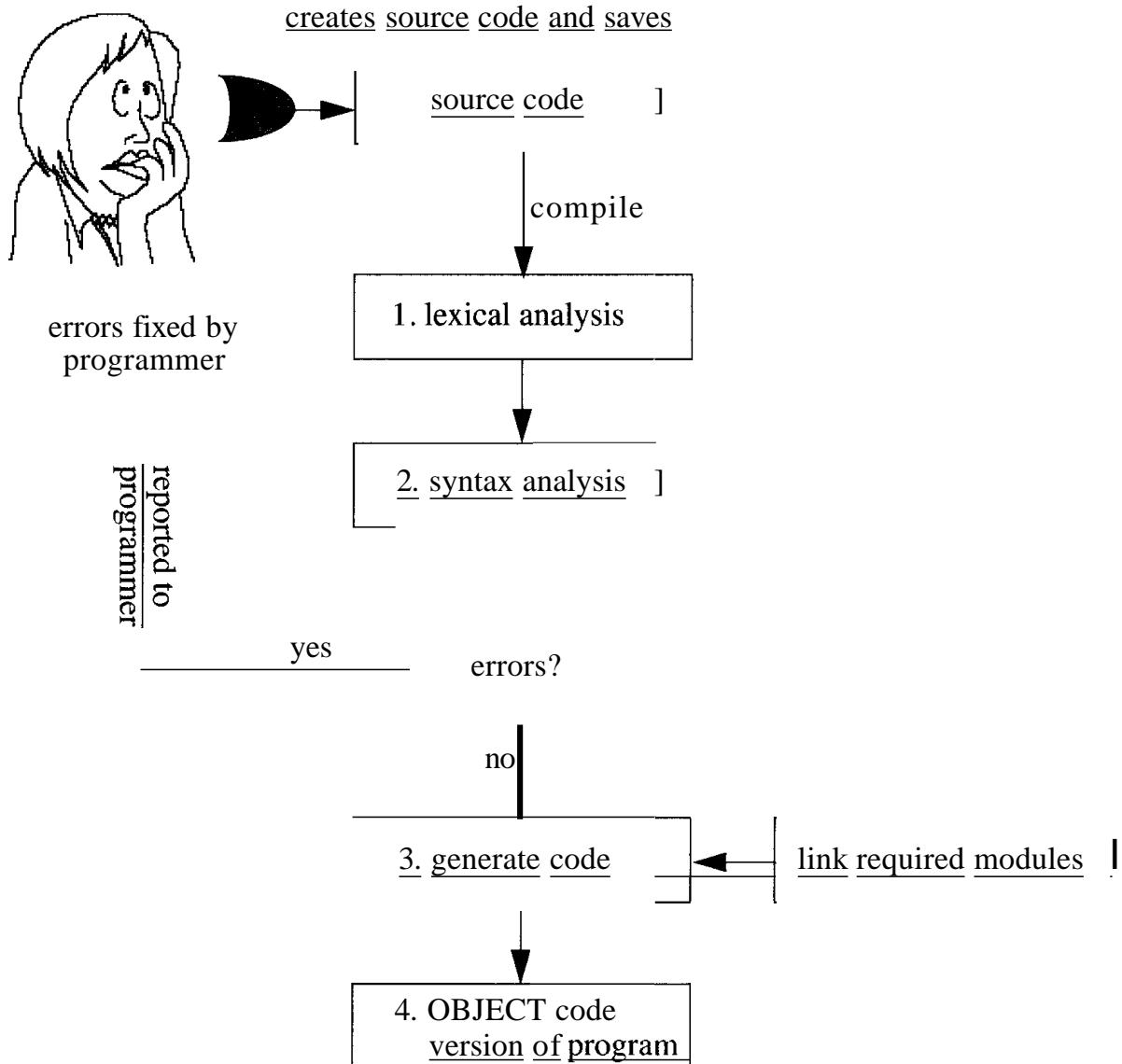
The process of compilation is a stepwise process.

1. The first step is termed lexical analysis. In this step the compiler removes all the spaces and comments and looks for reserved words used by the language such as for, while etc. which may be replaced by single-character 'tokens'. The compiler produces a new version of the code that is then passed to the next step.
2. The second step is termed syntax analysis. In this step the program is checked against the syntax rules of the language e.g. that brackets match, do is matched with a corresponding while statement etc. Syntax errors found at this stage are reported to the programmer.
3. The third step is termed code generation. In this step the actual code that will be run by the computer is created. Normally this is machine code. Variables are given memory locations and are added to a symbol table. Any standard library functions such as mathematical functions or special functions to perform the input and output are linked into the final compiled program.

The above steps are represented in figure 3.1 below:

Figure: 3.1

Programmer



In summary, the function of the compiler is to check for syntax errors and report any that are found back to the programmer to correct. The programmer then corrects the source code and repeats the compilation stage. If the source code is found to be syntactically correct, the compiler links in externally required modules and generates the required object code, which can then be loaded and executed. Otherwise the programmer must again attempt to correct the syntax errors and then attempt compilation.

Compilers produce source code that makes up a separate program. For example, `program.cpp` could be the name of a C++ source code and `program.exe` could be the name given to the compiled object code that can be run directly.

To compile a program, the compiler needs to be loaded into memory and it can take some time to compile long programs. Compiled code usually executes faster than an equivalent interpreted version of the same program. Compiled code can often be moved across a range of compatible architectures without the need to re-compile. However, source code produced on one type of computer may not run on another computer, especially if it has to operate with a different or incompatible operating system. In this case changes may need to be made to the source code before re-compilation is undertaken, often with a different version of the compiler.

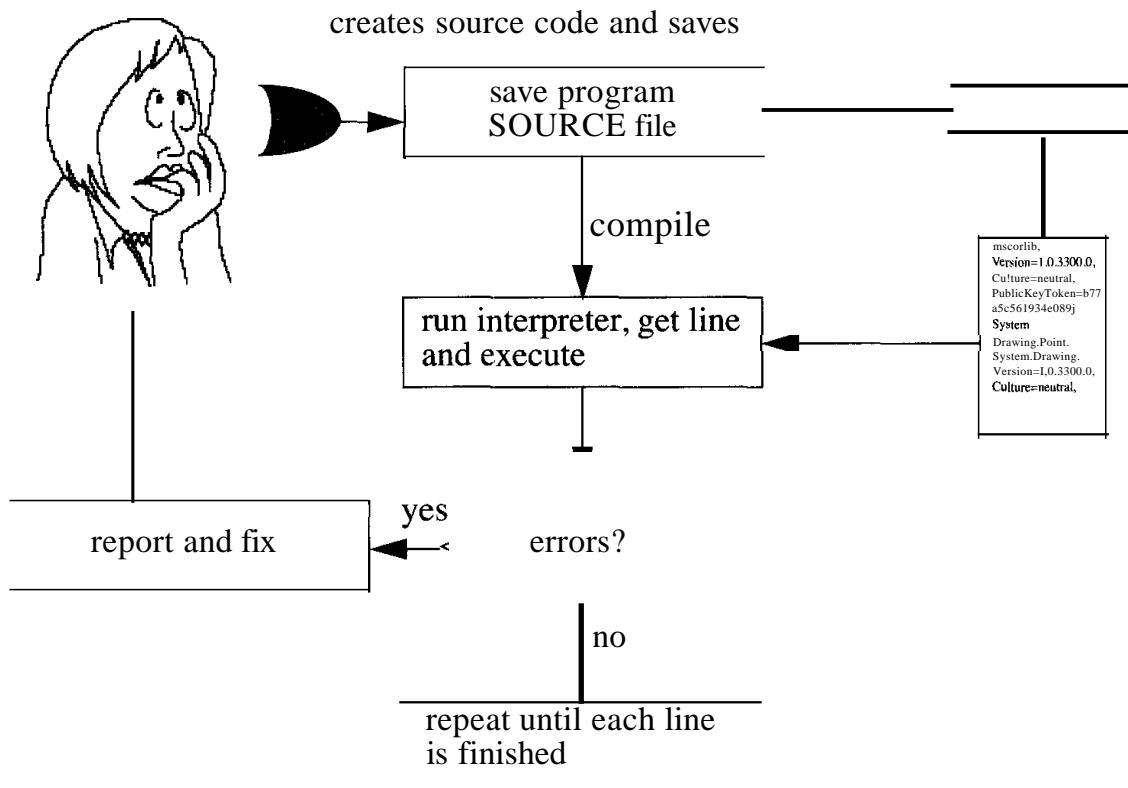
We will consider the process of using a translator called an interpreter in the next section. However, it is important to note at this point that a compiler differs from an interpreter in a number of ways. An interpreter analyzes and executes each line of source code in succession without firstly looking at the entire program. A compiler creates a completely new program that is executed. A compiler does not stop at the first error and continues the process by reporting the syntax errors found. It is not necessary to have the compiler loaded into memory to execute a compiled program.

WHAT IS AN INTERPRETER

An interpreter translates high-level instructions into a format that can be directly executed one line at a time. This is in contrast to a compiler which creates a completely separate program. Compiled programs generally run faster than interpreted programs. The key advantage of an interpreter is that the line of code being interpreted is executed straight away. This can be an advantage when developing large computer programs.

Unlike compiled programs the interpreter needs to be loaded each time the program runs and thus takes up memory space.

Figure: 3.2



Software development requires that the programmer has access to an editor to create the source code. It is also likely that the programmer has access to a debugging tool or debugger, which allows the programmer to perform a range of debugging functions in order to trace and remove bugs from the source code. Often the programmer has access to a software development environment that integrates the editor and the debugger into what is known as an integrated development environment (IDE).

A final point. Code generators are often associated with IDEs. For example, in GUI environments a design tool is likely to be available that allows the programmer to construct the GUI interface instead of writing the code directly to do this. At the end of the GUI development the code required to create the screen is generated by the IDE and the programmer can then view this code as source code.

In summary, the role of the translation process is to convert source code into object code that can be executed. Compiled code or object code is a completely separate program which can stand alone and execute without the compiler. Such object code can't be modified, whereas source code that runs via an interpreter can normally be viewed.

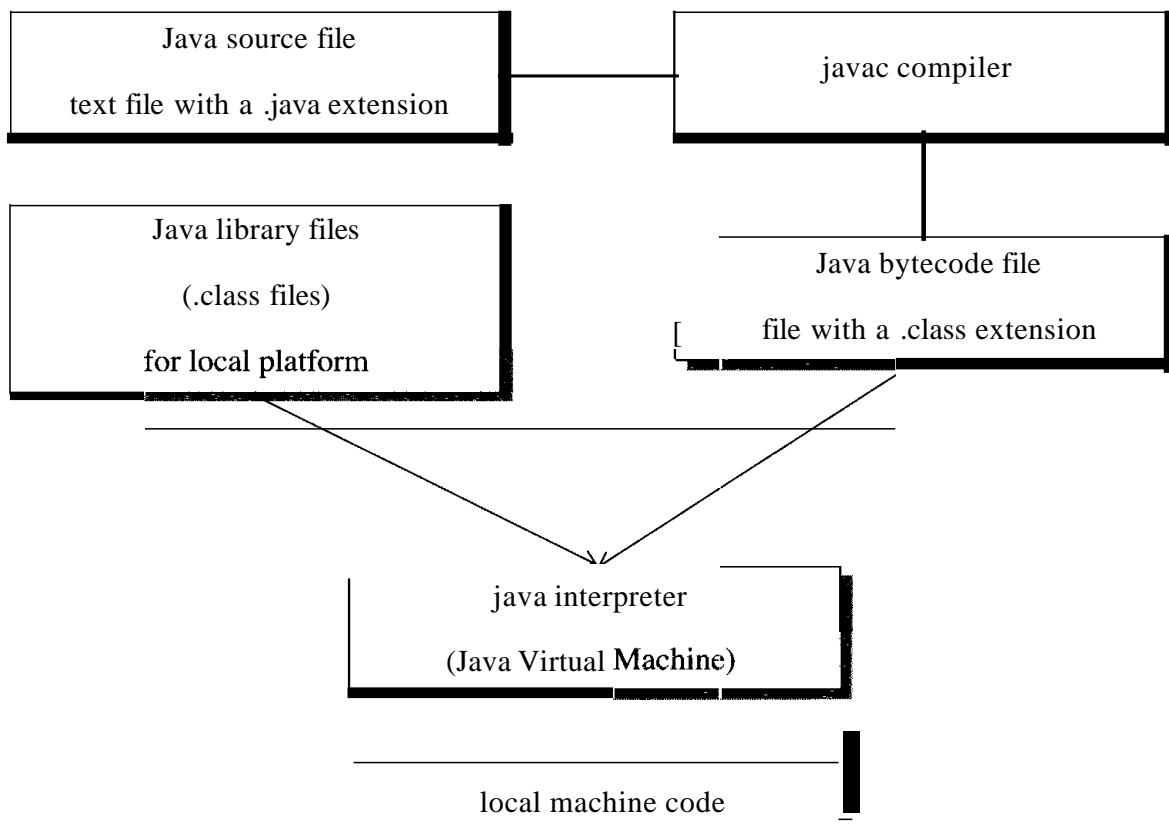
COMPILING AND RUNNING A JAVA PROGRAM

The case of Java is somewhat different as it was originally designed as a cross-platform language. Because of this, a separate compiler and library of modules would be needed everywhere Java source code is to be turned into machine language.

The designers of Java decided to have Java 'compiled' not to anyone native machine code for any particular hardware system but rather to compile to an intermediate stage (known as Java bytecodes). This stage is what is found in a java .class file produced by running the javac program on a Java source code file.

The bytecodes are a tokenised and otherwise compressed version of the original code which are then passed to an interpreter (the Java Virtual Machine) which has to be written for each platform Java is to run on. It is the JVM which actually produces the machine code from the bytecode file and from any pre-compiled library units needed by the local operating system.

Figure: 3.3



EXERCISE 3.1

1. Clearly define the terms 'syntax' and 'semantics'.
2. Compare High Level languages with Low Level languages.
3. Outline, giving examples, why there are many different types of High Level language.
4. Outline the differences between source code and object code.
5. If you were a program developer or software supplier, what advantage is there in distributing your software as object code instead of source code?
6. Describe the operation of a compiler.
7. Outline how an interpreter functions.
8. What is the role of a debugger?
9. Using your chosen program development environment, locate the text editor and debugging tools; is it an IDE? Does it provide any code generation features?



© IBO 2004 **3.1.3 SOFTWARE DEVELOPMENT TOOLS**

There are a range of software development tools that make the development of certain software easier. Some of these are briefly listed below.

HTML editor

A software tool that enables web pages to be created is termed an HTML editor. It enables HTML code to be created by using icons or menu options to generate the program. A good example is the basic editor included in the Netscape browser. By using such a tool, the developer does not need to know or remember the syntax of a range of HTML instructions and can concentrate on what they wish the HTML to do. Use of such tools should mean faster development.

DBMS

Date Base Management Systems (DBMS) enable databases to be created without the need to write the specific computer code. DBMS have a range of software modules that enable data dictionaries and files to be created, they provide a structured query language to enable the data to be queried, facilities to update the data and manage the data and a reporting facility to produce desired printed reports.

CASE Tools

Computer Aided Software Engineering (CASE tools). CASE tools are a category of software tools that provide a development environment for programming teams. They enable the stages of the systems development cycle to be implemented and managed in an integrated manner. CASE systems offer tools to automate, manage and simplify the development process. These can include tools for:

- Summarizing initial requirements.
- Developing flow diagrams.
- Scheduling development tasks.
- Preparing documentation.
- Controlling software versions.
- Developing program code.

Macros

When you use a product such as word processor you often need to be able to group a set of instructions into a single operation. Most of the standard office products provide a way to do this by enabling the creation of what are termed 'macros'. Macros are normally developed by recording a series of keystrokes which can be referenced later, so they can be played back. The referencing maybe provided by using some form of special icon or set of keystrokes, often termed a 'shortcut key'. The advantage of macros is that they save time and effort by automating tedious and repetitive processes. They can also be shared between users.

EXERCISE 3.2

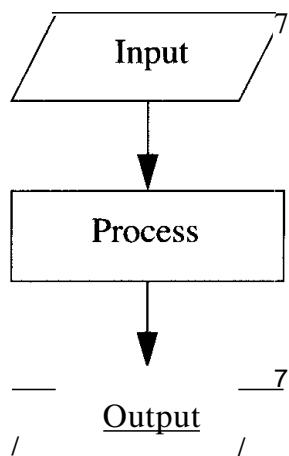
1. Define the term macro and outline a situation in which a macro could be used in a general purpose applications package.
2. An insurance company needs to keep track of personal and pay records as well as details of clients and their insurance contracts. Explain why a DBMS (Database Management System) is used in preference to several stand alone applications.
3. A group of musicians want to make their music available on the internet. Explain why they would use HTML and outline two reasons why the site would have different types of file (e.g. music and graphics).



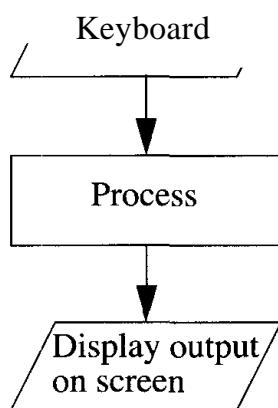
© IBO 2004 3.2 COMPUTER ARCHITECTURE

The term computer architecture refers to the logical layout and relationships between the different components of a computer system. A computer system is fundamentally an Input -> process -> output system. Thus, the fundamental architecture of a computer system shows the relationship between the input components, the processing components and the output components of the system. The simple block diagram shown in figure 3.4 outlines the relationship between the components of a computer.

Figure: 3.4



Thus, a PC can be viewed in this way:



©IBO 2004 3.2.1 THE CPU

The Central Processing Unit (CPU) is the engine of the computer. It is also known as the chip or processor. The CPU contains a control unit (CU) and an arithmetic logic unit (ALU) and two communication bus systems called the 'primary memory bus' and the 'address bus'.

The control unit controls the sequence of the execution of the program that is stored in the computer's memory.

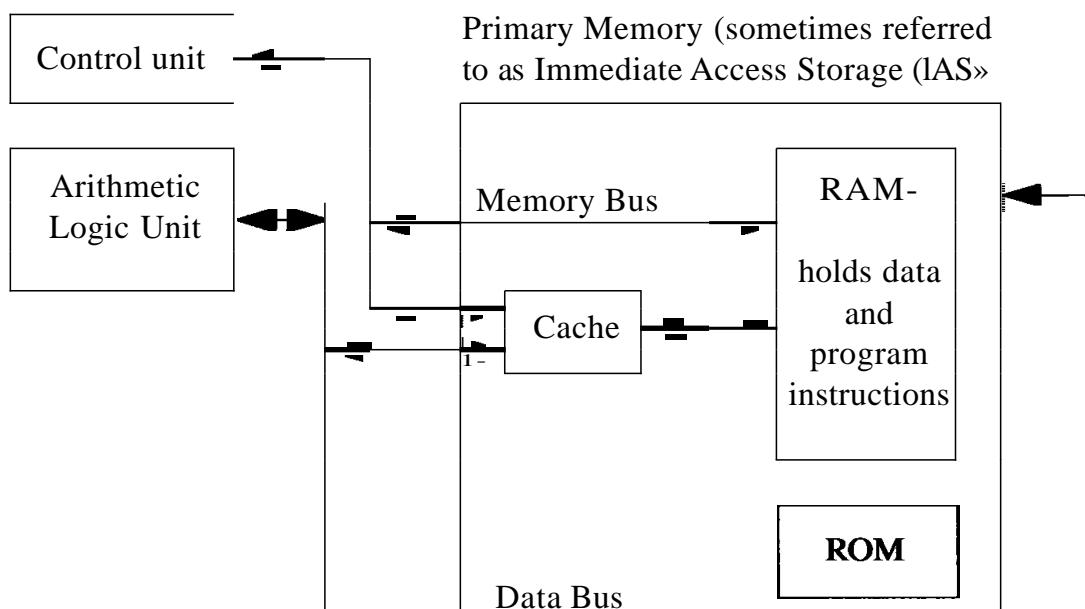
The arithmetic logic unit performs the logical operations such as comparisons and arithmetic operations such as addition.

The control unit of the CPU moves data in and out of the CPU via the data bus, which is connected to the memory of the computer.

The control unit of the CPU accesses memory addresses by using the memory address bus, which is connected to the address section of the computer's memory.

Data is moved around the CPU via the data bus. Memory addresses are accessed via the memory address bus. Each item of data or program instruction is stored in binary in a memory location. Every memory location has a unique address.

Figure 2.6 . CPU block diagram



In some definitions, the primary memory is not considered part of the CPU.

© IBO 2004 **3.2.2 MEANING OF TERMS BIT (B) AND BYTE (B)**

Computers are binary devices. They operate by storing instructions and data as sequences of 1s and 0s. In terms of these 1s and 0s the following definitions apply:

The term bit refers to the individual 1 or 0. In a 32 bit computer each memory location allows 32 bits to be represented. A 32 bit memory bus allows 32 bits to be transferred at anyone time. A bit is denoted by the symbol of the single letter b.

The term byte refers to the standard grouping of 8 bits into a single unit. A single character such as the letter A is stored in one byte. Thus in a 32 bit computer a computer can store 4, 8 bit bytes of data. A byte is denoted by the single letter capital B.

The notion of a byte being 8 bits is also used to denote the file size of files stored on disk.

The following table outlines the meaning of various prefixes which can be used to represent a quantity of bits or bytes. For example 8 Mb represents 8 million single bits and 8 MB represents 8 million bytes.

Figure: 3.5

| Term | Size |
|----------|-------------------|
| Tera (T) | 1,000,000,000,000 |
| Giga (G) | 100,000,000 |
| Mega (M) | 1,000,000 |
| Kilo (k) | 1,000 |

The above terms are commonly mixed with bits (b) and bytes (B) and you need to be careful when dealing with these types of prefixes.

For example there is significant difference between these two terms 10 MB and 10 Mb.

$$10 \text{ MB} = 10,000,000 \text{ BYTES} \text{ and}$$

$$10 \text{ Mb} = 10,000,000 \text{ bits.}$$

To compare these two terms we need to convert each to a common measurement, in this case bits is a useful common measurement for comparison.

$$10 \text{ MB} = 10,000,000 \text{ BYTES} = 10,000,000 * 8 \text{ bits per byte} = 80,000,000 \text{ bits}$$

$$10 \text{ Mb} = 10,000,000 \text{ bits.}$$

Thus 10MB represents a measure that is 8 times greater than 10Mb.

Scientific notation uses the decimal system ie 1 km means 1 kilometre and 1 kilometre is simply 1000 metres. In this case the measurements are precise.

In computing, the measurements such as K are abbreviated as shown above in the table. For example $1 \text{ KB} = 1024 (2^{10}) \text{ bytes}$, but we normally use the term KB as an approximation for 1000 bytes.

©IBO3.2.3 THE TERMS WORD, REGISTER AND ADDRESS AND THEIR USE IN THE STORAGE OF DATA AND INSTRUCTIONS

John von Neumann is often credited with the concept that program instructions can be coded and stored together in memory with coded data - the stored program concept. It was a significant step forward since the ability to store, and therefore change, the program in computer memory allowed the machine to become a general-purpose device. His landmark paper on the subject is a synthesis of work carried out by others, notably J Presper Eckhart Jr and John Mauchly at the University of Pennsylvania.

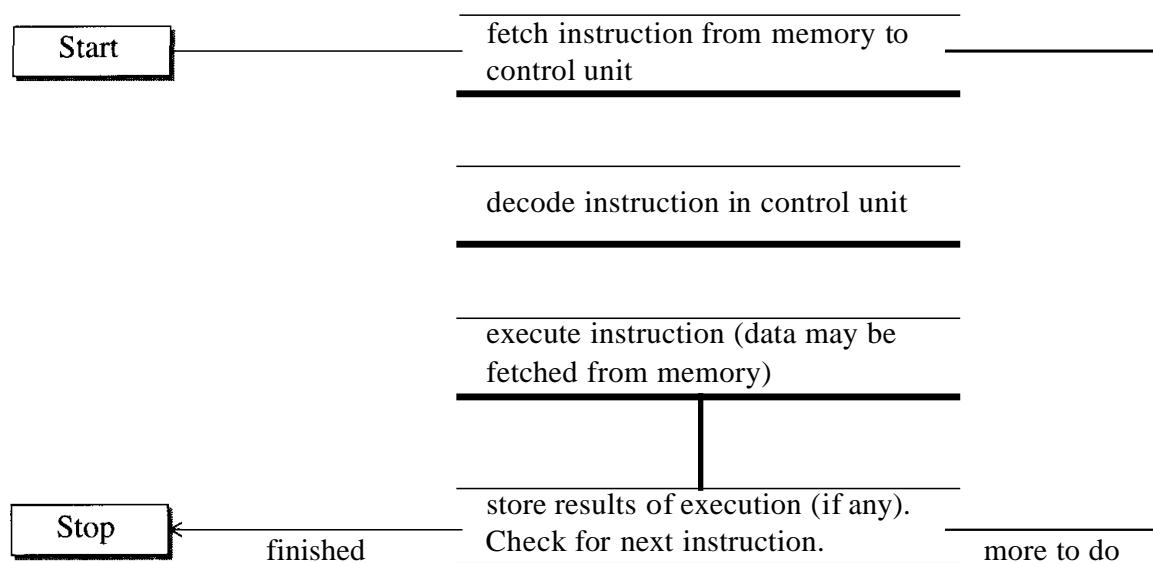
The CPU works with a fixed number of bits, usually a multiple of 8-bits. This is known as a word. Typically, this equates to the size of the registers - temporary storage locations capable of holding just one word. For example, most PC machines have 32-bit processors, therefore the word size is 4 bytes and the registers will each hold 32 bits. The memory will thus hold 32-bit data items too. If a larger size is required to store a given data type (e.g. an 8-byte integer), it will be held in successive memory locations.

Since the registers are also needed to hold the addresses of instructions and data, as well as the instructions and data themselves, the register size also determines how much primary memory a computer can have.

©IBO 3.2.4 MACHINE INSTRUCTION CYCLE

When a computer program is stored as a series of instructions in machine code in primary memory, the following steps are carried out (this model is simplistic compared with the operation of a real processor):

Figure: 3.6



©IBO 3.2.5 PRIMARY MEMORY AND SECONDARY MEMORY (STORAGE)

Primary memory is used by the computer to store currently executing programs and the data used by these programs. The program instructions and data are represented in memory as binary machine code by electronically indicating a 0 or a 1. This is done by using a constant voltage level to indicate a 0 and then using a fixed voltage increment to represent a 1.

Computer Fundamentals

There two main areas of primary memory: Random Access Memory (RAM) and Read Only Memory (ROM).

RAM enables programs and data to be loaded for execution and use. The contents of RAM can be over-written and, therefore, it is a general purpose storage area. The contents of RAM require the voltage level to be maintained. If power is lost, the contents of memory is wiped. For this reason RAM is sometimes known as volatile or short term memory.

ROM is used to store programs permanently e.g. the start-up instructions of the operating system are permanently loaded into ROM. The contents of ROM cannot be changed and for this reason ROM is sometimes referred to as non-volatile memory.

Another area of primary memory that is increasingly being used in the design of computer systems is that of CACHE.

CACHE MEMORY

There are two main types of RAM, fast dynamic RAM (DRAM) or even faster static RAM (SRAM). Naturally SRAM is more expensive and is thus not used for all of the IAS. However, some SRAM can be placed between main RAM and the processor as a temporary store for blocks of program instructions. When the processor has to look for the next instruction, chances are it can be found in the cache. Modern processors also contain a small amount of fast memory incorporated into the processor itself, internal cache, which is speedier still because of the very short distance travelled.

The cache on the microprocessor itself is referred to as Level 1 cache and that between main memory and the processor as Level 2. Whereas the primary memory may be anything from 16 to 128 Mbytes for a typical microcomputer (personal computer), the primary memory of a mainframe or supercomputer may reach Gigabytes in size. The cache memory is typically of the order of a Mbyte.

VIRTUAL MEMORY

For some programs, the primary memory is not large enough to hold the whole program at the same time. Various schemes are used to handle this but candidates simply need to know that fast, random access secondary memory (hard disc) can be used as an extension of primary memory. The area of disc used for this purpose is commonly known as a swap file and parts of the program are swapped into primary memory from here as needed.

©IBo3.2.6 SECONDARY MEMORY, SEQUENTIAL AND DIRECT ACCESS 2004

Secondary memory (storage) refers to permanent memory storage such as disks and tape.

Data stored in primary memory must be written to secondary memory if it is to be saved permanently. Data present in primary memory is lost if the power to the memory is switched off.

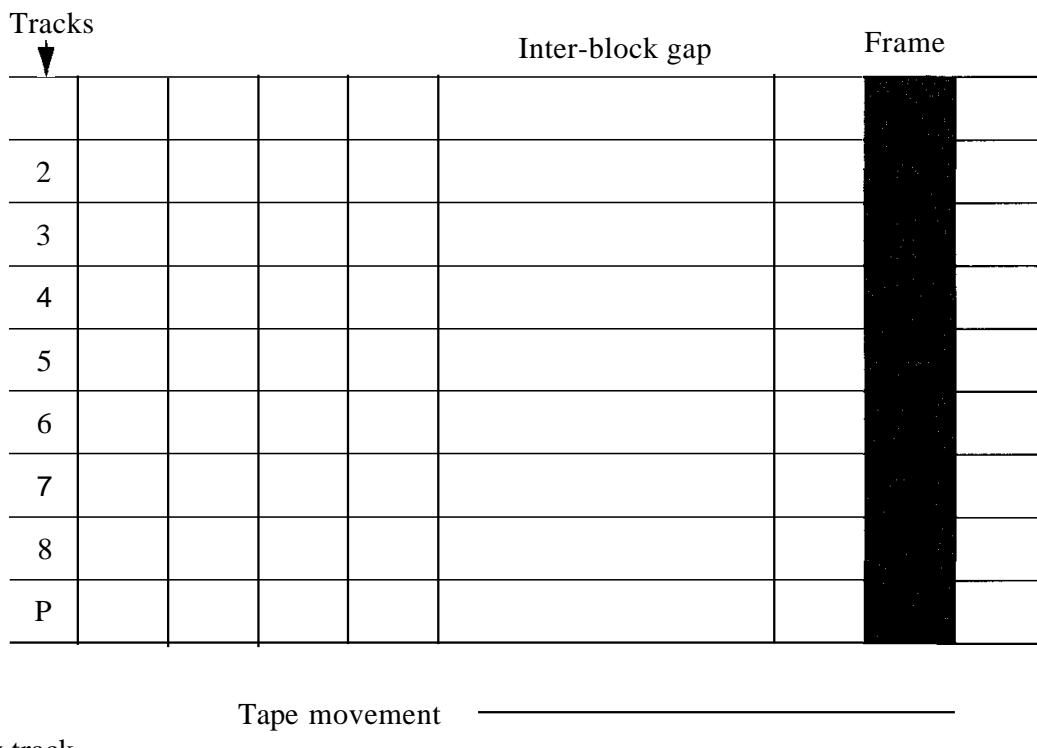
Backing storage is required to keep information which is not needed in memory all of the time and which may be too large to fit into the memory of the computer. Both programs and data are held on backing store. The two main ways of storing data on backing store are sequential-access and direct-access. For example, a payroll program has to access the data file containing all the data on all of a company's employees. It accesses this data one record at a time, one after the other. This is sequential access.

Direct access (sometimes called **random** access) would be used, for example, in a supermarket where details of all of the items for sale are held in a file. The computer needs to locate an item quickly by moving directly to its record.

MAGNETIC TAPE

This is a serial-access medium: to read a given record on the tape you must first pass by all of the preceding records. A typical magnetic tape as used in traditional data processing might be 2400 feet in length and half-an-inch wide. More modern versions are tape cartridges which are quite compact (about twice the size of a cassette tape, typically) and store several Gigabytes of data. They are often used for backing up large volumes of data. While no new systems would specify tape drives of the old reel-to-reel type, there are still legacy systems in use. The data on a tape is stored in parallel tracks:

Figure: 3.7



A bit (binary digit = 1 or 0) can be stored in each track, with 1 byte per frame as shown above. The diagram shows a 9-track tape (other types are possible). A basic unit of data transfer is the byte which is made up of 8-bits. The remaining track (not usually located on the edge of the tape by the way) is a parity track. When a byte is written to the tape the number of 1s in the byte is counted, the parity bit is then used to make this number (of 1s) even (even parity) or odd (odd parity). Then when the tape is read again, the parity bit is checked to see if an odd bit has been lost somewhere.

Figure: 3.8 Examples of Parity Checks

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | P |
|---|---|---|---|---|---|---|---|------|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | odd | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | even | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | even | 1 |

Computer Fundamentals

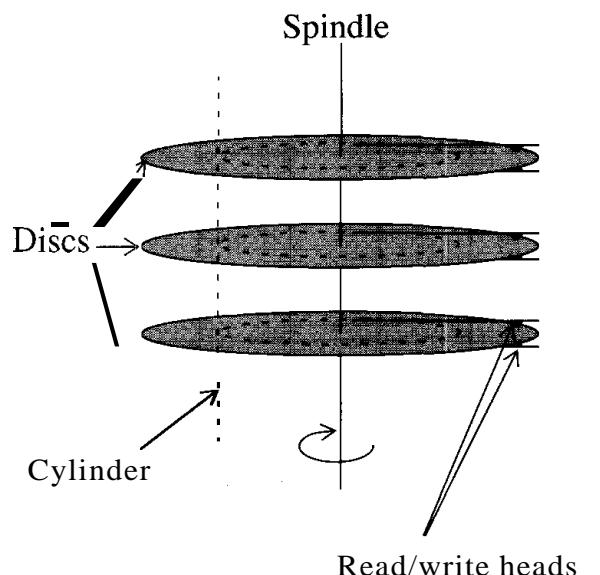
To avoid possible errors, the speed of the tape drive must be held very steady. However, it does take time for the tape to get up to speed so data is written in blocks separated by inter-block gaps - see diagram above.

Smaller tape units are available for microcomputers and are typically used for backing up information from fixed disc packs. Tape streamers are used in microcomputers (PCs) for the same purpose as they are a very efficient way to store a lot of data. A typical tape cartridge used for this purpose can store several hundred Gbytes.

MAGNETIC DISCS

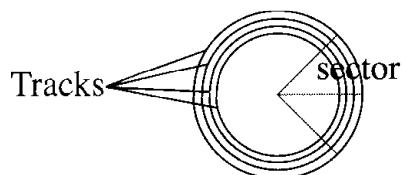
Serial access using tapes has its disadvantages, particularly if you are in a hurry. If you are running some kind of information system in which speed is important, you want to be able to locate a given record very much more quickly than by using serial access on magnetic tapes.

You can compare serial access and direct access by using a musical analogy, to find a given song on a tape cassette requires you to wind on through all those intervening tracks. On a CD player, however, you can directly select the piece you want to hear. There are two main types of magnetic disc in general use, the removable floppy disc and the fixed (or hard) disc. A floppy disc stores up to 1.5 Mbytes or so and a typical PC hard disc stores up to 100 Gbytes or so (these things change quickly, of course). A mainframe disc pack may well store several hundred Gigabytes. A hard disc consists of several plates of magnetic material arranged on a single spindle:



DISC SURFACE TERMINOLOGY (HL ONLY)

The disc surfaces themselves are divided up into sectors and tracks. A typical floppy disc has 10 sectors and 80 tracks. A hard disc may have several hundred tracks on each surface (or side).

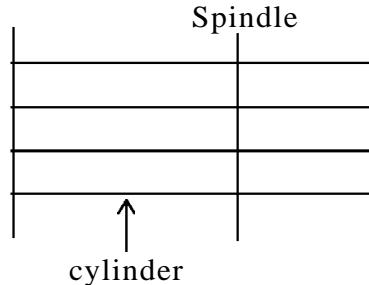


DISC ASSEMBLIES

In the disc pack, the read/write heads are all attached to a common arm, so that all are reading the same track and sector on each surface. This is known as a cylinder.

Each part of the disc pack can be referenced by its surface, sector and track number and this combination will be unique for the pack. This part is known as a block.

Depending upon the computer operating system in use, several blocks may be transferred from disc to Immediate Access Store (memory) in one go and this is referred to as bucket size. Because each block has a unique address, it is possible to store the addresses of files, and their sizes, in a catalogue on the disc itself and data (or programs) can be retrieved by looking up the address in the catalogue.



OPTICAL DISCS

Essentially DVD and CD ROMs are related technology. The CD ROM in your PC is the same as those used for playing music, but the data is stored in a different format. The surface of a CD has lands and pits in the surface each of which can represent binary states. These very small irregularities are read by a reflected laser beam. In practical terms CDs are useful for distributing information because they are portable and rugged (unlike, say, floppy discs) and they store much more data (from around 600 Mbyte for conventional CD up to 2.7 Gbytes for DVD formats). There are also CD-Rs (recordable CDs) on the market which you can write to once but read many times (Write Once Read Many =WORM). Also increasingly affordable are the CD-RWs which can be used like large floppy discs. These compete with such magnetic media as ZIP discs which typically hold a few hundred Mbytes of data.

EXERCISE 3.3

1. As a programmer you are asked to determine the size of a file. The file is to contain text data that is collected from an external source. The data for the file will be transmitted once per day for a period of 3 hours over a data line that transmits at a rate of 100,500 bps.
Calculate the daily size of the file and use appropriate prefixes.
It is required that the file be stored online for a period of two years and that a duplicate will also be need to be kept. Estimate the size of the data at the end of the two year period, assuming that no data is stored at the start of operation.

2. Justify a choice of secondary storage medium to store the daily file and the copied file.
Why might these be different?

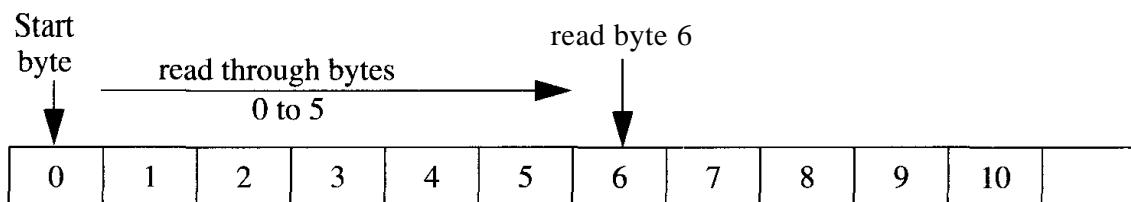
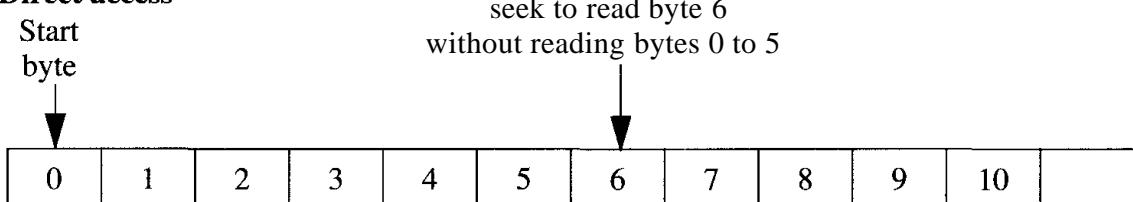
3. The daily file needs to be copied into memory and the system manager is worried that there might not be sufficient room. The computer they use has 2 MB of RAM of which 10% is taken up by the operating system, 20% by the normal I/O functions (i.e. display) and 40% is taken up by the other applications that run. Has the manager reason to worry and if so, by how much does the RAM need to be increased. Is this likely to be feasible?



SEQUENTIAL (SERIAL) AND DIRECT ACCESS

Sequential file access means that data records cannot be accessed without reference to other records in the file. A sequentially organised file on disk or tape can only be accessed in a sequential manner. For example, say we wish to access the record in the 100th position in the file and that the file is organised in a sequential manner. In order to do this we must read through the previous 99 records. The logical order of the records is important in terms of access.

A sequential file implies that the records are ordered in some way. Usually this means the records are sorted according to the value of the primary key. A serial file has a sequential organisation, i.e. the records must be searched one after the other, but the records are not sorted into any order.

Figure: 3.9**Sequential access****Direct access**

Direct file access is much faster in that data records can be retrieved by specifying the record position without reference to other records in the file. The file management system can then identify the track and sector position that the record is in and directly access the data by positioning the read arm of the disk unit in the appropriate way. The logical order of the records is not important in accessing the records. In fact the records can be in completely random order. Hence the term random access file.

In both cases, a key field is required called a primary key, which is used to identify the record. For example, a car registration number is a primary key. In the case of a direct access file it is necessary to be able to link this primary key to the record number in order to effect direct access.

ADVANTAGES, DISADVANTAGES AND APPLICATIONS OF SEQUENTIAL AND DIRECT ACCESS.

Sequential file access is used to store data that is normally processed in the order that it is stored. For example, a set of temperature data is normally only processed from the start to the finish and it is not necessary to access an individual record very often.

The key advantage is that it is simple to organise and there are few processing overheads associated with dealing with the file. Data stored in a sequential file can be written to either a disk or tape.

The main disadvantage is that interactive/online processing is not possible using sequential access because access times would be too slow.

Direct access files are used where high speed access is required. For example, a large bank would store their data on customers in a direct access file. In order to ensure that data could be retrieved quickly the customer's bank account number would be the primary key. This number would be used to gain direct access to the record number.

The key advantage is that records can be accessed quickly. The main disadvantage is that data can't be accessed in sequential or sorted order as the records are placed randomly on the disk. To gain sequential or sorted order access an index file is required.

Direct access files are only possible to implement using disk technology and can't be implemented on tape.

EXERCISE 3.4

1. Define the terms file, record and field.
2. Given that a file stores details about students, estimate the size of the record, using the normal prefix. You should include at least 10 fields. Assume a string takes one byte per character and a number (real or integer) takes up 4 bytes per NUMBER, not digit. (Note: related to previous section).
3. Define the term sequential file access.
4. Define the term serial file.
5. What advantage does a sequential file have over a serial file?
6. Define the term direct access.
7. What is the difference between the terms file organisation and file access method?
8. Why does DISK allow both sequential and direct access?
9. Why does tape allow only sequential access?
10. In general, compare the advantages and disadvantages of the use of either access method.
11. For each of the following, state why, and give reasons why, one access method is likely to be preferred to the other:
 - * Customer file in a video store.
 - * Customer file in a bank.
 - * Year's set of daily maximum and minimum temperatures for a single town.
 - * A file required to store the hours worked and rate of pay for each employee.
12. Make a list of 10 further applications that use files. For each, state what access method is used and give an advantage and disadvantage associated with the access method for each application.



© IBO 2004 **3.2.7 MICROPROCESSORS**

A microprocessor is a device similar to a CPU but lacking significant amounts of primary memory. It has a single program stored in non-volatile memory and typically a number of registers for volatile storage. It is, therefore, similar in some ways to a computer, but lacking the ability to change its program.

Microprocessors are very common. There are, perhaps, a couple of dozen in your home, school and car. Examples of microprocessor applications include the following (cars and washing machines are specifically mentioned in the program guide):

| Object | Function(s) of microprocessor |
|-------------------|--|
| Car | Controlling injection of correct amount of fuel to the engine; automated braking systems; airbag controllers; in-car navigation systems; automatic car washes; |
| Washing machine | Controlling water level, temperature, wash time, spin speed. |
| Camera | Controlling exposure time; auto focus calculation; conversion of image to storage as a file (digital camera). |
| Video Tape player | Timing; programming, station selection; |
| CD player | Track selection; extracting time and other track details; conversion of optical to electrical signals; |
| House | TV with remote control; TV with teletext; control of central heating; control of air-conditioning, digital clocks; microwave ovens; dishwashers; printers; computer ports (USB, printer, modem); burglar alarms; mobile telephone. |
| Classroom | Calculators; digital watches; aircon or heating control, security systems. |

Often, microprocessors are used for similar tasks and may have a standard set of input and output ports for analogue data. These are usually known as micro-controllers.

EXERCISE 3.5

1. Draw and annotate a diagram of a CPU. Include in your diagram RAM, ROM and CACHE. Make sure that you include the key components including any internal communication pathways.
2. Describe the function of the ALU.
3. Outline the function of the CU.
4. Describe the purpose of the memory and data bus.
5. Outline the role of RAM and ROM and explain why ROM can't be used to store user programs or data during program execution.
6. Outline the role of CACHE.
7. Define the terms primary and secondary memory.
8. Describe the purpose of primary and secondary memory.
9. Describe by use of a diagram how a disk operates.
10. Make a list of all the microprocessor controlled devices you use in one day.

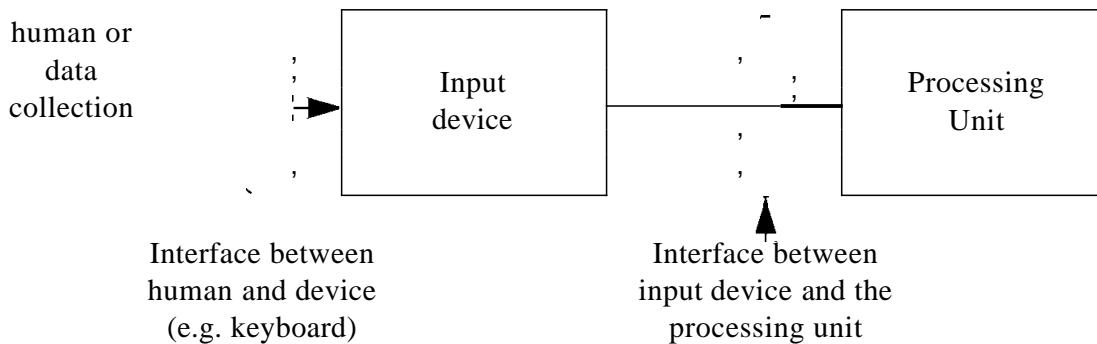


© IEO 2004 3.2.8 INPUT DEVICES

The fundamental role of an input device is to enable data to be entered into a computer system. Input devices are often used by humans to input data, e.g. a keyboard, but they can also be directly connected to a computer e.g. a bar code reader. The input device therefore acts as the connection point between the internal electronic world of the computer and the external world from whence the input comes. An input device must, therefore, be able to be interfaced, or connected, to the computer and to be interfaced, for example, with the human using the input device such as keys on a keyboard.

The diagram in figure 4 outlines these key features of input devices.

Figure: 3.10 - Input Process



We now turn our attention to a discussion of the main features, advantages, disadvantages and applications of a range of computer hardware used as input devices.

LIST OF DIFFERENT INPUT DEVICES

Keyboard

A keyboard is really a series of switches each of which generates a series of different pulse codes so that the computer knows the one that was pressed. They are very useful for typing in text, with word processors for example. Ergonomic keyboards provide wrist rests and place the keys at a more comfortable angle for prolonged use; thus preventing injuries (such as repetitive stress injury (RSI) or carpal tunnel syndrome).

Keyboards, of course, are very useful for entering and editing text and they feature short cut keys (function keys and numeric keypads) to speed up common operations.

Mouse

The movement of the mouse generates a series of pulses by which the computer can tell the direction of mouse movement. This is often used to move a pointer on a screen, select a menu option or change and create different windows in a Graphical User Environment (GUI). The most common example is the Windows operating system from Microsoft or the systems used by Apple Macintosh machines. This type of environment is sometimes called a **WIMP** environment:

WWindows

I leons

M Menus (or Mouse)

P Pointers (or pull-down menus)

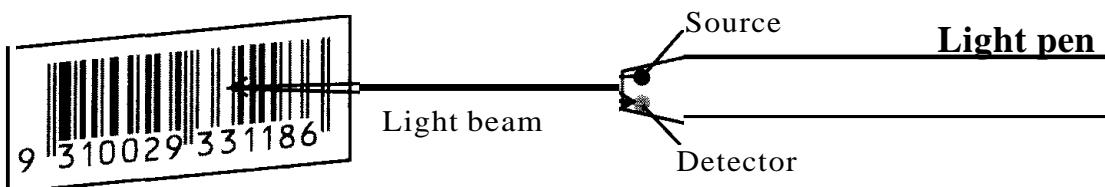
Graphics Tablet

A graphics tablet is a kind of flat board on which a pen is moved. The pen holds a magnet and underneath the tablet is a fine grid of wires. Movements of the pen cause disturbances in the electrical pulses in the wires and the co-ordinates of the pen can be detected. It is used, for example, in sophisticated computer art packages and video-editing systems for special effects.

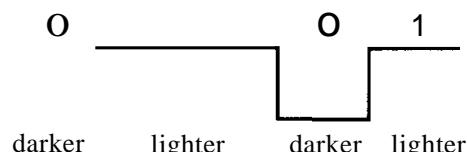
Light Pen

Works with light, there are two main types. One is used to point to areas on the screen and works by detecting the light emitted by the TV. TVs and monitors work by directing a narrow beam of electrons onto a fluorescent surface. This beam performs a regular scan, starting at the top left-hand corner of the screen and travelling down in closely spaced lines until it reaches the bottom. All of this happens too fast for us to detect but it does mean that the beam passes a given spot at a time which can be calculated from a knowledge of the beam's speed of travel and the number of (beam, not text) lines on the screen. Thus, this type of light pen can be used to return a position on the screen. It is mostly used in drawing programs and CAD (Computer-Aided Design) applications.

The other type of light pen, seen in shops, emits a light beam and detects the amount of light reflected. Dark areas reflect less, light areas more. A sensor detects the amount of light reflected.

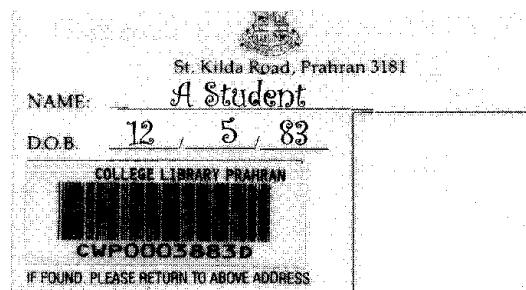


As the beam passes, light areas reflect more than the dark bars, giving a stronger pulse.



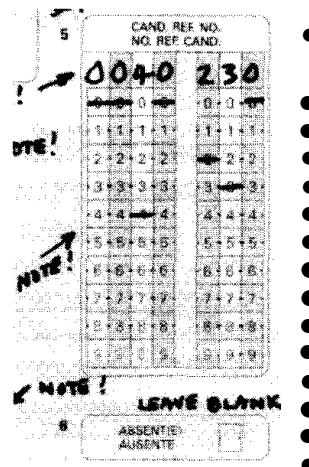
A typical use of this kind of light pen is to read bar codes, e.g. in department stores and libraries - see picture below). Supermarkets typically use a **laser scanner** which is fixed because this speeds up the reading of the barcodes, particularly because many items sold in supermarkets are easily handled and it is then quicker to pass them over a bar code reader than to use a hand-held light pen.

A library card with a bar code:

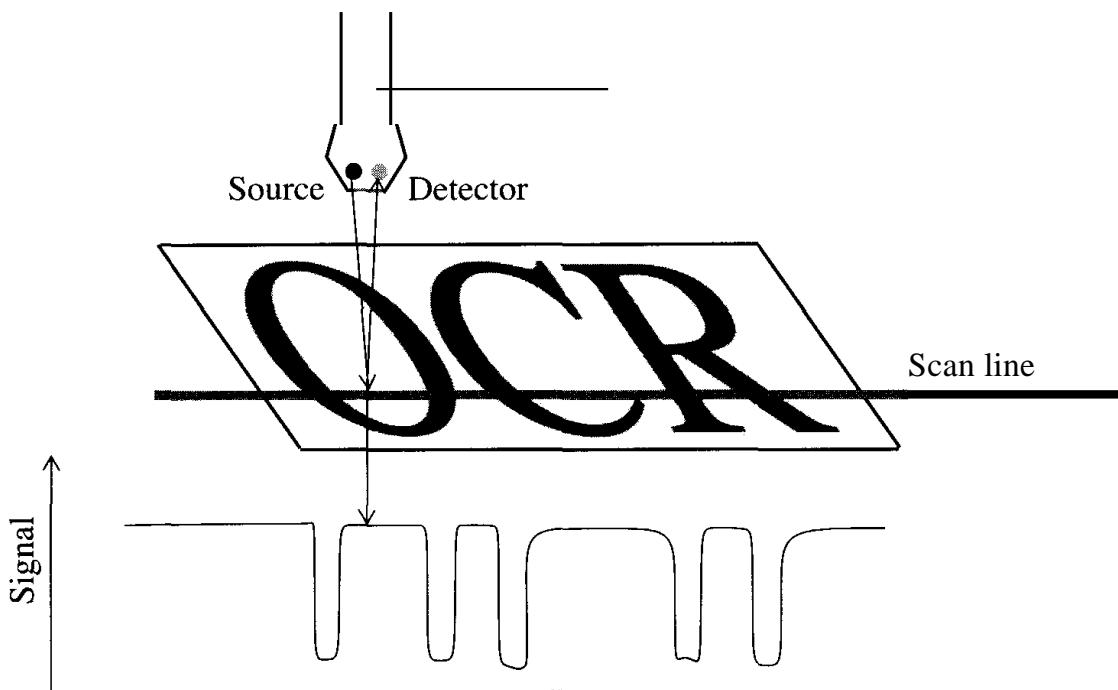


Optical Mark Recognition (or Mark Sensing)

This uses the techniques described above to detect black marks on white paper. A common application is in examinations where multiple choice papers have several boxes; candidates mark the box corresponding to the correct answer (they hope!), usually with a soft pencil. The forms are an example of pre-printed stationery, some information is already printed on the form (e.g., the questions). Your IE registration form is a good example of this. The picture shows part of the form used in multiple choice examination papers (together with some helpful hints on its use).



Optical Character Recognition

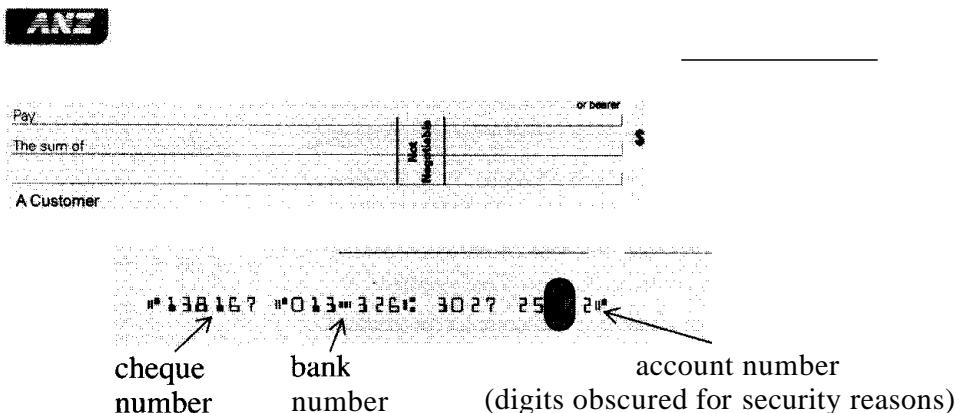


The principle of using reflected light is also applied to scanners, fax machines and optical character recognition (OCR). The scanner passes a beam of light over a page and measures the reflected light. This gives a two-dimensional pattern of light and dark, which can be represented by Is and Os.

When you have the characters or drawings converted to a binary pattern then you can send them as a string of electrical pulses to a fax machine or import them into a computer program. With OCR, you can try to match the binary patterns to patterns stored in the computer's memory and each recognized character is then assigned its standard code (ASCII usually).

Computer Fundamentals

Magnetic Ink Character Recognition (MICR)



The 'funny-looking' numbers at the bottom of the cheque are encoded in magnetic ink for use with MICR readers at the bank. Some banking systems prefer MICR because of the increased reading speed and extra security against forgery compared to OCR. However MICR is much more expensive than OCR. When the cheque is received by the bank, the amount has to be encoded in the blank space on the right before the cheque is processed.

Automatic data entry

The scanning and reading techniques described above can be grouped under the general heading direct data entry methods. Do not confuse this with automatic data entry. This occurs when a sensing device is connected permanently to a computer and the computer receives data from the sensor at intervals. Sensors are used to collect data when processes are automated, such as car washes, automatic washing machines, heating control in buildings, petrol delivery in a modern petrol (gas) station. These processes may be controlled with microprocessor applications rather than general purpose computers. In this case the program will be stored in ROM and very little RAM will be needed.

Digital cameras

These are becoming increasingly common, with photographic quality and prices are now comparable to good quality conventional cameras. They can be used with a monitor for applications like video conferencing or simply to take photographs to display on a web page. Usually, some form of data compression is used when storing and transmitting photographic images to save space and decrease transmission time. A common standard is JPEG (Joint Photographic Experts Group).

Speech recognition

In this technique, the application digitally records speech and attempts to match the digitized patterns to the patterns of known words in memory. Since there are a wide variety of speech tones, accents and pitches, each individual using such a system has first to 'train' the application with a set of standard words. The accuracy is said to be at about 90% for dictation into a word processing document.

Robotics

A huge topic in its own right, robots are not the kind of thing you see in science fiction movies but typically pieces of industrial machinery used in dangerous or tedious, repetitive tasks. A robot is a programmable machine.

EXERCISE 3.6

Note the section on input devices is part of a level 3 objective, hence students can expect questions that use the terms: analyse, compare, discuss, evaluate, explain.

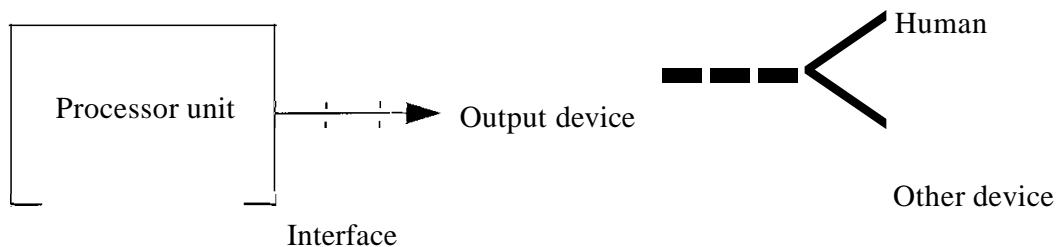
1. Construct and complete the following summary table. Use the following headings:
 - Name of input device,
 - key features,
 - advantages,
 - disadvantages,
 - brief list of possible applications.
2. Explain why a supermarket may choose to use a bar code scanner.
3. Discuss an advantage and a disadvantage of using a standard keyboard to allow customers in a store to look up the price of an item in a store.
4. Compare the use of voice recognition as opposed to the entry of a password via a keyboard to allow employees entry to a secure area of a building.
5. Activity. Robotics is a very exciting field. Using the resources available to you explore how robots work and the role input devices play in their functioning.



© IBO 2004 **3.2.8 OUTPUT DEVICES**

In a similar way to input devices, output devices are required to link the world of the computer to either our human world (i.e. display information on a screen or printer), or to interface with a communications channel to another computer device. Figure 5 outlines this key concept of interfacing to the output device.

Figure: 3.11 - Output Process



A range of output devices are now briefly discussed. It should be noted that a number of devices can act as both an input and output device. A good example is the touch sensitive screen or monitor.

Video Display Unit or Monitor

A VDU is like a TV screen but often has higher resolution, the combination of VDU and Keyboard together is usually referred to as a terminal. A typical mainframe computer will have very many such terminals connected to it. On a PC - which is essentially a terminal with its own microprocessor and RAM - the VDU is often known as a monitor. VDUs may be monochromatic

Computer Fundamentals

(single colour) or polychromatic. The definition or resolution of a VDU is a function of the number of picture elements or pixels on the screen itself. Each pixel requires a memory location to store information about its state (usually its colour). Therefore polychromatic VDUs require more memory to store information about the state of the screen. Very high resolution monitors (Graphical Display Terminals or GDUs) might be used in special applications such as cartography, commercial graphics and design work.

LCD

Liquid Crystal Display such as is found on small computers like Palm Pilots and other PDAs (Personal Digital Assistants). Because this type of screen doesn't perform well in low light conditions, it is often 'backlit'. However, they consume very small amounts of power compared to conventional displays and are thus useful in battery operated devices.



For laptop computers, passive and active LCD screens are available. The active types have one charge controller (colour element) per pixel giving a brighter and sharper display. Other advantages (besides low power) of these screens over conventional monitors using CRTs is that they are lighter, flatter, thinner and give out virtually no radiation. They are now replacing conventional CRTs on desktop machines as well.

Printers

Printers may be classified in a number of different ways:

- By the amount of text, i.e. characters by line or lines or page.
- By character formation: matrix vs non-matrix.
- By method of printing: impact vs non-impact.
- By colour: colour vs black and white.

Dot-matrix printers used to be about the cheapest option but have now been overtaken in price/performance by inkjet printers and are sometimes still found on low-volume systems such as PCs. They are character, impact and matrix printers, each character is produced by a set of pins punching an inked ribbon onto the page. The lowest quality are 9-pin printers, more pins (24-pin) means higher quality. Typical printing speeds are 50-200 characters per second. You can still see them in small businesses used for printing credit card and other receipts.

Daisy wheel printers by contrast work like old-fashioned typewriters, each solid character, held on the end of a spoke, strikes through an inked ribbon onto the paper. The quality is better than that of a dot-matrix printer but the character set is fixed and, as it is a non-matrix printer, graphical output is not possible. This type of printer is becoming rare. Good quality is now obtainable by other printers (inkjet, laser) at comparable or even lower cost and without the associated noise of the daisy wheel.

A typical **Lineprinter** is based on a similar concept but there is a solid character for each position across the page and therefore the print-head does not move. Since they print 1 whole line at a time they are very much faster than character printers.

Dot-matrix and daisy wheel printers are rarely seen and have been superseded by Inkjet and Desktop Laser printers. **Inkjet printers** hold a cartridge of ink which is sprayed onto the page in

small dots, they are matrix printers. The dots form the characters much like a dot-matrix printer. However, because the ink is liquid it spreads a little on the page and makes a much smoother appearance. Colour versions are also available and are more versatile than their dot-matrix equivalents. Laser printers print a page at a time using a whole-page matrix of dots, they are very similar to photocopiers in action. A laser beam creates very small dots of static charge on the paper and powdered toner (ink) is attracted to the charged areas. The paper is then heated and the toner melts onto the page. The quality of these printers is very high and colour laser printers are now increasingly affordable.

Plotters

There are two main types: electrostatic and pen plotters. Electrostatic plotters (available in black and white and colour) make an image by burning specially prepared paper with a small spark. Pen-plotters hold a cartridge of several pens of different colours and actually draw onto the page, giving a higher quality output. Both types are able to plot onto large sheets of paper which can be of fixed size (flatbed plotters) or on a large roll of paper (drum plotters).

Touch Screen

These screens are used both as input and output devices, the options can be shown with text and or graphics and the position of a press on the screen can be detected. These screens are often used where general information is being provided (banks, hotels, shopping centres) for users who may not be competent computer users (yes, there are still some of these!).

EXERCISE 3.7

1. As with input devices, construct and complete a summary table. Use the following headings: Name of output device, key features, advantages, disadvantages, brieflist of possible applications.
2. A school wishes to print the computerised student reports centrally. The school currently uses an 8 page per minute laser printer. There are 1500 students in the school and each student studies 8 subjects and receives a 'homeroom' report also. each report is an A4 single sheet. What are likely to be the limiting factors associated with the current output device.
3. In relation to the above question, recommend what action the school could take to ensure that the output task is accomplished in a reasonable timeframe.
4. In an architect's office they use a plotter as a key output device. Explain why the plotter is used in preference to an A4 laser printer.
5. A touch sensitive screen can act as an input and output device. In relation to a home personal computer (PC) compare the advantages and disadvantages of using such a device as compared to using a normal keyboard and monitor.



©IB03.2.9 RECENT DEVELOPMENTS IN COMPUTER SYSTEM ARCHITECTURE

This section reviews some recent developments in the following areas:

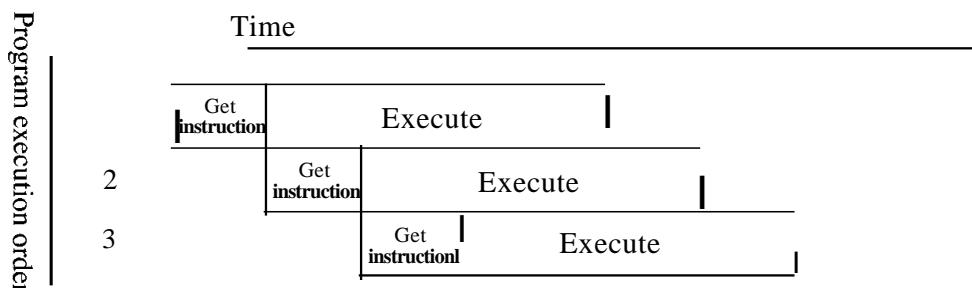
- Processor architecture.
- Memory technologies.
- Secondary storage devices.
- Data communications.

Processor Architecture

The architecture in which a single instruction is fetched into the CPU then decoded and executed is often called the 'von Neumann architecture' after John von Neumann who first described it in a formal paper. This can be restrictive (the von Neumann bottleneck) since, for example, it is often more efficient to have a single task performed by many people (as long as the task can be split into suitable sub-tasks).

Recent developments have included the single and multiple pipeline architectures. In the single version an instruction can be fetched and another decoded while the first is still executing:

Figure: 3.12



In the multiple version, pipelines work in parallel to increase the rate of processing of instructions still further.

A similar idea is to use multiple processors in parallel, but in both these cases some additional co-ordination is needed in case one instruction depends upon the results of another executing at the same time or later on.

Supercomputers and mainframe computers make extensive use of multiple processors to share the workload. In relatively recent developments such as Deep Blue and the Cray T3E supercomputer, massively parallel architecture is employed.

Primary memory Technologies

The standard RAM technologies have been SRAM (Static RAM) and DRAM (dynamic RAM). Static RAM is faster and more expensive so is used for cache memory. DRAM is usually used for the main memory. Refer to the activities section for places where you can find out more about the complexities of this subject.

The point always made about ROM is that it cannot be written to except by a special machine (the blower). An early development to allow testing of programs stored in ROM was the EPROM or Erasable Programmable ROM. These chips can be recognized by a special quartz window on top

of the chip. Shining UV light through this window will erase the chip's contents and allow it to be re-programmed.

With the very rapid advances in modem and other technologies (scanners, CD-Writers, digital cameras), there was a need for a ROM that could be upgraded in the component itself and these newer devices are fitted with Flash ROM in which the program can be changed by downloading suitable software from a website.

Secondary Storage Devices

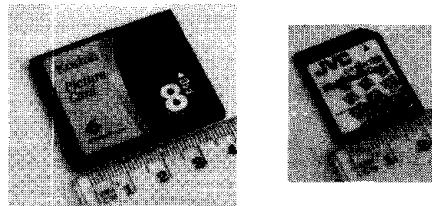
We have mentioned ZIP discs and drives and CD-RW which are relatively new techniques for storing large (100MB - 3 GB) data on portable media. There are a number of new storage media for portable devices like MP3 players and digital cameras. These include:

- Smart media.
- Compact Flash.
- Memory stick.
- Multimedia Card.

These media contain non-volatile memory, are typically very small and have low power consumption. They are ideal for use in small portable electronic devices which need to store data. Typical sizes range up to 64 Mb on a card about twice the size of a standard postage stamp.

Re-writable flash memories which are non-volatile are now common, typically in 128 MB and 256 MB sizes. These are small, robust and easily connected to USB ports. They may even contain executable code as well as data.

Media cards from two digital cameras



Data Communications

Increasingly available are two similar types of device. The mobile phone, which has data acquisition features for web browsing and email reading. With the advent of WAP (Wireless Applications Protocol) this technology is expected to improve the amount of data that can be transferred to a mobile phone thus enabling portable web browsing, contact with company intranets and VPNs (Virtual Private Networks).

The PDA (palm top or personal digital assistant) which is useful for storing contact information and can be synchronized with a PC. Probably these two devices will converge so that you can carry your schedule and reminders with your telephone and, again, these can be linked to your company's network (and company databases) too.

These kind of technologies are supplemented by ever faster 'backbones' (systems for transmitting data in networks, including the internet) and associated technologies such as ATM (asynchronous transmission) via cable modems (about 30 times faster than a 56 k modem).

Further resources

It is always worth reading about recent developments in technology, they happen faster than this book can cover them. Try computer magazines and technical websites for items of interest.

Computer Fundamentals

The examiner will probably not ask questions on specific technologies, unless they are part of the Case Study, so you don't need these; a more likely type of question on this topic is:

Outline any two recent developments in processor architecture which have improved the speed at which programs can execute [4 marks]

Then you are free to briefly describe any two that you happen to have read about. While you do not need details for your studies you may want to explore any of these the topics for an extended essay.

Websites

Of course, these may have moved or changed; you can use any search engine to find new ones.

More detail on processor architectures can be found at, for example, pages by John Morris of The University of Western Australia. There you will find notes such as:

<http://ciips.ee.uwa.edu.au/~morris/CA406/pipelines.html>

Deep Blue (a massively parallel chess-playing computer) information can be found at:

<http://researchweb.watson.ibm.com/deepblue/meet/html/d.3.2.html>

Supercomputers and their applications are described at:

<http://www.cray.com/supercomputing/>

Cray's customers include:

Department of Defense Naval Oceanographic Office (<http://www.navo.navy.mil/>)

Electronic Data Systems (<http://www.eds.com>)

NASA's Goddard Space Flight Center (<http://www.gsfc.nasa.gov/>)

United Kingdom Meteorological Office

(<http://www.met-office.gov.uk/research/nwp/numerical/computers/index.html>)

NOAA High Performance Computer Center (<http://www.hpc.noaa.gov/>)

Great information on hardware topics can be found at:

<http://www.hardwarecentral.com/hardwarecentral/tutorials> including a great article by David Risley on Memory types.

Since these sites may change, look for updated info at <http://www.ib-computing.com>

© IBO 2004 **3.3 COMPUTER SYSTEMS**

© IBO 2004 **3.3.1 DEFINE OPERATING SYSTEM**

An 'operating system' is a collection of programs which deal directly with the hardware system and sub-systems, provide user interface(s) and log the activities taking place in the system.

Examples of operating systems are Linux, MacOS and Windows. Network operating systems, such as Novell Netware, might be encountered in some schools.

© IBO 2004 **3.2 OPERATING SYSTEM FUNCTIONS**

Peripheral Communication

Printers, mice, keyboards, monitor screens, robot arms etc., are all peripheral devices (hardware outside the CPU). Hardware only deals with data coded into binary machine code (unless digital to analog conversion has taken place, see section 3.5.9). The operating system keeps track of the device drivers - software designed to interface directly with the hardware - on your system and signals if they are not operating correctly. The **AS** provides a standard interface between hardware devices and applications; an application can thus simply use a <print> type instruction within the high-level language rather than the low level commands associated with sending characters, line breaks etc. In practice it's a little more complicated.

Coordinating concurrent processing

Processes or jobs are running on your system all the time the computer is on. The curious user can get a list of these processes - although it won't always mean much to the un-initiated. The **AS** handles the loading and unloading of these processes to and from primary memory.

Memory management

The **AS** ensures that each process operates in its own (virtual) memory space and doesn't change memory belonging to another process (well, that's the theory, I'm sure we've all seen error messages where there has been a "memory access violation" or something similar). The **AS** also deals with the moving of parts of processes to the swap file mentioned earlier.

Resource monitoring

Running processes can be allocated resources - how much processor time they can have, how much memory they need and so on. The OS attempts to keep all processes running by managing this access. It doesn't always work, you may have noticed.

Accounting and security

Major operating systems designed to be used in a multi-user, networked, environment have to try to make sure that only those users who are registered with the **AS** can get access. The manager, via **AS** functions, controls this feature. Accounts are also usually kept of the activities conducted by each user. These logs of activity can be huge but may be required if there are security or other problems; one user may be found to be running a process that is 'hogging' all the resources. It is possible to discover the activities of unauthorised users (hackers) by looking at the 'audit trail' kept by the accounting software.

Program and data management

This function includes some of those already described, the **AS** must be able to keep track of which files, ports and other data resources a particular program is using to read or write information. Otherwise, for example, one program might end up writing to another one's files.

The OS will also handle the transfer of data and/or program instructions from files into primary memory and vice versa.

©IBO 3 **3.3 CHARACTERISTICS OF COMPUTER SYSTEMS AND A COMPARISON OF THESE CHARACTERISTICS AND APPLICATIONS OF DIFFERENT KINDS OF COMPUTERS**

Computer systems are made up of input, processing and output-communications hardware devices and the systems and application software required to operate and connect (interface) the components so that they can function and communicate.

Computer systems can be small and consist of single computer such as a personal computer (PC) or can be composed of a number of computers linked together into a large network of computers.

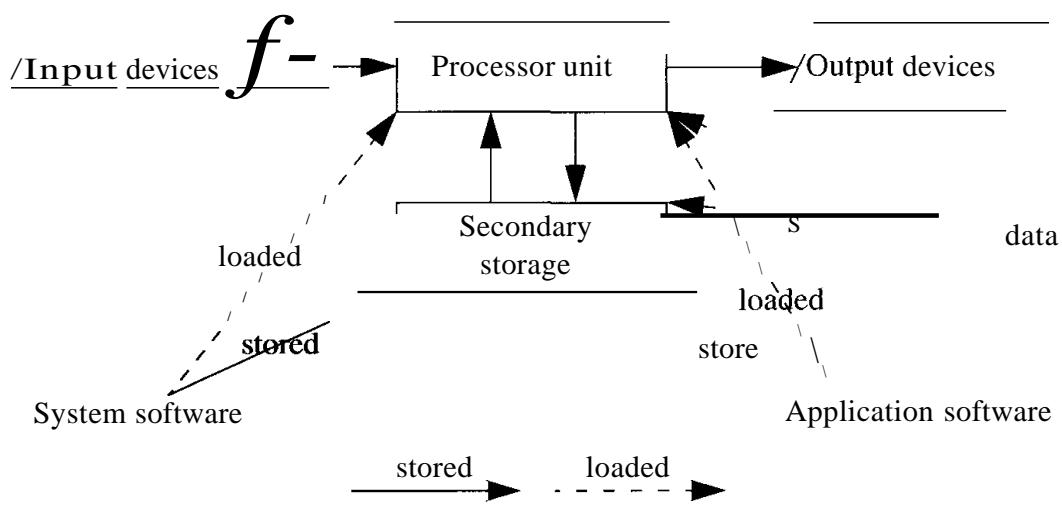
Computers can also vary in the size of their memory and secondary storage and speed of processor. And, they can vary depending on the type of operating system that is loaded.

A typical PC's mode of operation is as a single user computer. This means that it can be used by one person at one time. Most PCs also allow a single user to perform more than one task simultaneously e.g. to edit a word processed document and download a file from the Internet at the same time. This is known as 'multi-tasking'.

A single computer such as a PC consists of the following components:

- Input devices e.g. mouse, keyboard, scanner.
- Processing device: processor, primary memory and control unit.
Output and communication devices e.g. printer, modem.
- Backing store (Secondary storage) unit e.g. Hard disk.
- System Software e.g. operating system.
- Application software e.g. accounting package.
- Stored Data e.g. data files.

Figure: 3.13 PC Computer System



Larger computer systems also have these same components but also have connections to a larger number of peripheral devices and to other connected computers via network connections.

Computer systems can be classified into a number of categories. These include: Personal Computer, Portable Computer, Network Computers (Servers), Mainframes and Super Computers.

PERSONAL COMPUTER (PC S) AND PORTABLE COMPUTERS

The desktop PC (or microcomputer) typically has 32-128 Mb of RAM (primary memory), 5 - 20 Gb of Backing Store, a CD or DVD drive, keyboard, mouse and monitor. Monitors can range from simple mono screen resolution to high quality graphics screens.

Processor speeds of PCs have increased significantly over the past 5 years. As an example, the Pentium 4 is rated at speeds up to 2.2 GHz, which is extremely fast, especially if compared to the processors used in PCs in the early 1990s. PCs and portable computers normally use a 32 bit word size.

Portable computers have similar characteristics but can usually also run on batteries with a life of, typically 2 – 6 hours. Many will have low- power versions of standard microprocessors. Some may do without CD ROM to save space and weight. Monitors can also be colour or mono and range to high quality high resolution display.

Both utilize memory cache to improve performance. The Pentium 4 processor provides 8 KB of level 1 cache and 256 KB of level 2 cache.

Both provide multi-tasking, but are used in single user mode.

PCs and portable computers typically cost in the range of \$2,000 to \$10,000.

PCs have the following typical dimensions: processor box is between 30 to 5cm wide by 25 to 40cm deep by 10 to 15cm in height.

I/O devices are connected via a serial (e.g. keyboard) or parallel port (e.g. printer). Typically the number of 110 devices is limited. The recent development of the Universal Serial Port (USB) has been added to improve speed and to enable devices to be connected while the machine is running.

MAINFRAME COMPUTERS

Mainframes are designed to run a range of application software and process a considerable volume of transactions for a range of logged in users all at the one time i.e. use a multi-user mode of operation. It is not uncommon for a single mainframe computer to have thousands of simultaneous users all demanding resources from the mainframe.

A modern mainframe such as the IBM S/390 series can process at the rate of 900 MIPS (Million Instructions Per Second) when configured with 10 microprocessors. The processors are normal Pentium style chips that power the PCs, but, because of the mainframe's design, vastly more RAM and cache can be used by the processors. The processors are also often used in parallel. The device also uses a very high capacity disk configuration referred to as a 'disk farm' that can store and access TBytes of data. They can also have a considerably greater number of peripheral devices attached.

Mainframes cost considerably more than PCs and range in price into the millions of dollars.

Computer Fundamentals

Mainframes have traditionally been physically large and need to be housed in very expensive and sophisticated computer rooms. These incorporate expensive air cooling systems to keep the processors and disk units cool during operation.

SUPER COMPUTERS

Supercomputers are designed to run very complex programming tasks that simply require a very large amount of processor time to execute. They are very expensive with prices in excess of \$10m. The US Weather Bureau uses a super computer that runs at around 50,000 MIPS with 4 GBytes of primary memory and 4 TBytes of secondary memory.

NUMBER OF USERS:

MULTI-USER AND SINGLE-USER COMPUTER SYSTEMS

Computer systems can be characterised by the number of users that can access the system at any one time. A PC is a single user system and a large computer used by a bank, for example, is a multi-user computer. Multi-user computers need to be able to handle a large number of simultaneous logged in users. Users can either be connected via a dumb terminal or by a PC. A dumb terminal allows the user to view a display and to request actions via some form of interface that is usually a keyboard, but which does not perform any processing. The important point is that in a multi-user computer system the processing is done on the shared CPU.

Figure: 3.14

Single User System

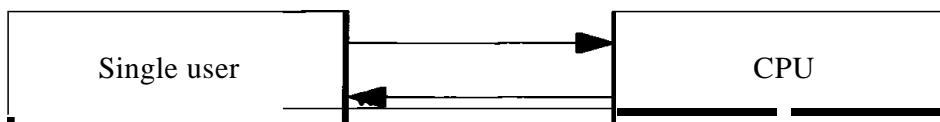
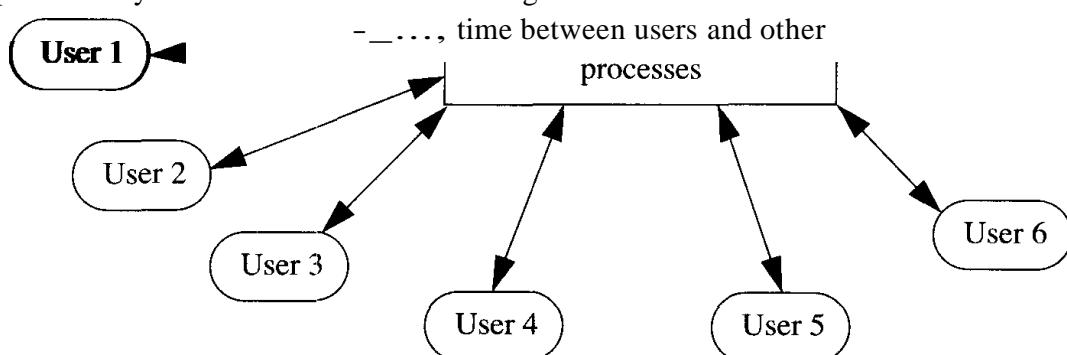


Figure: 3.15

Multiple User System



A single user or multi-user computer system often allows each user to execute more than one program at anyone time. This capability is referred to as 'multi-tasking'. For example, on a PC, you can print very long documents and continue to work on a spreadsheet at the same time.

© IBO 2004 3.3.4 COMPARISON AND DIFFERENT APPLICATIONS OF COMPUTER SYSTEMS

In the following section a comparison is made between the different computer systems. When comparing computer systems you should keep in mind these characteristics: size of primary store (RAM), backing store size (online disk and tape capacity), range of 110 devices used, physical size and cost, type of operation (Le. single or multi-user) and processor word length and speed.

Personal and Portable Computers

This range of computers is designed to support the individual either in the home or at work. A typical home or work PC is used to enable users to create, store, retrieve and print word processed documents and to connect to the Internet to enable access to the world wide web via a browser and to access email. However PCs are very powerful and can also perform complex financial calculations and graphical applications.

Mainframe

The mainframe is much more costly than the PC and is able to handle a number of simultaneous users. The size of the RAM, cache and disk capacity is also significantly greater.

Mainframes are typically used by banks, large government departments and insurance companies. These uses are focused around handling the enormous volume of transactions conducted on a single or distributed customer accounts database.

Super Computer

The super computer is vastly more expensive than either the mainframe or PC. It has a faster processor configuration often using 100s of PCs in parallel. It has more RAM and cache and can access very quickly a vast amount of data from the secondary or backing store.

Super computers are used to run single computer models of things such as a model used to predict the next day's weather or the impact of global warming.

EXERCISE 3.8

1. Define the terms single user and multi user in terms of computer systems.
2. What advantage does multi-tasking offer users?
3. Using your school as a resource, answer the following. Draw up a list of all the different types of computers used in the school. For each type state the following: primary store size (RAM), backing store capacity, I/O devices used, size of device, cost, processor details (word size, speed etc.), operating system used, multi-user or single user and brief list of uses made of the computer.
4. Select an application run by the most expensive computer in the school and compare its operation with the computer you normally use at school or at home. In your comparison, make reference to the points noted in the above question in relation to the main purpose of the computer.
5. Using the resources available to you e.g. local bank, newspapers or Internet, locate and describe, in terms of the points listed in question 3, an application of a mainframe and super computer.
6. Compare the operation of the mainframe and super computer from 5.
7. Compare the operating features of your school or home PC with that of either the mainframe or super computer you described in 5.



© IBO 2004 **3.3.5 COMPUTER SYSTEM OPERATION MODES**

Computer systems operation is controlled by the operating system and the desired mode of operation. We have already seen that operating systems can be either single or multi-user with the added capability of allowing a user to multi-task. There are number of possible modes of operation. These are now described.

- (a) Real Time Processing. Small computer systems that control the operation of VCRs and medical equipment such as heart monitors operate in real time without intervention by humans. These devices are embedded into a range of equipment and have all the basic characteristics of larger computer systems, however, they are pre-programmed to act on inputs they receive without the need to alert a human operator. The time between the initial input and the subsequent action is thus very much reduced.
- (b) Interactive On-line. This is a very common mode of operation. Most computer databases used to record customer booking details in hotels, hospitals, airlines and theatres need to ensure that it is not possible to double book or record the same room or seat as being allocated to different people at the same time. The operator of the computer system can interact with the booking program and does this by connecting directly. Thus the terms 'interactive' and 'online processing' are used to describe this mode of operation.
- (c) Batch processing. When a computer system is operated in batch mode there is a time gap between data collection and data processing. Batch mode is used to perform a set of processing steps on a set of data that has been collected over a period of time.

© IBO 2004 **3.6 APPLICATIONS OF DIFFERENT MODES OF OPERATION**

Real time systems are usually found in embedded chip technology that controls things such as industrial processes, medical equipment, automotive engines and household appliances. The major issue is that the computer system is required to operate without human intervention.

Online interactive computer systems normally operate where it is important that the human operators or users have direct access to the functions of the computer system. In a bank, data is updated immediately e.g. customer bank balances are checked before a withdrawal and updated immediately afterwards. Further examples of online systems include bank ATMs, hotel and airline reservation systems, database systems as used by large companies to process customer orders.

Batch operation is used by a range of computer systems. For example, a school enrolment system may require students to hand in an enrolment form. These forms would be collected at the end of the day and entered into the computer database at the one time. Payroll calculations and cheque productions are often performed in batch mode. Such a set of calculations usually requires a lot of computer time. Hence an operator could collect the data e.g. hours worked and rate of pay for each employee and store this in a file and then set the payroll calculation process to start at midnight by reading the file sequentially and working through each employee.

EXERCISE 3.9

1. Outline the principle characteristics of real time, online and batch processing.
2. Use the resources available to you and locate a range of computer applications and classify them as real-time, online or batch.

- 3.** For each of the following applications give reasons why a specific mode of operation is likely to be more appropriate than another:

Payroll and bank cheque or direct employee bank account update processing.

Connection of a heart monitoring device to a patient.

Car computerised fuel injection system.

Library book borrowing system.

Hotel reservation system.

Pollution monitoring device and data transmission.

- 4.** Connect each of the types of data processing to the correct system and example:

| | | |
|-----------|--|---|
| Real-time | <p>In this type of processing the data is collected first and then put into the computer to be processed in one go. Often, in this type of system, the data is checked for accuracy before it is entered into the system.</p> | <p>An electricity company needs to send out bills to its customers. To do this it collects a batch of meter readings together and enters the data into the system. The system then checks the customer records to see what the previous meter reading was, calculates the amount due and prints the bills. Once the data has been input there is normally no further need for operator intervention. This type of job can be programmed to run at night when there is not so much use of the computer system.</p> |
| On-line | <p>In this type of data processing system the computer is updated so that the information or data input is processed before another input can be made. The system must be fast enough to ensure that the data is processed immediately.</p> | <p>A computer system is being used to control the flow of a drug to a hospital patient, the drug is designed to regulate the heart rate. If the patient's heart stops, as well as audible and visible alarms, a stimulant is immediately supplied to the patient.</p> |
| Batch | <p>In this type of data processing there is a permanent connection between the operator terminal and the computer system. Usually this is because the operator needs constant access to the stored data to view or update it.</p> | <p>A certain company sells music CDs where the customer orders over the telephone. When the customer first applies they are given a client number so that the operator can identify their records (payment records for example). When they place an order, the operator checks the stock records in the system to see if that CD is available. If it is, they confirm that the order will be shipped immediately to the customer.</p> |



©IB03.3.7 RELATIONSHIP BETWEEN MASTER FILE AND TRANSACTION FILE 2004

A master file contains the main data for a computer system or application.

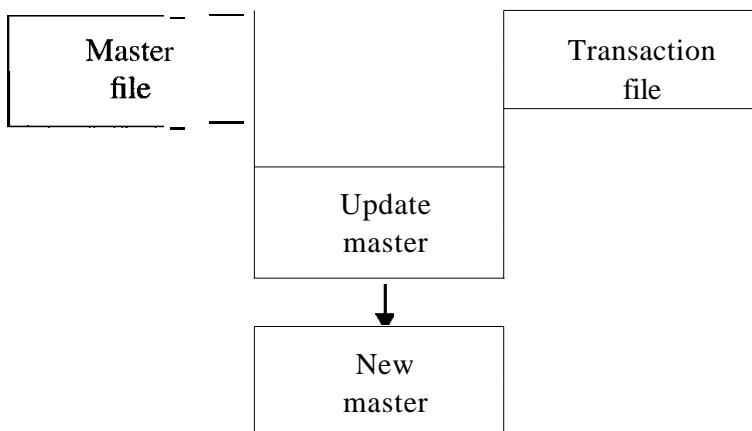
A transaction file typically holds a list of the changes that need to be made to the data held in the master file. A change is referred to as a transaction.

In an online system the transactions e.g. a bank withdrawal are used to update the master file directly as the transaction occurs.

In a batch processing system the transactions could be collected and stored in a separate file called a transaction file. At an appropriate time the records in the transaction file could be read sequentially and used to update the master file. An important feature is that the transaction and master files are both sorted so that the process can be completed in a single pass.

The relationship between the master file and transaction file is shown diagrammatically in figure 3.14, which is shown below. In the diagram the update process is run as a batch process. The data records are read from the transaction file one at a time and the master file record that matches the transaction record is then accessed and updated. This sequence is repeated until the last record in the transaction file has been read.

Figure: 3.16



The key difference is that, in the batch processing method of operation, the data in the master file is not necessarily up to date i.e. the transactions have been recorded and stored in a transaction file. In an online system, the transactions are applied to the master file as the transactions occur or are recorded.

EXERCISE 3.10

1. Define the terms 'master file' and 'transaction file'.
2. Explain the use of a master file and transaction file in relation to the payroll example given in the previous set of questions. You may use a diagram to help explain your answer.
3. Construct an algorithm to describe the major steps involved in the above processing.
4. Explain why the algorithm works more efficiently with sorted files.
5. Explain the link between sorted files and sequential access methods (e.g. holding the files on tape).



© IBO 2004 3.3.8 RELIABILITY OF SYSTEMS

The reliability of any system is only as good as the data that is entered. The correctness of data in a system may be described as the 'integrity' of the data. As systems become more complex and autonomous (making key decisions on software-based rules) then mistakes can happen, with serious consequences.

An X-ray machine that is designed to emit X-rays at a given power to be used in a cancer treatment process can be computer controlled. If, due to a data entry error the dose of radiation is entered wrongly, a patient could be killed or, at least, badly burned.

Closer to home, students would probably like to be sure that their grades are correctly entered into whatever computer system the school is using. These grades may be used to write reports, which, in turn, may be used to write university and college references. Thus an incorrect grade could have a direct impact on the future of a student.

When the systems are actually in place and running, the consequences of hardware or other system failures has also to be taken into account. It is not too serious if a bank mainframe computer goes offline for a short time, the transactions can be recorded on disc or even on paper and entered into master files later. However, as mentioned in 3.3.5, systems like air traffic control or the monitoring of a nuclear power plant cannot fail and therefore two complete systems operate in parallel in case one fails. In extreme cases, a triple system may even be used.

Data that is in systems needs to be protected against accidental or deliberate loss or damage. Some examples of threats to data are:

- Unauthorised users (hackers) may gain access and alter or remove data.
- Physical media (discs, tapes) may be stolen.
- The hardware may be stolen (along with fixed backing store).
- There may be fire or flood damage.

Additionally there may be threats to the security of personal and company data - this is also often referred to as 'data protection'. In this case, particular dangers are that data may be copied, leaving no evidence that a copy was made or that data may be accessed over networks remotely.

An important method of protecting data from unauthorised access is the use of passwords and privileges, particularly on networked systems. Only people who have been given a logon name and allowed to select their password can access a system or parts of a system. Privileges can be assigned to users; low-level users can access data but not change it or make a copy to a local floppy disc, for example. In a supermarket you may see this happen if a price has been incorrectly recorded in the computer database. The check-out person usually does not have the privilege to change prices and will therefore call a supervisor who will tap in an access code and correct the situation.

To be effective, passwords must be of a reasonable length (usually 6 characters or more) and hard to guess (not your partner's/child's/dog's/parakeet's name) and containing special symbols besides alphabetic characters.

If, while you're at the supermarket, you look around carefully you may see examples of physical security - locking computer systems in special rooms accessible only by personal equipped with key cards or special codes.

When data is transmitted over networks it may be encrypted if especially sensitive (large financial

transfers, police data on suspects, government military secrets) so that it is very difficult (not impossible) to decode. Encrypting data helps to ensure that, even if data is accessed, it is not readable. A PIN number on the magnetic strip of a bank card is encrypted for this reason.

Methods used to detect and correct errors are discussed in section 3.6.

BACKUP STRATEGIES

To protect data against irreversible damage, backup copies are kept in a safe place; in most cases they will be kept in a different building. These can be used to restore a system to the state existing at the last backup. When did you last backup your data? Most businesses cannot function without the data from their computer systems; therefore they must have a thorough and thoroughly tested backup strategy.

EXERCISE 3.11

1. In weather forecasting, data is collected for input into weather models. Discuss the implications of mistakes being made in collection or transmission of the raw data.
2. A hospital has medical records on paper and is transferring them into a new computer system. Discuss the importance of transferring these records correctly.
3. Compare the importance of the reliability of systems that monitor patients in intensive care with those that carry out stock control tasks.



© IBO 2004 **3.4 NETWORKED COMPUTER SYSTEMS**

© IBO 2004 **3.4.1 DEFINITION OF THE TERMS LAN. WAN. CLIENT AND SERVER**

Networks allow computers and peripheral devices to communicate. A network connection is usually provided by means of a connection device, e.g. a network card, and a facility to enable transmission, e.g. cable or microwave. Transmission is facilitated by the use of a send and receive protocol that is understood by the communicating devices.

The term 'server' refers to the software and computer that provides the services that are available via the network. For example, a file server provides the ability to store and distribute files to users on the network. An email server would manage the flow of email in and out of the network, check that an email address is valid and allow users to access their email.

The term 'client' refers to the software application that requests actions from a server. For example, to use email a user needs to run the email client that allows the user to access their email on the email server. The email client will allow the user to manage their email, e.g. create and send a new piece of email. However, without the email server the email will not actually be sent.

There are essentially two forms of network: Local area network and wide area network.

These two terms are now defined.

LOCAL AREA NETWORK (LAN)

A local area network is a term used to refer to a network of computers and peripherals that are directly linked via cable or microwave transmission within a single area. Typically the area is that of a building or office. Most schools operate LANs. A LAN is typically made up of a central server computer that stores the shared application software and data. Individual PCs and/or workstations and dumb terminals are then linked to the server via a network card and transmission medium. The most common transmission medium is some form of cable.

The advantage offered by a LAN is the ability to share peripheral devices such as printers. Data from shared files can also be accessed. Application software can also be loaded from the file server onto a PC. The normal mode of operation is to run the required software from the PC's hard disk to avoid congestion. In the school it means, for example, that a student can access their data files and email from any connected workstation. This increases flexibility.

The most common mode of operation is that of client/server. The server is the central node in the network and coordinates and supplies services to the clients in the network. To enable this type of operation the server runs a network operating system that enables clients to login and use the facilities on the network. A person wishing to access the LAN needs to first login using a PC or workstation that has loaded the client software. The client software provides the link to the network operating system and allows the user to then access the network services e.g. shared printing.

LANs can connect to other LANs and to wide area networks. This is done by using a device commonly referred to as a GATEWAY. Gateways allow access to services such as the Internet or to services available on another LAN.

WIDER AREA NETWORK (WAN)

Wide area networks allow computing devices to connect to a networked computer facility from remote location. The Internet is an example of a WAN. Many large organisations that are located over a wide geographical area also utilize WANs. The diagram below shows that a WAN can be made from a range of different computer facilities and utilises a range of communication media.

The main features and differences between a LAN and a WAN are shown in figure 2.15 below. A LAN allows local connection and sharing of resources within a confined area e.g. an office block, whereas a WAN allows wide spread connection between users over a much larger geographic area. Many LANs are themselves connected to WANs and allow local users to operate as if they were on a private traditional LAN, but to also access the facilities of a WAN.

Figure: 3.17

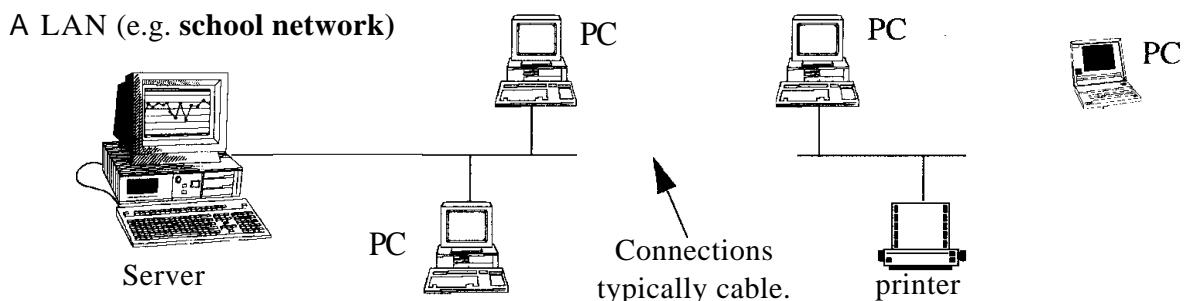
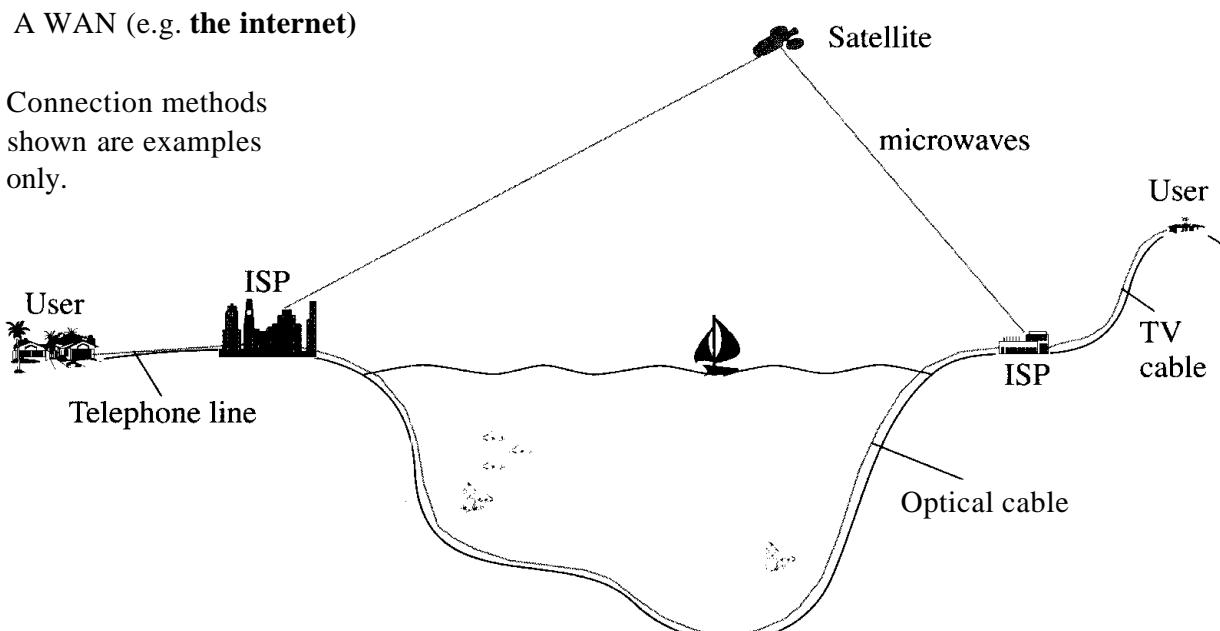


Figure 2.16



EXERCISE 3.12

1. Define the term client.
2. Define the term server.
3. Define the term LAN.
4. Define the term WAN.

5. Using your school as a resource describe the purpose of the school's LAN.
6. If your school is connected to a WAN, describe how this is done and the advantages this connection provides to the users of the school's network.

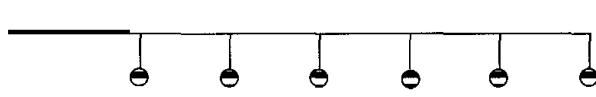


©IBO 2004 3.4.2 DIFFERENT NETWORK TOPOLOGIES

Networks can be structured or configured in a variety of ways. A specific network arrangement is said to be the network's topology. There are basically four main topologies as shown in figure 3.16. Each of these is now briefly discussed.

Figure: 3.18 . Network topologies

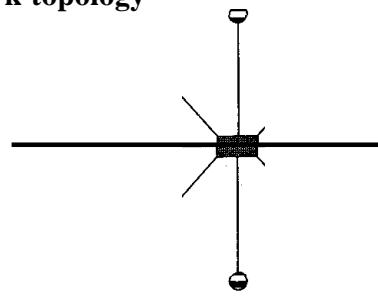
Bus network topology



Key

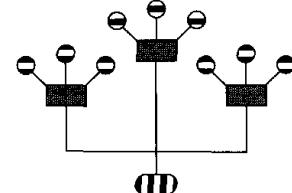
| | |
|------------|------|
| ○ | Node |
| — Hub | |
| ○ Computer | |

Star network topology

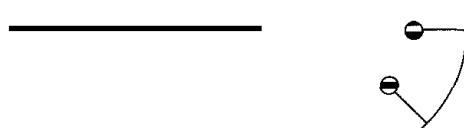


Tree network topology

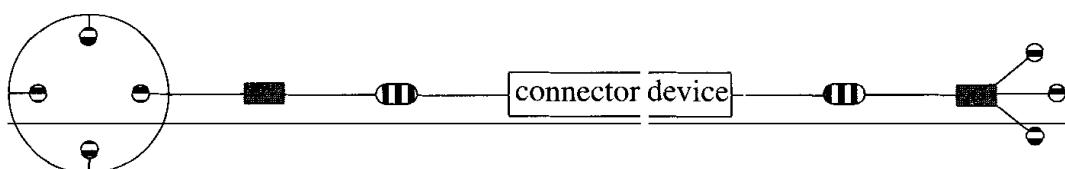
(Hierarchical)



Ring network topology



Hybrid



Star:

a star network requires each node of a network to be connected to the central file server or host computer via a single cable. To enable the connection to take place a piece of equipment called a 'hub' is used. Each cable leading from a node is plugged into the hub and a single cable is used to link the hub to the file server node. Thus each node has its own direct connection but congestion can occur between the hub and the file server. The file server shares out its time between the nodes that demand service.

Bus:

this is a very simple topology where a single cable is used to link all the nodes. The bus cable is therefore shared by all the nodes and can become congested. Data is transmitted with a node number and each node takes

Computer Fundamentals

from the data bus data that belongs to it or passes the data on. Bus cables must be terminated and do not return signals to the server.

The following types are not mentioned in the subject guide, but are included for the sake of completeness.

Ring: this is used to link computers of equal importance. For example a bank may use four mainframe computers. To enable these to talk to one another, the four could be linked by a common cable in a circle. In such a setup each computer can perform processing and can also share the resources of the other computers. Such a setup allows decentralised processing.

Hierarchical: in a hierarchical topology one main computer is said to be the 'main computer' and there can be other computers linked to this computer, which in turn can be host to other smaller networks. Such a setup allows a centralised approach but allows different sections of an organisation to have their own network facility.

Terminal network: provides for centralised control of processing and access but does not allow for any processing to take place at the user's end.

Peer-to-peer: allows for nodes to act as both servers and clients.

Client/server: provides for the centralised control of the network via a single main computer. Thus only one computer needs to be able to perform the tasks of a server. Client server setups also transfer much of the processing to the server in much the same way as traditional multi-user operating systems operated for dedicated terminal users. A database client, on receiving a request to list all records located in a particular town, would return the required sub-list from the entire database. Traditional file servers would have sent the entire file and the processing would then be done on the user's PC.

We can also link different networks such as a star or ring into a single hybrid network.

©IBO 3.4 .3 HARDWARE REQUIRED IN NETWORKING

Network Card: allows a device to be connected to a network. Can be an interface card that is connected via a communications port or be an integrated part of the device. With the advent of wireless LAN technology, the connection need not be via a physical cable.

Cable: typically networks are implemented by the use of some form of cable. However, the advent of wireless technology is enabling networks to be implemented that require far less use of physical transmission media.

Hub: there many types of hubs. The main function of a hub is to allow different sections of a network to be connected. It is common in a network to split a communications channel into several smaller parts. Thus server devices may in fact share a single channel. The device that connects the different segments and passes the data onto the appropriate channel is termed a Hub. A simple hub will simply pass on data packets from the one set of input channels onto the entire set of output channels. A switched hub will pass the data packet onto the appropriate destination channel only.

- Router: a router can be used to direct LAN traffic from one LAN or part of a LAN to another. A router is able to identify the proper destination of data, unlike a hub.
- Switch: A switch is used in hybrid networks to connect the different segments and pass packets of data between them (this is explained in more detail in section 3.4.4).
- Gateway: a device that is used to connect users of a LAN to another network, which uses different protocols. The best example is to consider how a school LAN is able to allow a user (a student or teacher) to connect to the Internet. This is done via the use of a gateway device that allows users to connect to the Internet via the ISP (Internet Service Provider).

© IBO 2004 **3.4.4 PACKETS, PROTOCOLS, INTEGRITY AND SECURITY OF DATA**

When discussing LANs and WANs we distinguished them on the basis of distance. It is also true that different technologies are usually used: LANs typically use **broadcast** techniques where every computer listens on a common cable, whereas WANs use **switching** techniques since direct connections are not practical over large distances.

There are two main types of switching network (circuit switched and packet switched) of which, only packet switching is mentioned in the subject guide.

In a packet-switching network, as you probably guessed, data is sent in small, discrete chunks called 'packets'.

A **packet** typically contains:

- information about its origin.
- information about its destination.
- information about where in the sequence of packets it belongs.
- information about how long it has been travelling.

Because packets are transmitted by different computer systems outside the control of both the sender and receiver of the data, an internationally agreed set of rules is needed, known as **standard protocols**.

PROTOCOLS

To enable two devices to exchange digital signals it is important that both understand what is being sent and received. This is done by adopting a set of rules known as a protocol - an agreed-upon format for transmitting data between two devices. The protocol determines the following:

- The type of error checking to be used.
Data compression method, if any.
- How the sending device will indicate that it has finished sending a message.
- How the receiving device will indicate that it has received a message.

There are a variety of standard protocols from which programmers can choose.

Computer Fundamentals

Each has particular advantages and disadvantages. For example, some are simpler than others, others are more reliable, and yet others are faster.

From a user's point of view, the most relevant aspect about protocols is that the computer or device must support the right ones if it is to communicate with other computers.

Data integrity is concerned with making sure that what is received is what was transmitted; data security is concerned with preventing unauthorised access to network data, both of these topics are discussed further in sections 3.4.7 and 3.4.8 respectively.

© IBO 2004 **3.4.5 SOFTWARE INVOLVED IN NETWORKING**

In order for a PC to connect to the Internet from home it is necessary to have loaded system software that enables the PC to receive and transmit data via the modem, which is in turn connected to a phone line.

In order for a PC to connect to a LAN it needs to have the appropriate communications client software loaded.

Communications software deals with protocols and data security, both for LANs and WANs and handles the need for both integrity and security of data as previously discussed.

© IBO 2004 **3.4.6 DATA INTEGRITY IN TRANSMISSION**

This is generally known as 'noise'.

Data that is transmitted over communication lines is also subject to interference which can alter the nature of the data represented. Parity checking is used to check on such errors and, if an error is detected the network, will try to recover the data, often by requesting a resend of the data packets.

PARITY CHECKING

As mentioned above data, can be altered during transmission either within the computer or between computers. Prevention of such errors is related to the robustness of the design of the computer system and the environment within which the computer is used. However, it is possible to set up methods to detect if an error has occurred after a group of bits have been moved from one location to another. This is done by using a parity check.

Figure 3.8 shows the working of a parity check in diagrammatic form. Let's say we moved a byte 1010 1100 from location A to B and in the process the 2nd bit was flipped ie 1110 1100 was received at B. How could this be picked up. One way is to append a parity bit and check this matches what is expected.

There are two forms: odd and even parity.

In odd parity we append a 1 as the parity bit only if it makes the number of bits set to 1 odd in total.

In even parity we append a 1 as the parity bit only if it makes the number of bits set to 1 even in total.

In our example lets assume we are using even parity. At location A we have 1010 1100.

We note that there is even number of 1s thus we add a parity bit of 0 ie there are now still 4 bits set to 1.

We therefore send 1010 1100 and a parity bit of 0.

When this data is received it has been changed to 1110 1100 and parity bit of 0. We now check that the parity bit makes sense. There are now 5 bits set to 1, thus our parity bit should set to 1 to make 6 bits set to 1 i.e. and even value.

As our re-calculation of the parity indicates an inconsistency we report an error in transmission.

To this point we have concentrated on checking to ensure that the integrity of the data is preserved. We need also to be concerned about the security of data. By this we mean protecting the data against unauthorised access, which might result in a change being made, or by preventing unauthorised interception. We do this by ensuring that users are authenticated to login and access the data via a login name and password and by encrypting data when it is transmitted.

CHECK SUMS (BLOCK CHARACTER CHECKS)

Check sums are sums produced from set of binary data by the application of an algorithm that is applied to the bits in the binary data.

In a block character check (one type of check sum), successive bytes are added together and the sum of these is transmitted. A fixed number of bytes would be followed by the block character check. It could be necessary to truncate the BCC since it might exceed one byte of storage.

EXAMPLE:

| | | | | | | |
|---------------|----|-----|-----|-----|-----|-----|
| Decimal value | 72 | 101 | 108 | 108 | 134 | 111 |
| Character | H | e | 1 | 1 | BCC | 0 |

↑
 BCC every
 4 bytes

In this example, a maximum value of 255 can be held in 1 byte.

The sum is used to check that the binary data matches what is expected. The format of the check sum is part of the protocol. Check sums are used to check network data transfer i.e. a check sum may be included with each data packet and, on receipt, the check sum is recalculated and checked against the transmitted check sum to detect errors. If there is an error, re-transmission is usually requested.

Check sums can also be applied to other forms of binary data such as graphics files or other digital images such as finger prints or music files. In the case of the finger print, the check sum would be unique. An algorithm is applied to the bits that make up the file and appended to the data bits. This check sum can be checked after the file is copied or transmitted. For example, if a virus had been incorporated into the file the check sum would be wrong - unless of course the virus was clever!

© IBO 3.4.7 DATA SECURITY 2004

As mentioned above, data security is concerned with preventing unauthorised access and students are expected to be able to explain the difference between security and integrity of data.

Data stored on a network is vulnerable since it is potentially open to view by any computer connected to the network, however far away it might be.

Computer Fundamentals

User Login

In your school you probably access a network and/or use the Internet at home. In both cases you will normally be required to enter a valid user name and valid password. The password is the main form of network security. Passwords should be set sensibly and changed at reasonable time intervals. Passwords should not be easily associated with the user e.g. name, initials, date of birth, name of suburb, pet's name or parent's name etc. The password should be a random collection of characters including letters and digits. A copy should be stored in a safe place and not given to other people.

Passwords are stored on the computer but are stored in encrypted format so that if they are located they cannot be read.

Data encryption

The other major security problem is the interception of data transmissions by an unauthorised third party. Data is typically encrypted when transmitted. This means that the data is scrambled at transmit time i.e. encrypted and then de-encrypted when it is received. If the data transmission is intercepted it cannot be de-encrypted without the use of an encryption key.

Permissions

As well as passwords to enable users to login, each user also has a set of permissions associated with their logon name or group. Some users, administrators or super-users, can look in any data file, change user passwords and delete any file on the system. Ordinary users can only access their own files and use specified resources such as printers and CD burners.

© IBO 3.4.8 THE NEED FOR SPEED IN DATA TRANSMISSION 2004

As you probably appreciate, the internet and other networks can be busy. Therefore, we want to transmit data quickly and efficiently. To see how data transmission speed can be improved, we briefly examine the compression of graphical data and the common formats: JPEG and BMP.

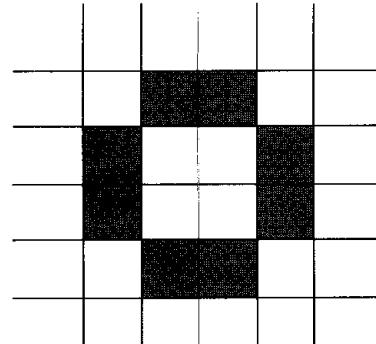
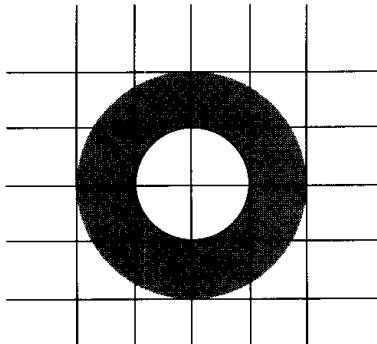
'TRANSMITTING GRAPHICAL DATA

As we saw earlier, data is split into packets and transferred across the internet or other networks. Each packet typically can find its own route through the network (sometimes packets are all sent on the same route - known as a virtual circuit) - strictly speaking, this type of packet is known as a datagram.

The sending of information with all the attached data is obviously time-consuming since they have to be assembled, routed then dis-assembled at the destination. If the original file can be compressed in some way then the speed of transmission (of the whole file) can be improved. This is especially applicable to media files such as graphics, sound and movies (films). The IE Subject guide specifically mentions the 13MP and JPEG graphical formats.

Bitmaps (BMP)

A photograph or oil painting is an analog representation (section 3.5.1) and, as such, can't be stored in the computer as a set of 1's and 0's. To make this conversion, the original picture is sampled, effectively represented by a series of rectangular picture elements or pixels:



Each pixel can only be used to represent an individual colour, it can't be half one colour and half another. As with all conversion from analog to digital, some approximation or inaccuracy is introduced by the conversion process. Clearly, the finer the grid of pixels, the more accurately the original can be reproduced.

Each pixel is given a unique code corresponding to a particular colour, a black and white image could be represented by a 0 and a 1 (see section 3.5.1). For three colours, 2 bits would be required and so on.



Thus any picture can be reduced to a set of binary number codes:

| | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 001 | 001 | 000 | 001 | 001 | 001 | 001 | 001 | 001 | 001 |
| 001 | 000 | 000 | 000 | 111 | III | 111 | 001 | 001 | 001 |
| 001 | 001 | 001 | 001 | III | III | III | 001 | 001 | 001 |
| 001 | 001 | 001 | 001 | 111 | III | III | 001 | 001 | 001 |
| 010 | 010 | 010 | 010 | III | 110 | 111 | 010 | 010 | 010 |
| 010 | 010 | 010 | 010 | 010 | 110 | 010 | 010 | 010 | 010 |
| 010 | 010 | 010 | 010 | 010 | 110 | 010 | 010 | 010 | 010 |
| 010 | 010 | 010 | 010 | 010 | 110 | 010 | 010 | 010 | 010 |
| 101 | 101 | 101 | 101 | 101 | 110 | 101 | 101 | 101 | 101 |
| 101 | 101 | 101 | 101 | 101 | 101 | 101 | 101 | 101 | 101 |

| code | colour |
|------|---------------|
| 001 | sky blue |
| 000 | white |
| 111 | green |
| 110 | brown |
| 010 | dark blue |
| 101 | bright yellow |

This is a bitmap. In order to transmit this picture, without further loss of detail, every binary code must be transmitted.

Compression of Bitmaps (JPEG)

The requirement to transmit every bit does not mean that we have to take every grid cell (pixel) and transmit its binary code. If we look more closely at the '*picture*' in the last section we will notice that certain blocks of the same colour occur together. We could describe a block by two numbers, for example:

A three digit colour code

A number representing how many cells of that colour occur (including any 'wrap-around' from the last column)

Thus the first part of the picture could be: 001 2000 1001 8, or if we prefer to put it all in binary (with an extra 0 to convert 3 colour codes to 4-colour codes):

0001 0010 0000 0001 0001 1000

or in three bytes:

00010010 00000001 00011000

EXERCISE 3.13

1. How many bytes of data are transmitted with compression and without?
2. What are the limitations of using 4 bits for the run number? How can they be overcome?
3. Is there any particular type of data that this method would be good for or bad for in terms of the compressed file size?
4. Convert the picture to a series of hexadecimal digits.



The above technique (run-length encoding, used in GIF files) is said to be 'lossless' since all the original data is preserved and the image can be restored completely. As you can probably see, the technique would not achieve much compression of photographs or oil paintings.

JPEG compression works by assigning very similar colours the same value, perhaps a sky scene has a lot of very similar shades of blue. Maybe few people will notice if you assign them all the same colour code. Part of the original information is then lost by this conversion (thus it is known as 'lossy' compression).

VECTOR GRAPHICS

The Subject guide doesn't mention these, but we'll include them for the sake of completeness.

Vector graphics are the things we use in our good old drawing programs like Fireworks and Corel Draw. Even Word has vector graphics. Each graphic, like a circle can be described by a mathematical equation - x , y coordinates and radius ought to be enough to describe a circle. Other attributes like line thickness, colour, fill pattern, line style etc. can also be stored as numerical information.

One great advantage of this method is that objects can be scaled up without loss, unlike bitmaps. Another is that vector graphics files are generally smaller in size than their bitmap counterparts.

A disadvantage of vector graphics is that they can't be used for complex images like photographs.

There is no generally agreed standard for the transmission of vector graphics to internet browsers although, if you have a product such as Macromedia Flash Player installed, you can download and play animated vector graphics image files.

© IBO 2004 **3.4.9 DISCUSS APPLICATIONS AND IMPLICATIONS OF NETWORKING FOR ORGANISATION**

LANs and WANs provide a range of productivity applications. The above example has introduced some of these. In general terms the range of productivity applications include the following.

Improved Internal communications:

The ability to utilize email and messaging systems allows employees of organisations to communicate without need to always rely on person to person contact via either conversation or leaving notes.

Email provides the ability to send messages to individuals or group of individuals. Larger documents can also be attached and distributed using mailing lists. This reduces the need for photocopying and saves time and effort.

External communications

By connecting to a WAN, employees can email other employees in geographically dispersed locations. To the employees concerned it appears as if they are all in the same office.

Conferencing

As bandwidth improves the ability to communicate using video conferencing will make it possible to hold face to face meetings without the need to be physically present.

Distributed processing

Many applications are required to share data and this has been traditionally been done by sharing access to a centralised database i.e. one file system. However it is possible to set up local files that also have the ability to act as one centralised file system. The users see no difference.

An organisation may also wish to break up the work over a number of decentralised processing centres.

The implications of networking in an organisation revolve around the need to ensure security of access and to ensure that employees' work practices make sensible use of the network without the organisation resorting to invasive monitoring processes which have the potential to raise privacy issues related to employees' rights.

BENEFITS OF NETWORKING FOR USERS

Networks provide a number of benefits for users. In summary they are:

Access to a variety of shared internal resources, e.g. printing, thus reducing the cost because expensive peripheral equipment such as colour laser printers need not be duplicated but simply connected to the LAN.

- Access to shared programs, e.g. an order entry or invoice system.
- Access to shared data.

Users can store their personal private data centrally and can therefore access it from any device. If the network access is available they can access the data from external sources e.g. from home or whilst away from the office.

- Access to external computer systems e.g. world wide web and email via the Internet. The access to email is very convenient and therefore improves communications and helps the organisation achieve its objectives.

A comparison of the general benefits and limitations is provided in the table below.

| LAN | WAN |
|---|--|
| Lower cost to set up, suitable for small business/organizations. | More expensive hardware required, more suitable for large a organization operating over a large area (e.g. a country). |
| Limited range of data transfer, expense rises rapidly because of fixed cabling costs (your locality). | Security depends on communications system. Telephone system is cheap and insecure, dedicated lines are very expensive, satellite links exorbitant. |
| No external communications system required. | Data processing can be centralised, avoiding the need for more than one mainframe installation with attendant maintenance costs. |

The essential limitation of a LAN is that unless it is connected to a WAN you will not be able to access the information sources available via technologies such as the world wide web or world

wide email. Communications will be restricted to traditional forms such as the phone. But, a LAN is secure from external access and it provides a reasonably cost effective way to enable data and resources to be shared.

A WAN provides access to external information sources and to the growing world of electronic communications. But this comes at a cost to the organisation in terms of additional infrastructure and access costs. A WAN also implies that the organisation needs to be aware that security is an issue, either from people attempting to access the organisations computer system from outside or by intercepting transmission from the organisation to the WAN.

EXAMPLE:

An organisation called POTS sells cooking ware. The organisation currently operates a LAN at its headquarters in Hong Kong and is considering allowing the world wide set of sales representatives to be able to access an online catalogue and sales administration system. It is proposed to enable the current catalogue and sales administration system to be accessible via a secure gateway via the Internet. The sales staff who are not direct employees of the company but who act as agents will have a login and password that will allow them to access only that part of the system they need. Additionally, the sales agents will be able to communicate via email to the various service representatives located in Hong Kong instead of by letter or phone.

What are the benefits and limitations of such a proposal?

The benefits relate to the increased level of communication that will be possible between the agents and the service employees. From this you would expect problems to be resolved more quickly, which in turn should mean more satisfied customers and therefore increased business. The fact that the sales system can be accessed online should also streamline operations and may lead to cost savings.

The limitations or disadvantages of the proposal relate to the costs associated with the provision of the new online service. Costs will be incurred by the company to pay for an Internet connection, new software, increased security and possible additional staff or extensive training of existing staff to administer the new system. Security will need to be monitored to ensure that no illegal access occurs to the sales system.

Organisations have also found that providing online Internet access and email access to employees has resulted in employees using the access for an increasing amount of non-work related activities. This in turn has lead many employers to consider increasing electronic surveillance to deter this type of activity.

EXERCISE 3.13

1. Define the term 'network topology'.
2. Explain what is meant by describing a network as having a 'star topology'
3. Explain what is meant by describing a network as having a 'ring topology'
4. Explain what is meant by describing a network as having a 'bus topology'
5. Explain what is meant by a hybrid network topology and list the advantages that such a topology might provide
6. Explain the functions of the following hardware associated with networks: hub, node and router.

Computer Fundamentals

7. Outline the characteristics of the following transmission media: cable, microwave and fibre optics.
8. What is meant by the teon network protocol?
9. Why are passwords a vital security measure on a network?
10. Why is data encryption used in data transmission?
11. How is security typically handled on a network?
12. Explain why communication software is required.
13. Describe the network topology used in your school and the way the network functions.
14. Suggest an alternative topology for your school's network and discuss the advantages and/or disadvantages of your proposal.
15. Student Activity. Encryption is a key issue for Computer Science. Use the Internet or some other appropriate source to research the area of data encryption. During your research try to uncover two or three significant areas of computer science research related to the area. (Hint: network security and encryption can often provide fertile ground upon which to write the extended essay)
16. In terms of your school, summarise the benefits the students and staff gain from having a network.
17. Discuss the difference in the benefits derived by the use of the network by the school's administrative staff as compared to the students.
18. In general tenus what advantages does email offer the organisation?
19. Define the tenu 'distributed processing' and explain why this type of processing may provide advantages to an organisation.
20. In general tenus compare: the benefits of a WAN with a LAN
21. Consider your school and its connection to the Internet, which is itself a reasonably large WAN! What advantages does the school receive by being connected to the WAN and who gains most?
22. What limitations or disadvantages are associated with the school being connected to the WAN?
23. Activity: Using your local area or other resources locate, an organisation that uses a LAN and/or a WAN and discuss the benefits and limitations and implications of the use of the network.
24. Investigate some aspects of the ftp http and tcp/ip protocols. List 4 items which are part of a protocol.
25. Write a Java program, with a method, that takes a string message and adds BCCs every 4 characters. Provide another method to restore the original string and verify that it is correct. This is an exercise that can be extended as further programming skills are developed. For example, a message can be stored in a text file and a 'filter' method used to introduce random errors that student's programs must find.



**© IEO 3.4.10 WEB BROWSERS AND SEARCH ENGINES
2004****Functions of a web browser.**

A web browser such as Explorer or Netscape provides a range of functions. These include:

Ability to access hypertext documents by using the Universal Resource Locator address.

Ability to scroll up and down the desired document.

Ability to store a history of sites visited and to be able to move forward and back through this history. You can also return to home page and to refresh a page i.e. reload the page.

Ability to print desired pages.

Ability to configure the browser. There are a number of options that can be set. For instance you can ask the browser to cache the most recently accessed pages. This speeds up the operation of moving back through your history. You can also set your desired home web address.

You can display the HTML code.

You can save the page onto your local computer.

You can bookmark your most popular pages.

You can configure various security options, including encryption.

FUNCTIONS OF A SEARCH ENGINE

The world wide web functions because each HTML page can be catalogued by a search engine. This can be done in a number of ways. For example by the use of HTML tags to act like keywords or by notifying search engines of your document. Search engines can also open documents and look for key words so that the document can be classified.

Search engines effectively work by searching large indexes using the key words that you enter. Many apply boolean logic to enable searches to be made more efficient.

The basic functions of a search engine can be summarised as:

Looking across the world wide web for new documents.

Cataloging these documents using keywords to build and update the search database.

Provide a query/search facility for users to enter search text, which is then used to search the database and report back to the user the results of the search.

EXERCISE 3.14

1. Using the list of browser functions above summarise the functions provided by the browser that you use to access the world wide web.
2. What security functions does your browser provide?
3. What customising features does your browser provide?
4. Use the Internet to locate a site that reviews the available search engines and the advantages offered by each.



© IBO 2004 **3.5 DATA REPRESENTATION**

© IBO 2004 **3.5.1 BINARY DATA REPRESENTATION**

Computers traditionally represent data using a two-state voltage system. One voltage level represents the first state and this is represented by the digit 0, the other voltage level represents the second state and this is represented by the digit 1. Because there are two states, we refer to this as a binary system. Such a binary system allows the use of sequences of Is and Os to represent data values.

RAM is used to store these binary sequences. It does this by using lower fixed voltage level (V) to represent a 0 and to represent a 1 a small further fixed voltage level increment is added (e). Thus main or primary memory (RAM) requires electrical power to maintain the state of memory. For this reason primary memory is sometimes referred to as volatile memory.



Each 1 or a 0 is referred to as a BIT and these are grouped together into 8 bits to form a BYTE.

Because computers are binary systems the binary or base two number system is used to represent numeric values such as Integers and Real numbers.

The binary number system is referred to as base 2. It therefore uses only two digits 1 and 0.

A simple relationship exists between the number of bits used to represent something and the possible number of different or distinct representations.

With two(2) bits it is possible to represent the following four (4) patterns:

00,01, 10 or 11

With three (3) bits it is possible to represent the following eight (8) patterns:

000,001,010,011,100,101,110,111

The relationship can be stated mathematically as follows.

The number of distinct combinations of n -bits is given by 2^n .

We can confirm this by applying the formula to the example above.

$$n=2 \text{ gives } 2^2 = 4$$

$$n=3 \text{ gives } 2^3 = 8$$

APPLICATION OF THIS RELATIONSHIP

Pixel Colour Intensity

Let us assume that the Red-Blue-Green (RGB) colour represented by a pixel on a screen is calculated by allowing 4 bits for each colour intensity i.e. 4 for red, 4 for blue and 4 for green.

Thus you would be able to represent $2^4 = 16$ levels of colour intensity for each of red, blue and green. This would therefore allow $16 * 16 * 16 = 4096$ different colours, which is not very many.

Memory Address Size

An 8 bit bytes allows $2^8 = 256$ different patterns of Is and Os. Using the relationship of 2^n we can therefore work out how many different memory locations we can have if we use 16 bit, 32 bit and 64 bit memory addresses.

using 16 bit memory we can address 2^{16}

using 32 bit memory we can address 2^{32}

using 64 bit memory we can address 2^{64}

EXERCISE 3.15

1. State the relationship between the number of bits and the number of patterns of 1s and Os that can be represented. What is crucially significant about the relationship?
4. With respect to the colour intensity of a pixel, what impact would there be on the number of different colours possible if the number of bits allocated to each colour was increased to 8 and then to 10? What implication does this have for RAM when storing an image?
- S. In the above example related to memory size, the final calculations were not completed. Complete them now and use suitable approximations and prefixes.



~~©IB03~~ 5:Z NEED FOR STANDARD FORMATS FOR DOCUMENTS AND GRAPHICS

In order that documents and graphics can be interchanged between computer systems it is important that standard formats exist.

Several formats for storing data have been developed for different applications. For example your word processor stores data in a form that other word processors cannot access, unless they have a conversion utility. Because the ability to use different formats in different programs (e.g. photographs in a word processor or graphics in a spreadsheet) is important to users, different software makers have, over the years, agreed on standard formats for exchanging this kind of data. Common examples include:

- gif files for diagrams on web pages.
- jpeg for photographs in word processor documents, graphics programs or web pages.
- wav for sound files in operating systems or web pages.

EXERCISE 3.16

1. Outline why it is necessary to have standard formats for storing documents and graphics files.
2. Assume that a graphics file is displayed using a resolution of 120 pixels wide by 300 pixels high and 8 bits per pixel. Calculate the size of the graphic in terms of bits and then convert this to bytes.

Computer Fundamentals

3. What implication is there for the size of the above graphic if we reduced the width by half?
4. Using the original dimensions of the graphic, what would have to be done to reduce the total size of the graphic by 50% so as to retain the direct proportions of the graphic?
5. What implications, if any, are there if the graphics file used 8 bits to represent colours and we wished to convert it to another format that used 10 bits?
6. What implications are there if we chose to convert a graphics file which used 16 bits to represent colour to a graphics format that used only 6 bits to represent colour?



ASCII is the acronym for the 'American Standard Code for Information Interchange'.

Pronounced ask-ee, ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase M is 77. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Text files stored in ASCII format are sometimes called 'ASCII files'. Text editors and word processors are usually capable of storing data in ASCII format, although ASCII format is not always the default storage format. Most data files, particularly if they contain numeric data, are not stored in ASCII format. Executable programs are never stored in ASCII format.

The standard ASCII character set uses just 7 bits for each character. There are several larger extended ASCII character sets that use 8 bits, which gives them 128 additional characters. The extra characters are used to represent non-English characters, graphics symbols, and mathematical symbols. Several companies and organizations have proposed extensions for these 128 characters. The DOS operating system uses a superset of ASCII called extended ASCII or high ASCII. A more universal standard is the ISO Latin 1 set of characters, which is used by many operating systems, as well as Web browsers.

STANDARD ASCII (ALPHANUMERIC CHARACTERS)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|-----|-----|----|----|----|-----|
| 0 | NUL | SOH | STX | ETX | Ear | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 2 | SP | | # | \$ | % | & | | | | * | + | | | | | |
| 3 | 0 | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | < | | > | ? |
| 4 | @ | A | B | C | D | E | F | G | H | | J | K | L | M | N | 0 |
| 5 | P | Q | R | S | T | U | V | W | X | Y | Z | | \ | | ^ | |
| 6 | a | b | c | d | e | f | g | h | | | k | | ii | n | o | |
| 7 | P | q | r | s | | u | v | w | x | y | z | { | | | | DEL |

More detailed descriptions of the first two lines of characters are:

| | |
|-----------------------------|---------------------------------|
| NUL (null) | SOH (start of heading) |
| STX (start of text) | ETX (end of text) |
| EOT (end of transmission) | ENQ (enquiry) |
| ACK (acknowledge) | BEL (bell) |
| BS (backspace) | TAB (horizontal tab) |
| LF (NL line feed, new line) | VT (vertical tab) |
| FF (form feed) | CR (carriage return) |
| SO (shift out) | SI (shift in) |
| DLE (data link escape) | DC1 (device control 1) |
| DC2 (device control 2) | DC3 (device control 3) |
| DC4 (device control 4) | NAK (negative acknowledge) |
| SYN (synchronous idle) | ETB (end of transmission block) |
| CAN (cancel) | EM (end of medium) |
| SUB (substitute) | ESC (escape) |
| FS (file separator) | GS (group separator) |
| RS (record separator) | US (unit separator) |

Thus, for example, Q is coded as 51 and m as 6D.

EXERCISE 3.17

1. Convert your surname to a binary sequence using the ASCII code for each of the letters in your surname.
2. Convert the binary sequence to a HEX string.
3. Use the Internet to look up the details about the 16 bit character code known as UNICODE.
4. Why is it possible in a programming language such as C or Java to assign a letter to a char and then assign the char value to an integer variable?
5. What does this algorithm do:

```

char x = 's'
int p
p = x
p = p + 2
x = p
Output p

```

Explain the result.

6. Use the Internet or another suitable resource and research what is meant by the term 'extended ASCII code' i.e. how many bits are used and what this allows.
7. Use the Internet or another suitable resource and research what is meant by the term 'UNICODE'. How many bits are used in UNICODE systems and what implication does this have?

Note: Java uses the UNICODE system and ASCII remains as a subset for compatibility with other devices.



©IB03.5.3 EXPRESS NUMBERS IN DECIMAL, BINARY AND HEXADECIMAL

The decimal number system uses a base of 10 to determine the place values. The place values increase by powers often as we move from right to left i.e. $10^0, 10^1, 10^2, 10^3$ etc. The digits for the base ten system range from 0 to 9. Note the maximum digit is always one less than the base. Thus the range of digits for different bases is always 0 to $N-1$, where N represents the base.

The table below shows the first four place values in the decimal system. We can use it to show what is meant by a particular decimal representation by multiplying the digit by the corresponding place value and adding these up. For example, what quantity is represented by 3432 base 10 or 3432_{10} ? Note when showing the base it is subscripted and, if this is omitted, the number is assumed to be base 10.

| 10^3 | 10^2 | 10^1 | 10^0 |
|--------|--------|--------|--------|
| 1000 | 100 | 10 | 1 |
| 3 | 4 | 3 | 2 |

Thus the value 3432 represents:

$$3 \text{ lots of } 1000 = 3000 +$$

$$4 \text{ Jots of } 100 = 400 +$$

$$3 \text{ lots of } 10 = 30 +$$

$$2 \text{ lots of } 1 = 2$$

When these are all added together we get 3432 base 10 i.e. three thousand four hundred and thirty two.

When representing fractional quantities to the right of the decimal point, the place values of the denominator are powers of 10 that start at 1 and increase by 1 as you move from left to right. This means the place values are (10^{-1}) i.e. $\frac{1}{10^1} = 0.10$, (10^{-2}) , $\frac{1}{10^2} = 0.01$, (10^{-3}) , $\frac{1}{10^3} = 0.001$ etc.

These are referred to as one tenth, one hundredth, one thousandth etc.

Thus the fraction 0.123 represents the following:

$$1 \text{ lot of tenths} = 1 \times 0.10 = 0.10 +$$

$$2 \text{ lots of hundredths} = 2 \times 0.01 = 0.02 +$$

$$3 \text{ lots of thousandths} = 3 \times 0.001 = 0.003$$

When these are added together we get the value 0.123 base 10, which is read one hundred and twenty three thousandths.

THE BINARY NUMBER SYSTEM

The binary number system uses a base of 2 and thus only allows the digits 0 and 1. The place values are thus powers of 2 starting at 0 i.e. $2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 = 16$ etc. In the fraction part, the place values are determined using a denominator of powers of 2 starting at 1 i.e.

$\frac{1}{2^1}, \frac{1}{2^2}, \frac{1}{2^3}, \frac{1}{2^4}$ etc. These values can be represented using negative indices as $2^{-1}, 2^{-2}, 2^{-3}$ etc.

Representing binary numbers

The table below shows how to convert from a binary value to a decimal integer equivalent. The table can also be used to convert from decimal to binary.

| 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| | | | 1 | 1 | 0 | I | 1 |

The first row shows how the place value is determined. The second row shows the quantity represented in decimal form. The third row shows the binary sequence 11011_2 .

Thus to convert 11011_2 to decimal we add up the place values where a I appears.

$$1 \times 1 + 1 \times 2 + 0 \times 4 + 1 \times 8 + 1 \times 16 = 1 + 2 + 0 + 8 + 16 = 27_{10}$$

It should be noted that computer memory is usually displayed as a series of Is and Os. In respect of the ASCII code each character's representation also has an integer value equivalent. This is used to convert alphabetic characters from lower to upper case by adding a fixed integer value to the character's integer value. Most programming languages allow direct conversion from character to integer by assignment.

HEXADECIMAL (BASE 16)

The hexadecimal system uses base 16 and thus allows digits 0 to 9 and uses letters to represent the quantities $10 = A, 11 = B, 12 = C, 13 = D, 14 = E, 15 = F$. There are 16 digits in all. The place values are powers of 16 starting with 0 and increasing by one as you move from right to left.

The table below shows how to read hexadecimal numbers.

| 16^3 | 16^2 | 16^1 | 16^0 |
|--------|--------|--------|--------|
| 4096 | 256 | 16 | 1 |
| 1 | D | F | 3 |

The relationship between binary and hexadecimal numbers

The hexadecimal number $IDF3_{16}$ shown in the third column can be converted to its decimal equivalent by multiplying the place digit by its corresponding place value.

The calculation is shown below.

$$I \times 4096 + D \times 256 + F \times 16 + 3 \times 1$$

Computer Fundamentals

We now substitute the decimal values for D and F i.e. 13 and 15. We now have.

$$1 \times 4096 + 13 \times 256 + 15 \times 16 + 3 \times 1 = 4096 + 3328 + 240 + 3 = 7667_{10}$$

EXERCISE 3.18

Using tables of place values, convert the following to binary or hexadecimal.

1. Convert 123_{10} to its binary and hexadecimal equivalents.
2. Convert 101101_2 to its decimal equivalent.
3. Convert $1AF_{16}$ to its decimal equivalent.
4. Convert IAF_{16} to its binary equivalent.



The hexadecimal system is really just the binary system except that it groups the bits into lots of four. The place values of the binary system start at 1 and continue to two as represented in the table below. The hexadecimal place values can be read by taking every fourth value.

| Place number | binary system value | hexadecimal system value |
|--------------|---------------------|--------------------------|
| 1 | 1 | 1 |
| 2 | 2 | |
| 3 | 4 | |
| 4 | 8 | |
| 5 | 16 | 16 |
| 6 | 32 | |
| 7 | 64 | |
| 8 | 128 | |
| 9 | 256 | 256 |
| 10 | 512 | |



EXERCISE 3.19

1. What is decimal value of the 13th position place in the binary system?
2. By using the table above, what are the place values in the hexadecimal system for the first 3 places? What place values do these correspond to in the binary system?
3. What is the decimal value of the 4th hexadecimal place?

© IBO 3.5.4 CONVERSION OF NUMBERS IN DECIMAL, BINARY AND HEXADECIMAL
2004

CONVERTING BETWEEN BASES

We can convert binary and hexadecimal representations to their decimal equivalent by following the procedures shown above. But, from time to time, we may wish to convert decimal to binary, decimal to hexadecimal, binary to hexadecimal and hexadecimal to binary.

Converting decimal to binary

EXAMPLE

Convert 12_{10} to binary.

SOLUTION

This can be done by simply looking across the place values and picking a value, which is closest to, but, not greater than the desired value. Then move to the right and select values that add up to the desired number.

By doing this we get 1 lot of 8, 1 lot of 4, 0 lots of 2, 0 lots of 1 to give 1100_2 as equivalent to 12_{10} .

This is a bit hit and miss and is difficult to describe in an algorithm. Fortunately there is an easier way. Repeatedly dividing the starting decimal value by 2 and keeping the remainders gives us a simple way to convert. We stop when the number to be divided is 0.

2112

2 I 6 remainder = 0 (from dividing into 12)

2 I 3 remainder = 0 (from dividing into 6)

2 I 1 remainder = 1 (from dividing into 3)

2 I 0 remainder = 1 (from dividing into 1)

By reading upwards the remainders are 1100_2 which is the binary representation for 12 base 10.

CONVERTING BETWEEN BINARY AND HEXADECIMAL NUMBERS

As mentioned above the hexadecimal representation is really just binary. Note that $16 = 2^4$. Note also that the decimal values of hexadecimal digits can all be represented by 4 binary digits as shown in the table below.

| Binary | Hexadecimal |
|---------------|--------------------|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | 10 (A) |
| 1011 | 11 (B) |
| 1100 | 12 (C) |
| 1101 | 13 (D) |
| 1110 | 14 (E) |
| 1111 | 15(F) |

We can therefore convert binary numbers to their hexadecimal equivalent by grouping the binary digits into lots of 4 and representing the group by the hexadecimal digital equivalent. Start grouping from the right and if there are less than four digits remaining in the last group as you move to the left add enough zeros (0) to make four bits.

Thus to convert binary $110\ 1111_2$ to hexadecimal we would group the bits into two groups. We start from the right hand side (not the left!). In this case the first group is 1111 and the second now has only three bits so we add a zero to the left hand side so that the second group is now 0110. We now look up the table to find the hexadecimal equivalent.

0110 1111 is equivalent to 6 F base 16.

Likewise to convert from hexadecimal to binary we replace each hexadecimal digit with the corresponding four bit binary representation.

EXAMPLE

Convert FFD base 16 to binary.

SOLUTION

From the above table note that $F = 1111_2$ and $D = 1101_2$,

Thus the required binary number is $1111\ 1111\ 1101_2$

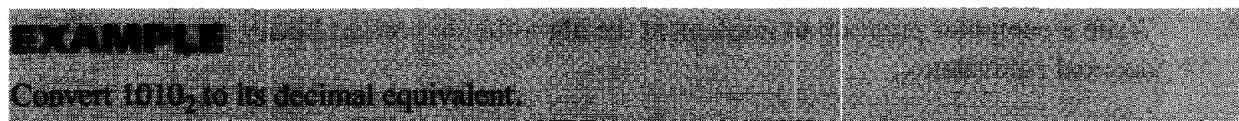
CONVERTING BINARY TO DECIMAL

How can we convert 101101_2 to its decimal equivalent? One way is to use the table approach and start with left-most bit and work towards the right calculating the decimal place value until no digits remain and then add up the values.

An algorithmic method exists. The steps are as follows:

```
Set total to 0
While not last bit
    add bit to total and then double the total
    move RIGHT to the next bit
Add last bit to total: STOP
total is the decimal value
```

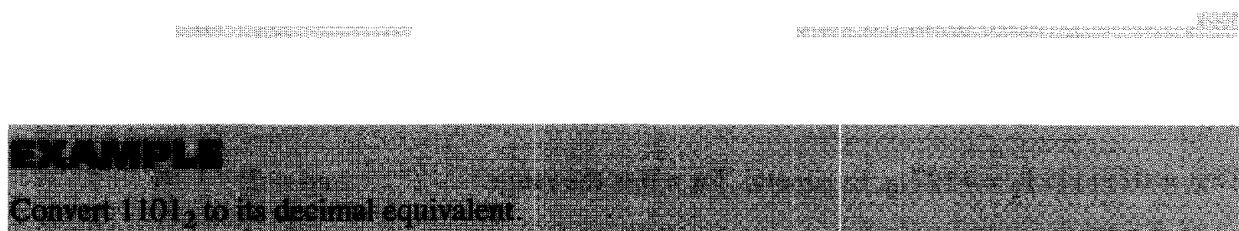
Let's apply this algorithm to 1010_2 and then 1101_2



SOLUTION

```
Set total =0, start at left
Add 1 to total giving total = 1
Double and move to right
Add 0 to total and double and move to right. total = 4
Add 1 to total and double i.e. total = 10
Last bit. Add 0 to total and Stop.
Total = 10
```

Thus the decimal value of $1010_2 = 1010$



SOLUTION

```
Set total = 0, start at left
Take 1 and add to total and then double total i.e. total = 2
```

Take 1 and add to total and then double total i.e. total = 6
Take 0 and add to total and then double total i.e. total = 12
Last bit. Add 0 to total and Stop.
Total = 13

Thus the decimal value of $1101_2 = 13_{10}$

EXERCISE 3.20

1. All values are base 10. Convert 7,63, 127,255 to base 2. Use the remainder method. Do you notice a pattern?
2. Convert the decimal values in 1 to hexadecimal values.
3. Convert 101101110110_2 to hexadecimal.
4. Convert 1001111_2 to hexadecimal.
5. Convert 10101_2 to decimal by using the conversion algorithm.
6. Convert 101110110111_2 to decimal.
7. Write a computer program to implement the algorithm to convert a decimal number to its binary equivalent.
8. Write a computer program to implement the algorithm to convert binary numbers to their decimal equivalents.



© IBO 2004 **3.5.5 REPRESENTING INTEGERS**

Negative numbers in a computer are represented by using complementary arithmetic. This representation has the advantage of turning subtractions into additions, thus simplifying the computer's design.

Negative numbers

To do this we use what is called the 2's complement method.

In 2's complement the MSB has a negative place value. This means that if we use an 8 bit representation the MSB (the 8th bit on the left hand side) would have the place value of -128. Thus $10000000_2 = -128$.

To represent positive numbers we use the normal method and set the MSB to 0. This means that for 8 bits the range of positive integers is from 0 to 127. Note that we can calculate the maximum integer by use of the formula $2^{(n-1)} - 1 = +127$, where $n = 8$ bits.

That is $01111111_2 = +127_{10}$. In general, for n bits the range is $-2^n - 1$ and $+2^n - 1$.

An aside. Note that $10000000_2 = 128_{10}$ and that $01111111_2 = 127_{10}$. This is an important general result. If we add binary 1 to 01111111_2 we get 10000000_2 . In other words we get a carry flowing all the way across the number.

How do we calculate the 2's complement of a negative number?

We will first do this in decimal, but we are assuming 8 bits.

To represent -34 we need to use a 1 in the MSB i.e. represent -128. Thus to get -34 we need to add 94 to -128 to give -34.

Mathematically this means solving the equation:

$$\begin{aligned} -128 + x &= -34 \\ x &= -34 + 128 \\ &= 94 \end{aligned}$$

We now represent 94_{10} in the remaining 7 bits in the normal way.

Thus we get the representation 1101110₂

The table below will help you to work this out. The first row shows the decimal place values with the left most value being -128. The remaining place values are 64 down to 1, which are determined in the normal way i.e. $1 = 2^0$ and $64 = 2^6$.

| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------|----|----|----|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

Using the above table the representation for -34 is shown in the second row. The third row shows the representation of +34.

Now, if we add these values together i.e. $-34 + 34$ we should get 0.

$$\begin{array}{ccccccc} 1 & 1 & 0 & 1 & 1 & 1 & 0 & (-34) \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & (34) \\ 1 & 1 & 1 & 1 & 1 & 1 & & (\text{carry bits}) \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & \text{(result, the last bit that carries over is not used)} \end{array}$$

Algorithmic method

We can also apply an algorithm approach. To convert a sequence of binary bits to the equivalent 2's complement representation we flip the bits and add binary 1. By 'flip' we mean changing the Os to Is and the Is to Os. This is very easy to implement.

To apply this method to a negative integer we first represent its positive value in binary, then flip the bits and add binary 1.

EXAMPLE

Convert -010101 to 2's complement.

SOLUTION

Step!: flip the bits. Thus 010101 becomes 101010

Step 2: add binary 1. Thus $101010 + 1$ gives 101011

EXAMPLE

Convert -34 to an 8 bit 2's complement format.

SOLUTION

We know the answer to this from the above example.

Step 1: represent 34 as an 8 bit binary number (note we use the full 8 bits) 00100010

Step 2: Flip the bits 11011101

Step 3: Add binary 1.

$11011101 + 1 = 11011110_2$ is $=-34$ and corresponds to what we found above.

EXERCISE 3.21

1. Using 6 bits and 2's complement, represent the following numbers:

(i) -12 (ii) -31 (iii) -2

2. For each number, confirm that the addition of the positive number gives zero.

3. Convert -12 to 2's complement by using the algorithmic method.

4. Convert -31 and -2 to 2's complement using the algorithm method.



3.5.6 DEFINE ANALOG AND DIGITAL DATA

Analog data (A) is represented by some form of continuous representation, for example a wave that represents a physical quantity or a sound level. The sound made by the human voice is called a sound wave and can be represented by the use of a graph that shows the sound levels made by the voice. These waves can be represented by $\sin(x)$ functions.

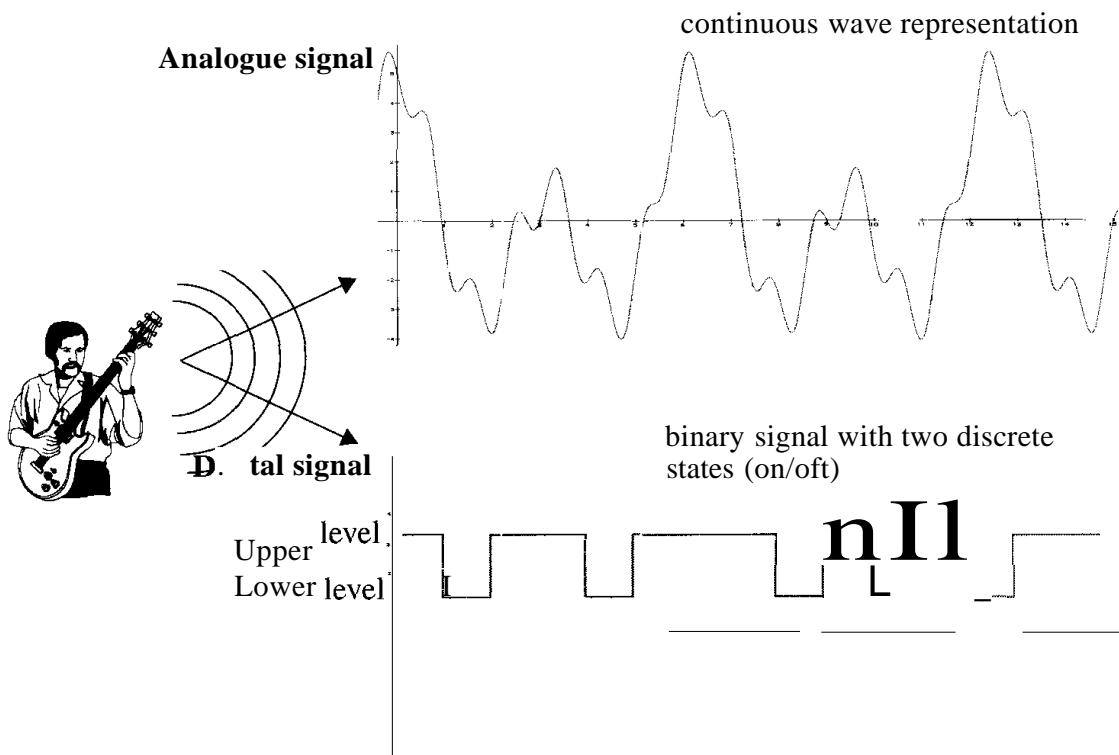
Digital data (D) is data that can be represented as discrete quantities as opposed to continuous quantities. For example, the number of people at a football match is certainly not a continuous piece of data! Data inside a computer is stored using the binary system, which is itself an on-off system. Data within a computer therefore must be stored using a binary system.

In the normal world a great deal of data that is collected or captured is of an analog variety. For the analog data to be stored in a computer it must be converted to a digital format. In many cases this conversion process results in loss of accuracy as not all possible continuous values of the analog data can be represented using a binary code.

The diagram in figure 3.19 shows the difference between analog and digital data. It also depicts the process of input as a conversion from analog to digital and the process of output as often

involving the conversion of digital to analog e.g. playing back music that has been stored digitally.

Figure: 3.19



At regular intervals, the values of the analogue sound wave are converted to binary numbers representing, for example, pitch, amplitude and duration. This is known as 'sampling' - see the example in Section 3.5.8.

EXERCISE 3.22

1. Define the terms 'analog data' and 'binary data'.
2. Why is a wave often a suitable representation of analog data, for example, to represent sound, but is inappropriate to represent binary data?
3. List some examples of each form of data.



© IBO 2004 3.5.7 CONVERSION FROM DIGITAL TO ANALOG

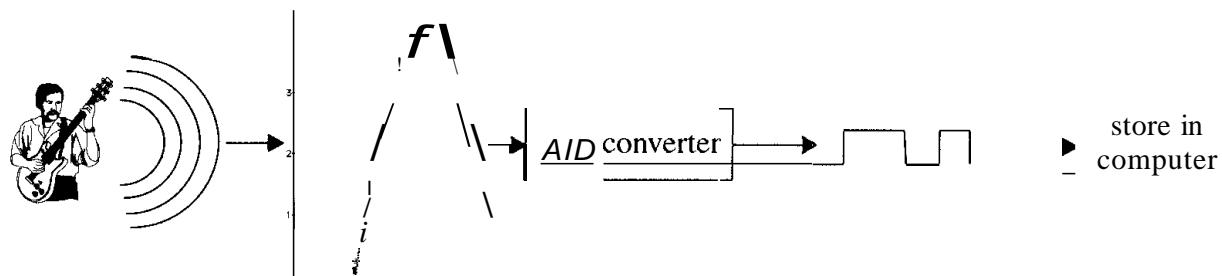
It is often the case that a computer is used to control the operation of an electrical device such as a motor. The computer is a digital world and an electrical device is an analog world i.e. voltage is measured in a continuous manner, not a digital manner.

To enable this control we need to use a digital to analog (D-A) converter.

It is also often the case that a computer needs to accept inputs from analog sources. To achieve this we need to use an analog to digital (A-D) converter.

Modems convert output from digital to analog format and then convert input from analog to digital format. The diagram below depicts the sequence of events.

Figure 2.20 - Sound to computer storage



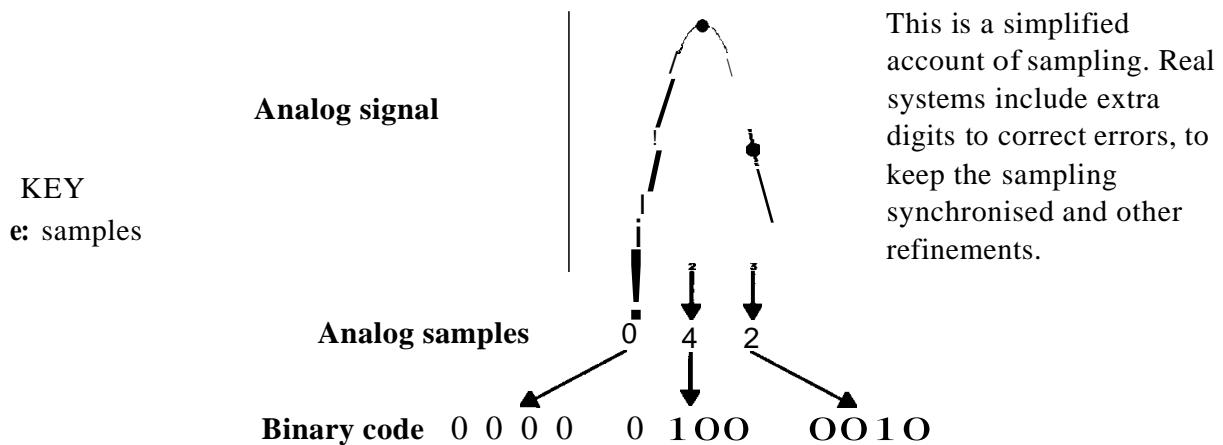
© IBO 2004 **3.5.8 APPLICATIONS OF CONVERSION**

There are many applications that require conversion. Some of these are now briefly outlined below:

Sound waves need to be represented in digital form in a computer system. They are obviously analog in the natural environment.

Temperature sensing requires that readings from thermometers are able to be converted from the analog reading to a digital representation. Thus an A to D conversion is required.

Figure: 3.20 - Sampling



Voice recognition software needs to convert an analog voice signal into a digital format.

Optical Character Recognition requires the conversion of analog representations of letters into a digital format.

EXERCISE 3.23

1. Explain, by use of a diagram, why inter-conversion of data between digital and analog formats is required in a computer system?
2. Discuss the converting of sound or temperature data from analog to digital and digital to analog.
3. What implications are there if more accuracy is required when representing an analog signal?



© IBO
2004

3.6 ERRORS

© IBO
2004

3.6.1 HOW DATA ERRORS OCCUR.

Data errors can occur in a variety of ways. Data is stored on secondary storage devices and the surfaces of these devices can be damaged. Physical Damage such as a hard disk crash and electrical interference can also lead to errors in data. The bits stored can be flipped i.e. a 1 can become 0 meaning that the byte 10011011 could be changed to 11011101.

Data errors can also occur at input time i.e. data entry errors. Data entry errors are often due to human error in either recording the original data or from the lack of data validation techniques to detect simple transcription errors e.g. entering 345 instead of 245 or transposition errors e.g. entering 345 instead of 354 in which the 5 and 4 digits have been transposed. Usually such errors occur accidentally.

Data can also be changed deliberately. For example a student might gain entry to the student record system and alter a mark stored for either themselves or for someone else.

© IBO
2004

3.6.2 PREVENTING AND DETECTING ERRORS

The detection of and/or checking for errors in an effort to prevent changes being made to the data or prevent mistakes being made at data input time is an extremely important process. The terms used to denote this process are that of maintaining the INTEGRITY or ACCURACY of the data and protecting the SECURITY of the data from unauthorised access.

Checking for errors

It is important to distinguish between validation and verification.

- Verification means making sure that the data on the source documents is exactly the same as that input to the computer system.
- Validation means attempting making sure that the data input into the system makes sense.

If it helps, remember that verification (*ve*) means making the data *equal* whereas validation (*va*) means checking if the data is *acceptable*.

Verification

There are two main methods in common use:

Visual verification (or proof-reading) which means... (can you guess?)

- Double entry verification which involves having the data from the source document entered twice as a check. Typically the application will lock the keyboard and give an audible warning so that the data entry person can check carefully.

One of these methods is more reliable than the other.

Validation

Data entered into the computer can be unreasonable. A data entry person in a medical clinic can easily mistype a person's weight, say, or the number of cigarettes they smoke per day. This type of error is not caught by verification since the data may have been written down wrongly in the first place.

Computer Fundamentals

Validation also checks for incomplete or inaccurate data. The main types of validation checks are range checks, type checks and format checks.

Range checks

Numeric data can be checked in a variety of ways. One way is to apply a reasonable upper and lower bound. A date can be checked to ensure that the day is between 1 and 31 and the month falls within the range 1 to 12 inclusive.

The typical algorithm for a range check is to use a function that accepts as the data value and values for the upper and lower limits of the range and returns a boolean true or false depending on the result. A while loop is used to control the process.

Data type checks

Data values can be checked to see that they match the expected data type.

e.g. a char value for lower case can be checked to see that it falls within the expected integer ASCII values.

A variable such as an age can be checked to see that letters have not been entered.

Data can be checked against valid lists of data. For example, a data type of MONTH could contain a list of the valid months. If a month name was entered it could be validated against the contents of the valid list.

Data format checks

A string field for name could be required to conform to the format requirement that at least some letters are input i.e. the field is required to have a value and not be left blank. Some fields might have a fixed length and some parts of the field may be character. For example, a date format: 14JUL1950. A check could be made that the data is in ddmmmyyyy format.

Check digits

Input data or data that is transmitted from one location to another via a network can be validated using the 'check digit' concept. Parity checking which is discussed below uses this same concept.

The notion is quiet simple. We apply some algorithm to the data and from this determine a number that is appended to the data value. This value is called the 'check digit'. On the entry of the data, or receipt of the data, this value is recalculated and the newly calculated check digit is compared to the original one. If they match, the data is assumed to be correct.

A more technical definition is that a check digit is a digit calculated from the other digits in the data item and appended to the data item. At input or prior to use the check digit is recalculated to detect any error in the data. The method does not provide a way to fix the error other than to alert the user that an error is present in the data. Check digits are not be confused with check sums, which are applied to bit streams or binary data.

There are a number of ways of calculating a check digit. More complex ways are presented in chapter 4. In this section will consider only two.

Method 1: Unweighted check digit.

Consider the data value 2345.

We can calculate a check digit by adding the digits up $= 2+3+4+5 = 14$

As 14 is not a single digit we repeat the addition ie $1 + 4 = 5$

5 is our check digit.

Thus our data value could be 23455, where the last digit is the check digit.

On input this digit can be recalculated.

Thus if 22455 was entered i.e. a transcription error has occurred with 2 being entered for 3.

Calculate the check digit ie $2+2+4+5 = 13 = 1+3 = 4$

which does not match 5, the original check digit. Hence we conclude that the data value is incorrect and we have successfully detected an error in the data. We could then either request a resend or re-enter the data to ensure that we entered it correctly.

This method does not pick up transcription errors. If 32455 was entered, i.e. the 3 and 2 have been transposed, this method would not detect the mistake. Let's check.

recalculate the check digit $= 3+2+4+5 = 14 = 1 + 4 = 5$, which unfortunately matches. If we used this validation technique we would not have detected the mistake and thus not prevented the data error.

Method 2: weighted check digit.

What we do is weight each digit. We start at the first digit and multiply it by 1, then move to the left and multiply by 2 and so on until there are no more digits. We do not include the original check digit. This calculation is done below.

check digit $= 1*5 + 2*4 + 3*3 + 4 *2 = 5 + 8 + 9 + 8 = 30 = 3+0 = 3$

Thus our new check digit is now 3 and the data value would be 23453

If we transposed the first two digits, i.e. entered 32453, this would be detected as an error during validation. The calculation is shown below.

new check digit $= 1*5 + 2*4 + 3*2 + 4*3 = 5 + 8 + 6 + 12 = 31 = 3+1 = 4$

As 4 does not match the check digit of 3 an error is detected.

There are a range of other similar methods that can be applied. For example, a sequential file could have as its first record the number of records in the file. This could be used to validate that the file has been copied correctly in terms of the number of records.

Hardware errors

Data that is stored on disk can be corrupted by a malfunction of the hardware. This can alter data, but often means that the data cannot be read, in which case the data needs to be retrieved from backup.

Software related errors

Errors can occur with software as well as the data entry errors described above. Software errors are usually put into one of three categories:

Logic errors: The coding of the program has been incorrect in sequencing or choice of conditions, such as:

```
// algorithm to add up 9 numbers
// example of logic error - contains 3 logical errors

count = 0 // error 1 count is not incremented
numbers = 9
while count <= numbers // error 2 loop executes 10 times
    sum = 0 Ierror 3 sum is reset each time!
    input (number)
    sum = sum + number
endwhile
output (sum)
```

Logic errors should be detected during program testing and can only be prevented by thorough testing and rigorous analysis of the requirements of the program!

Runtime errors: There are many things that can go wrong as a program runs, some examples are:

- Division by zero.
- Truncation errors.
- File not found.
- Printer not ready.
- Illegal memory access.

Runtime errors can be prevented by placing error traps within the code to detect if a run time error has been triggered e.g. a 'file not found' condition test. Such exception errors can be detected by most program run-time environments. This assumes that all possibilities have been predetermined by the programmer. It is not always possible to do this, but programmers should attempt to make their programs as robust as possible.

Syntax errors: These are errors in the syntax of the program language - such as mis-spelling a keyword (eg `publuc vode someMethod()`); these are caught at the compilation stage or while a program is being interpreted.

Final comment. Is there a real difference between preventing errors occurring and detecting them? In the above parity example we have detected an error but did not prevent it in the first place, but we did prevent a change occurring to the data! The use of verification prevents incorrect data being entered by detecting the possibility of a data input error. Data validation also detects the likelihood of an error and therefore prevents an erroneous piece of data being input and used!

© IBO 2004 3.6.3 RECOVERY FROM ERRORS**Re-input**

If data is input incorrectly and caught by verification, or is not acceptable and is caught by validation, it can simply be re-input in an online process. If it is a batch process, normally the errors will be reported (printed out) and the batch process will continue with the next input. The entries in error can then be included with the next batch.

Re-transmission

Transmission errors caught by parity checks can be automatically corrected in some circumstances but more commonly there will be a request for a re-transmission of the data.

Backing up

For organisations whose very existence depends upon computer-stored data (banks among many others) data is protected very carefully. Backups are taken every day and, indeed, every transaction is backed up somewhere, at an ATM or a local branch, for example. Backup media are typically stored in a fireproof safe or at a different location to the computer installation in case of a fire, flood or other disaster. Full backups save a new copy of every file, incremental backups save only those files which are new or have changed since the last backup. Tape is a common backup medium because it is relatively inexpensive and, hopefully, the backup data will never need to be accessed directly.

EXERCISE 3.24

1. Describe examples of how errors data entry errors can occur.
2. Describe how errors can occur in data due to the deliberate actions of individuals
3. How can software errors occur?
4. How can hardware problems cause errors?
5. Define the terms 'verification' and 'validation'.
6. Explain how parity checking operates by use of an example.
7. Use the resources available to you to find out how parity can be used to determine the actual bit that has changed, therefore making automatic error correction possible.
8. Discuss, by use of examples, a range of data validation methods.
9. Explain how re-transmission might operate in a network to aid recovery from an error in data transmission that has been detected.
10. A school reporting system fails on the morning that reports are to be printed. The deputy principle asks that the data file be recovered from the nightly backup. Explain why the backup is important and then outline a procedure for restoring the backup. Assume the backup is held on a tape stored in the school safe.
11. Explain the difference between a check digit and check sum. Use the Internet to look up how the TCP/IP Internet protocol implements a check sum error detection process.



© IBO 2004 **3.7 UTILITY SOFTWARE**

The term 'utility software' refers to system software that is used to perform some form of system level function to do with managing the data or performing some form of check on the data. The software utilities covered include: data compressors, virus checkers, file managers and de-fragmentation software.

© IBO 2004 **3.7.1 AND 3.7.2 FUNCTIONS AND NEED FOR THE ABOVE UTILITY SOFTWARE.**

File managers

File managers perform a range of functions:

Copy: allows files to be copied from one location to another.

Delete: allows the user to delete files.

Rename: allows the user to rename the file.

Create folders (directories): allows the user to make separate areas.

Formatting a disk

Hard disks and floppy disks store data in sectors. These sectors have a fixed size e.g. 512 Bytes. When a disk is formatted, the sectors are linked in a long chain. The operating system maintains a file allocation table that contains the filename and the address of the first sector of the file. If the file extends over more than one sector the links between the sectors are used to retrieve the file. As files get deleted and reduced or expanded in size the sectors need not be contiguous i.e. 'side by side'. The more the sectors of a file are split up over the disk the longer it takes to access the whole file or sections of the file.

Find: allows the user to search a specified disk area for a file.

Backup:

Backing up data will be a routinely scheduled operation in every business whose activities depend upon data held in their computer systems. For many organizations, their data is everything; banks, for example, deal as much or even more with data as they do with money.

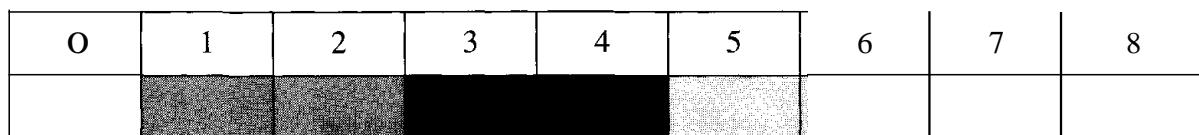
In organisations with very large amounts of data which is rarely accessed but still needs to be kept (legal firms keeping details of past cases, insurance companies keeping old contracts, oil companies keeping exploration data on different countries) the data will be archived. The weather case study of 2001 is a case in point. Vast quantities of data are gathered and stored for use with medium and long-range forecasting models. Often such data is stored in an automated retrieval system. For example, cassette tapes can be used with programmable machines used to fetch data for a particular week.

Restore: allows the user to request a file to be restored from the backup.

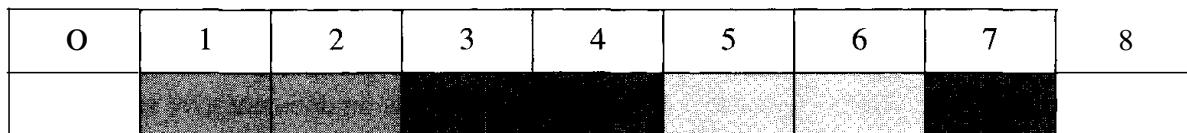
DEFRAGMENTATION

A disk's surface is also divided into used sectors and free sectors. In a fragmented disk the free space and used space tend to get mixed up. To increase performance a defragmenting software utility is run. It places all the files into closely linked sectors that occupy closely related sectors. This improves performance.

The diagram below will help explain the process. When files are first stored on a newly formatted disc, they are stored in continuous blocks i.e. all blocks related to the one file are linked next to each other.



If the file `'A'` is removed and another file added which, say, occupies 3 blocks; the operating system uses the first two available block previously occupied by the file `'A'` (blocks 3 and 4) and then links to the rest of the file in the block 7.



The file `'B'` is now fragmented. If this process continues with the addition and deletion of many files then a single large file can be split over different parts (even different surfaces) of a disk. This increases the time taken to access and load the file into memory.

The purpose of defragmentation is to group the related disk areas together to improve the speed at which data is recovered or read.

DATA COMPRESSORS

Data files can take up a great deal of space and can take considerable time to transmit. An obvious and rich source of interest in computer science is to ask 'is it possible to somehow reduce the size of file in some way to speed up transmission and save space'?

Data compression software can be used with text and other files to:

- save space on backing store.
- save time when transmitting data (e.g. over the internet).

When compressing text files it is important to be able to decompress them to get back exactly what was there in the beginning.

An example of how text compression might work:

The brown cow jumped over the lazy dog.

Certain words appear commonly between spaces (such as "the") and can be represented by a single "token" or symbol thus saving 4 spaces in the file. The same for common combinations of letters like "ow" or "er" or "ed", these can then be represented as a single token. The compression! de-compression utility needs to maintain a table of tokens and their expanded meaning. This would make an interesting HL dossier topic.

Another common form of text compression works by self-reference. Consider the phrase:

The brown cow rowed the brown boat.

The word 'brown' starts at position 5 and occupies 5 characters. It also begins and ends with a space. Thus the second 'brown' can be replaced:

The brown cow rowed the 47 boat.

How many characters are saved here? What is likely to happen as text size increases?

In fact, more subtle techniques can be applied here since repeated sequences can cross, or be within, word boundaries. How far can you reduce the above sentence?

Try searching the internet for more details on Ziv Lempel compression.

Compression is also used with media files such as graphics, music and video files. In this case, unlike with text, some quality is sacrificed in the compression process.

Consider a red square image of 300 pixels by 300 pixels. Such an image contains only the one colour. An alternative way of storing the image would be store integer values for the height and width and then store the colour as an integer. If we assume an integer take 2 bytes we have reduced the image from 90,000 Bytes to 3 bytes!! Most compression does not give such startling results.

VIRUS CHECKERS

Viruses range from the harmless to the deadly. A virus can be loosely defined as:

a computer program which has the ability to replicate itself and/or has a "payload" - i.e. does something on the target system

This definition does not take into account subtleties like 'trojan horses' but it is sufficient for this course. You can find more information by searching the world wide web. A suitable reference is to be found at the symantec web site at

<http://www.symantec.com/avcenter/refa.html>

A virus checker has two roles. The first is to scan files and to report if any viruses are detected and to then attempt to remove the virus. The second role is to check incoming files to determine if they have a KNOWN virus.

EXERCISE 3.25

1. Outline the major feature of the following utilities:

virus checker.
defragmentation.
decompression.
file manager.

2. Some consider the virus checker to be the most important utility. Discuss this claim with reference to the impact that viruses can have.

3. Computer system performance can be enhanced by the use of file decompression and file defragmentation. Explain, by use of a diagram, how performance can be enhanced. Make sure you mention what type of performance is being improved.



4

Chapter contents

- 4.1 Number Systems and representations
- 4.2 Boolean Logic

© IBO
2004

4.1 NUMBER SYSTEMS AND REPRESENTATIONS

INTRODUCTION

This section covers the theory relating to the representation of integers and real numbers in computers.

The IB specific learning objectives are then covered in the sequence they are specified in the syllabus i.e. 4.1.1 to 4.1.6.

Note: some aspects of earlier SL level work covered in either chapter 2 or 3 are repeated in context here.

© IBO
2004

4.1.1-3 CALCULATE IN BINARY, DECIMAL AND HEXADECIMAL

Binary addition has the following four rules. Note: all calculations are using base 2.

$$0+0=0$$

$$1+0=1$$

$$0+1=1$$

$$1+1=0 \text{ carry } 1.$$

Why the carry?

Recall that in decimal $9 + 3 = 12$ carry the 1 to the tens column thus we get 12 base 10.

In binary $1 + 1 = 10$ base 2. In decimal $1 + 1$ is 2 and 2 is represented as 10 in base 2.

These rules are reasonably simple to apply. All the following examples are in base 2:

EXERCISE 4.1.1-3

Find $(100)_2 + (110)_2$

SOLUTION

$$\begin{array}{r}
 0 \quad 0 \\
 0 \quad + \\
 1 \quad \text{carry digits} \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 1 \quad 0 \\
 1 \quad 0 \\
 \hline
 \end{array}$$

Binary Subtraction:

Rules for binary subtraction are as follows:

$$0_2 - 0_2 = 0_2$$

$$1_2 - 0_2 = 1_2$$

$$1_2 - 1_2 = 0_2$$

$$0_2 - 1_2 = 1_2 \text{ , with a 'borrow' of } 10_2 \text{ and 'payback of } 1_2$$

We adopt the normal subtraction algorithm of 'borrowing' and 'paying back'.

Thus $0 - 1 = 1$ after borrowing. When we borrow in binary, we borrow 10_2 i.e. 2 in decimal.

When we borrow, we borrow decimal 2 and thus $(10_2 + 0_2) - 1_2$ is 1_2 . When we pay back we pay back 10_2 or 1_2 into the column to the left.

Let's try a simple example first and subtract $10_2 - 1_2$ in base 2 i.e. $2 - 1$ in base 10, which should give us 1.

EXAMPLE

Find $10_2 - 1_2$ in base 2

SOLUTION

Step 1: Borrow 1, i.e. 10, and thus get 1 and pay back 1

$$\begin{array}{r} 1(10)0 \\ 1 - \end{array}$$

1

Step 2: pay back of 1 as shown in the next part

$$\begin{array}{r} 10 \\ 11 - \end{array}$$

01 (answer) i.e. $2 - 1 = 1$ in decimal

EXAMPLE

Subtract 3 from 6 i.e. $110_2 - 011_2$

SOLUTION

$$\begin{array}{r} 1 \\ O \\ \hline O & 1 - \end{array}$$

Step 1: borrow 10 as shown in next part

$$\begin{array}{r} 1 \\ O \\ \hline (10)0 & 1 - \end{array}$$

Step 2: pay back 1 as shown in next part

$$\begin{array}{r} 1 & 1 & 0 \\ O & 1(1) & 1 - \end{array} \text{ (note: } 1(1) \text{ is really } 1+1 = 10, \text{ we now have } 1 - 10 \text{ in column 2)}$$

Step 3: we now have to borrow to 10 from column 3 and this gives $10 + 1 = 11$

$$\begin{array}{r} 1 & (11) & 0 \\ O & (10) & 1 - \end{array}$$

1 (note: perform $11 - 10$ to give 1)

Step 4: pay back 1 into column 3 as shown in next part

$$\begin{array}{r} O \\ 1 - \end{array}$$

O i.e. 3 in decimal, which is what we expected!

ADDITION OF HEX VALUES

Recall that hex is base 16 and uses digits 0 through to 9 and then the letters A through to F to represent the digits 10, 11, 12, 13, 14 and 15.

Place values in hex are powers of 16 and are therefore $16^0, 16^1, 16^2$ and so on.

When we add in decimal we carry values into the next column across when we exceed what we can represent in a column with the available digits. Thus $12+9$ gives a carry of 1 into the next column, i.e. a 10 is carried over and added to the current 10 to give 20 and the digit 1 remains in column 1, giving 21 as shown.

$$\begin{array}{r} 12 \\ 9+ \quad \text{gives} \quad 10 + (2+9) = 10 + 11 = 21 \end{array}$$

This process is often represented using a carry digit.

Likewise with hex, when we add values that equal or exceed 16 we need to perform a carry to the next column as shown when adding $8_{16} + 9_{16}$, (Note: $16_{10} = 10_{16}$)

$$\begin{array}{r} 8 \\ \underline{9+} \\ 17_{10} \end{array} \quad \text{which is 1 lot of 16 i.e. } (10 + 1)_{16} \text{ unit i.e. } 11_{16},$$

Let's do three more examples.

EXAMPLE 1

Add $A_{16} + 8_{16}$ to give 12_{16}

SOLUTION

A is 10 in decimal, thus we have $10 + 8 = 18$ in decimal, which is $16 + 2$.

$16 + 2$ in hex is $10_{16} + 2_{16} = 12_{16}$ i.e. we often say "2 and carry 1".

EXAMPLE 2

Add $F + F$ to give $1D_{16}$

SOLUTION

F is 15 in hex, thus we have $15 + 15 = 30$ in decimal, which is $16 + 14$.

$16 + 14 = 10_{16} + D_{16} = 1D_{16}$.

EXAMPLE 3

Add $A5_{16} + FD_{16}$ to give $1A3_{16}$

SOLUTION

$$5 + D = 5 + 14 = 19_{10} = 16 + 3 = 10_{16} + 3_{16} = 13_{16}$$

This means we have a carry of 1 into the left hand column, as we can only represent one digit in a column, and retain the digit 3 in the first column.

To complete the addition we have:

$A + F + 1 = 10 + 15 + 1 = 26_{10} = 16 + 10 = 10_{16} + A_{16} = 1A_{16}$. The two digits occupy positions 3 and 2.

Hence, putting the result together we form the final value of $1A3_{16}$.

RULES OF HEX ADDITION

Provided that the result of adding two hex digits together does not exceed 15_{10} we can simply write down the digit. But, where the results exceeds 15_{10} , we have a carry involved.

You can construct a 16 by 16 table to record all possibilities. This is left to you to do as an exercise, if you wish.

To calculate the carry, convert the sum to a decimal and then back to its hex representation, noting the carry as shown in the above example 3.

EXERCISE 4.1

All numbers in the first two questions are binary.

- 1.** Perform the following additions:

$$(i) \quad 101 + 10 \qquad (ii) \quad 111 + 11 \qquad (iii) \quad 1010101 + 101011$$

- 2.** Perform the following subtractions:

$$(i) \quad 10 - 1 \qquad (ii) \quad 11 - 1 \qquad (iii) \quad 1011 - 1011 \\ (iv) \quad 1111 - 100$$

- 3.** Perform the following hex additions, note the base has been left out.

$$\begin{aligned} & 13 + F \\ & FF + DD \\ & 12A + 2F \\ & FFF + ADF + ACF \end{aligned}$$



REPRESENTING INTEGERS

An 8 bit byte uses 8 bits. Thus it allows $2^8 = 256$ different combinations of Is and Os, i.e. 11111111 down to 00000000. This would allow us to store all the positive integers from 0 to 255 or 256 different numbers.

But this is only useful for representing unsigned positive integers. To represent both positive and negative numbers we could use what is termed 'sign and magnitude representation'. This is done by assigning the left-most bit (most significant bit - MSB) to represent the sign i.e. 1 for a negative and 0 for a positive. Thus: $1111\ 1111_2 = -127_{10}$ and $0111\ 1111_2 = +127_{10}$.

We could now represent all the positive and negative from -127 to +127. Thus, in 8 bits we have the ability to store integers in the range (+ or -) $-2(8-1)$ - 1 to $+2(8-1) - 1$.

This can be generalised for n bits. That is, if there are n bits available and we are using the MSB as a sign bit, the range of integer numbers that can be represented is between $-(2^{n-1} - 1)$ and $+2^{n-1} - 1$.

EXERCISE 4.2

1. Using 6 bits, what range of numbers can be represented?
2. Confirm your answer by applying the formula.
3. In general, if there are n bits and we use an unsigned representation, what can you say about the number represented by 1s in all but the MSB, which is 0? What is the number represented by the MSB being set to 1 and all the other bits set to 0?



Negative numbers in a computer are represented by using complementary arithmetic. This representation has the advantage of turning subtractions into additions, thus simplifying the computer's design.

REPRESENTING NEGATIVE NUMBERS

To do this we use what is called the TWO's (2's) complement method.

In two's complement the MSB has a negative place value. This means that if we use an 8 bit representation the MSB (the 8th bit on the left hand side) would have the place value of -128. Thus $10000000_2 = -128$.

To represent positive numbers, we use the normal method and set the MSB to 0. This means that for 8 bits, the range of positive integers is from 0 to 127. Note that we can calculate the maximum integer by use of the formula $2(n-1)$ (= +127, where $n = 8$ bits).

That is $01111111_2 = +127_{10}$, In general, for n bits the range is -2^{n-1} and $+2^{n-1} - 1$.

An aside. Note that $10000000_2 = 128_{10}$ and that $01111111_2 = 127_{10}$. This is an important general result. If we add binary 1 to 01111111_2 we get 10000000_2 . In other words we get a carry flowing all the way across the number.

How do we calculate the 2's complement of a negative number?

We will first do this in decimal, but we are assuming 8 bits.

To represent -34 we need to use a 1 in the MSB i.e. represent -128. Thus, to get -34 we need to add 94 to -128 to give -34.

Mathematically, this means solving the equation:

$$\begin{aligned}-128 + x &= -34 \\ x &= -34 + 128 \\ &= 94\end{aligned}$$

We now represent 94_{10} in the remaining 7 bits in the normal way.

Thus we get the representation 1101110_2

Number Systems

The table below will help you to work this out. The first row shows the decimal place values with the left most value being -128 . The remaining place values are 64 down to 1 , which are determined in the normal way i.e. $1 = 2^0$ and $64 = 2^6$.

| | | | | | | | |
|------|----|----|----|---|---|---|---|
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

Using the above table the representation for -34 is shown in the second column. The third row shows the representation of $+34$.

Now, if we add these values together i.e. $-34 + 34$ we should get 0.

$$\begin{array}{ccccccccc} & & \text{O} & & & & \text{O} & & (-34) \\ & \text{O} & \text{O} & & \text{O} & \text{O} & \text{O} & \text{O} & (34) \\ & & & & & & 1 & & (\text{carry bits}) \\ \\ \text{O} & \end{array}$$

(result, the last bit that carries over is not used)

Algorithmic method

We can also apply an algorithm approach. To convert a sequence of binary bits to the equivalent 2's complement representation we flip the bits and add binary 1. By 'flip' we mean changing the Os to Is and the 1s to Os. This is very easy to implement.

To apply this method to a negative integer we first represent its positive value in binary, then flip the bits and add binary 1.

EXAMPLE

Convert -010101 to 2's complement.

SOLUTION

Step 1: flip the bits. Thus 010101 becomes 101010

Step 2: add binary 1. Thus $101010 + 1$ gives 101011

EXAMPLE

Convert -34 to an 8 bit 2's complement.

SOLUTION

We know the answer to this from the above example.

Step 1: represent 34 as an 8 bit binary number (note we use the full 8 bits) 00100010

Step 2: Flip the bits 11011101

Step 3: Add binary 1.

$11011101 + 1 = 11011110_2$ is $= -34$ and corresponds to what we found above.

WHY THE 2'5 COMPLEMENT ALGORITHM WORKS

Think of n bits, in this test case $n = 4$



All members of B_n start with a 0.

All members of C_n therefore start with a 1.

Each member of B_n has a corresponding one and only one member in C_n .

When we add the corresponding members we form the sequence 1111, e.g. $0101 + 1010 = 1111$.

This is true for ALL pairs.

If we now add binary 1 to all C_n members, we get a new set of sequences that are also related to the same B_n members in a one to one relationship.

i.e. B_n 0101 is linked to C_n 1010 $+ 1 = 1011$.

If we now add the two sequences we get 10000! i.e. (B_n) 0101 $+ (C_n + 1)$ 1011 $= 10000$.

If we ignore the MSB bit the resulting sequence is equivalent to zero.

This result enables us to implement the conversion of binary sequences to their complementary format with the property that if we add them, we get zero.

This is the basis of the 2's complement algorithm!

EXERCISE 4.3

1. Using 6 bits and 2's complement represent the following numbers.
 - (i) -12
 - (ii) -31
 - (iii) -2
2. For each number confirm that the addition of the positive number gives zero.
3. Convert -12 to 2's complement using the algorithmic method.
4. Convert -31 and -2 to 2's complement using the algorithmic method.



REPRESENTING REAL NUMBERS

Real numbers differ from integers in that they can contain a fractional component e.g. 45.0098. Real numbers can be positive or negative. There are two ways of representing real numbers: fixed point or floating point. Real numbers can be represented in any base and the whole number and fraction part are separated by a 'radix point'. In the base 10 (decimal) system this point is known as the 'decimal point'.

The binary number system represents the fraction component of a real number by using powers of 2 that are negative. The table below shows the decimal equivalent of the first eight bits working to the right of the radix point.

| 2^{-1} | 2^{-2} | 2^{-3} | 2^{-4} | 2^{-5} | 2^{-6} | 2^{-7} | 2^{-8} |
|----------|----------|----------|----------|----------|-----------|------------|-------------|
| 0.5 | 0.25 | 0.125 | 0.0625 | 0.03125 | 0.0015625 | 0.00078125 | 0.000390625 |

Thus a binary number such as 11.01_2 represents 3.25_{10} .

Fixed point

Fixed point assumes a fixed position for the radix point. Using normal decimal representation we could write a number as 23456534 and say that the decimal point is assumed to be between the 3rd and 4th digit. That is the number represented here is 23456.534 using base 10.

How does this work in a computer? Assume that a computer uses 8 bits to represent real numbers and that the radix point was always fixed between the 3rd and 4th bit position. This means that the bits in positions 4,5,6,7 are able to represent the whole number part and bits 3,2 and 1 are used to represent the fraction part. Bit 8 is used to represent a negative value.

This setup is represented by the diagram below.

| 8 th | 7 th | 6 th | 5 th | 4 th | 3 rd | 2 nd | 1 st |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 0 | 0 | 1 | 5 | 2 | 3 | 4 | 1 |

The number represented is + 152.341, assuming the MSB was a sign bit.

We shall discuss briefly below the disadvantages of this method.

However, computers can only store binary values of 1 or 0. Thus, using the above setup we could store only values between the minimum 11111.111 and the maximum 01111.111.

Note: the decimal fraction parts allows fraction values $\frac{1}{2}$, $\frac{1}{4}$ and $\frac{1}{8}$.

The minimum is thus:

$$-(8+4+2+1+\frac{1}{2} + \frac{1}{4} + \frac{1}{8}) = -15\frac{7}{8} = -15.875_{10}$$

The maximum is:

$$+8+4+2+1+\frac{1}{2} + \frac{1}{4} + \frac{1}{8} = +15\frac{7}{8} = +15.875_{10}$$

EXAMPLES

- (a) Using the above fixed point representation, what is represented by 00111.010₂?
- (b) Using the above fixed point representation, represent -6.5.

SOLUTIONS

$$(a) (+) 4 + 2 + 1 + \frac{1}{4} = +7\frac{1}{4} = +7.25_{10}$$

$$(b) (-) 0\ 1\ 1\ 0 . 100 = 10110100$$

This system can equally well be applied to the 2's complement notation where the sign bit is replaced by a bit having the same place value, but which is negative. To illustrate some limitations of fixed point, we now place the binary point between the 4th and 5th bits as well as using the 2's complement.

| 8th | 7th | 6th | 5th | 4th | 3rd | 2nd | 1st |
|--------|-------|-------|-------|---------------|---------------|---------------|----------------|
| -2^3 | 2^2 | 2^1 | 2^0 | 2^{-1} | 2^{-2} | 2^{-3} | 2^{-4} |
| -8 | 4 | 2 | 1 | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ | $\frac{1}{16}$ |

Thus 00111.010₂ in this system represents the decimal value $-8 + 0.5625 = -7.4375$.

Note that all the normal rules of 2's complement discussed in Chapter 3 still apply. Verify this by working out the value of +7.4375. This is still obtained by flipping all the bits and adding 1.

EXTENSION ACTIVITY:

Draw a number line and mark off ALL the possible numbers that can be represented in each of the above systems. This is not as difficult as it seems! What do you notice about the distribution and spaces between groups of numbers. What is the 'trade-off' between accuracy and range?



FLOATING POINT REPRESENTATION OF REAL NUMBERS

Real numbers such as 12.5 and -12.5 can be positive or negative and can have a fractional or decimal part i.e. 0.5 in the numbers stated here.

Over the years, the representation of floating point number in computers has caused a number of problems and it is only recently that a reasonably standard way of representing them has been implemented. This is known as the IEEE 734 standard. But, as it uses a slightly different way of presenting negative binary values than 2's complement, and also uses a variation on normalised format as used in the IE course, we will not be considering it here. Readers are referred to the following web site for a discussion of the IEEE standard and floating representation in general:

http://www.wikipedia.org/wiki/IEEE_Floating_Point_Standard

Real numbers can be written in scientific normalised format. This is done by moving the decimal point until the number is between 0 and 1 in the decimal system to form the MANTISSA and then multiplying by 10 raised to the appropriate power i.e. the number of places moved, which is called the EXPONENT.

For example: -123.0098::: -0.1230098*10³.

(Note: this is not quite the same as the mathematical definition of normalisation).

EXAMPLE

Convert 12.5 to normalised decimal format and then into normalised binary format.

SOLUTION

Normalised Decimal Format

Step 1: Move the decimal point two places to the left to give 0.125

Step 2: Count up the number of places moved and the direction to give +2

Step 3: Write the normalized format as 0.125 * 10²

Thus 12.5::: 0.125 * 10² (in normal maths, it is written as 1.125*10¹)

The mantissa is 0.125 and the exponent is 2

Normalised Binary Format

Step 1: Convert 12.5 to binary i.e. 1100.1₂

Step 2: Move the point 4 places to the left i.e. 0.11001₂

Step 3: Write the exponent 4 in binary i.e. 4]0::: 0100₂

Step 4: Write the number as (mantissa * 2^{exponent})

0.11001₂ * 2⁴



EXERCISE 4.4

Write the following in decimal and then binary normalised format: 8.25 & 11.5. State the mantissa and exponent in each case.



How is the binary format stored?

You will have noted that floating point representation uses two values: the mantissa and the exponent. Any internal computer representation needs to store these in a fixed number of bits. We will consider the issues of negative mantissas and exponents shortly.

Assume that we have 10 bits: 4 for the exponent and 6 for the mantissa. Also, assume that we cannot actually use the left most bit in either because we will use this to represent negative values using 2's complement. In the example that follows the left most bit is set to 0.

Our format looks like this with the MSB being reserved for the sign of the mantissa and hence the number ie 0 is + and 1 is -.

S m M M M M S E E E

Using this format the binary normalised format of 12.5 would be as shown:

S m M M M M S E E E

0 1 1 0 0 1 0 1 0 0

EXERCISE 4.5

Using the values from the previous exercise, represent each in this format.



What have we actually stored?

We have stored a fraction and an exponent.

The mantissa has this value:

$$0.11001 = \frac{1}{2} + \frac{1}{4} + \frac{1}{32} = \frac{16}{32} + \frac{8}{32} + \frac{1}{32} = \frac{25}{32} = 0.78125$$

The exponent has this value: 0100 = 4

To calculate the decimal value stored, we perform the following calculation.

$$0.78125 * 2^4 = 0.78125 * 16 = 12.5_{10}$$

EXERCISE 4.6

Select one of the values from the previous exercise and confirm that the representation is equal to the desired decimal value.



What about negative real numbers or small fractional numbers?

Recall that negative binary numbers are stored in 2's complement format. To derive the 2's complement format we flip the bits and add binary 1.

Thus negative numbers are stored using 2's complement to represent the mantissa and similarly for fractions, the exponent is negative and represented in 2's complement.

EXAMPLE

Convert -12.5 to binary format using the previous 10 bit format.

SOLUTION

Given that both the mantissa and the exponent are written in 2's complement format we can now write in the decimal values of the sign bits for the mantissa and exponent.

The exponent is -8 and the mantissa is -1 i.e., the leftmost bit is negative. Hence the place values of the 10 bit format are as shown in the following diagram.

| Sm | M | M | M | M | M | Se | E | E | E |
|----|--------|--------|--------|---------|---------|----|---|---|---|
| -1 | 1 2 | 1 4 | 1 8 | 1 16 | 1 32 | -8 | 4 | 2 | 1 |

Note: there is a binary point between the Sm and the M, and the MSB indicates the overall sign of the number.

How to convert -12.5

Step 1: Convert 12.5 to fixed point 01100.1

Step 2: Normalise (shift 4 places) 0.11001 exponent 4.

Step 3: Convert exponent to base 2 0.11001 exponent 0100

Step 4: Convert the mantissa ONLY by flipping bits and adding 1 1.11001 0100

Step 5: Convert back to decimal as a check:

$$(-1 + \frac{4}{32} + \frac{2}{32} + 1) \times 2^4 = (-1 + \frac{7}{32}) \times 2^4 = -\frac{25}{32} \times 16 = -12.5$$

It is outside the scope of the course, but to subtract two numbers we add the complement mantissa and the other mantissa i.e. $12.5 - 12.5 = 0$ as shown below

011001

100111 + complement

000000 i.e. the mantissa is zero and it is irrelevant what the exponent is.

What does the computer actually store? In this instance, the value of the mantissa is -0.78125 as shown below, and the exponent is 4.

| Sm | M | M | M | M | M | Se | E | E | E |
|----|--------|--------|--------|---------|---------|----|---|---|---|
| -1 | 1 2 | 1 4 | 1 8 | 1 16 | 1 32 | -8 | 4 | 2 | 1 |
| -1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

$$\begin{aligned}
 \text{Thus the mantisa} &= -1 + \frac{1}{8} - \frac{1}{16} + \frac{1}{32} \\
 &= -1 + \frac{4}{32} - \frac{2}{32} + \frac{1}{32} \\
 &= -\frac{32}{32} + \frac{7}{32} \\
 &= \frac{25}{32} \\
 &= -0.78125
 \end{aligned}$$

The exponent is 4 and thus $2^4 = 16$.

Hence the decimal value represented is $-0.78125_{10} * 16 = -12.5_{10}$

Thus there are alternative ways of evaluating and converting numbers and students will benefit from trying different strategies 'by hand' rather than relying on a calculator.

EXERCISE 4.7

Take each of the values in exercise 4.4 and assume they are now negative. Using the method shown above, convert each to the equivalent binary representation. Use the 10 bit format.



How to represent small fractions.

EXAMPLE

Represent 0.125_{10} as a binary floating point number.

SOLUTION

$$0.125_{10} = 0.001_2$$

In nonnormalised format this is $0.1 * 2^{-2}$

We note that the mantissa is positive but the exponent is negative.

The mantissa is therefore 010000, using the 6 bit mantissa.

Thus we need to convert the exponent of -2 to a 4 bit 2's complement format (remember we are using the 10 bit format).

$$\text{Thus } -2 = -8 + 4 + 2 = 1110_2$$

The exponent is thus 1110₂ and the full format is 010000 1110₂

Note the MSB is 0, indicating that the number is positive.

EXERCISE 4.8

1. Using the 10 bit format, represent the following values: 0.5, 0.25, -0.5, -0.25
2. What range of numbers can we represent using the 10 bit format?



Let's review some number line concepts. Assume zero is in the middle.

1. As you move to the right of zero the numbers increase in size and are positive.
2. As you move left from positive 1 towards, but not past zero, the numbers decrease in dimension and are fractional with increasing numbers of leading zeros to the right of the decimal point e.g. 0.000000000125_{10}
3. As you move to the left of zero the numbers decrease in dimension and get small i.e. -10 is smaller than -9 (would you rather give away 10 amounts of gold of the same size or 9?). e.g. -0.125_{10} is smaller than -0.00125_{10} .
4. As you move right from -1 towards, but not past zero, the negative values get close to zero but not equal to zero and the number leading zeros increases, for example: -0.00000125_{10} .

EXERCISE 4.9

Draw a number line and, using the 10 bit format, convince yourself of the above points.



Largest Positive magnitude number

Assume we are still using our 10 bit format i.e. 6 bits in 2's complement format for the mantissa and 4 bits in 2's complement for the exponent.

On the number line the term 'largest magnitude' is used to refer to positive and negative numbers as they move away from zero. Thus the largest positive number is in fact the number with the largest positive magnitude. Likewise the negative number that is the furthest away from zero is referred to as the negative number with the largest negative magnitude.

Numbers close to zero are said to have small magnitude and can either be positive or negative.

The largest positive floating point number has the largest mantissa and largest exponent.

The maximum exponent is $0111_2 = 7_{10}$

The maximum mantissa is $011111_2 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \underline{\frac{1}{32}} = \frac{31}{32}$

Maximum $= \frac{31}{32} \times 2^7 = 0.96875 \times 128 = 124_{10}$

Another way to think of this is that we want the normalised fraction component to be as close to 1 as possible so that the multiplication by the largest exponent value is maximised.

Largest negative magnitude number

The largest negative magnitude number is formed by using the minimum mantissa.

i.e. $100000_2 = -1_{10}$ in 2's complement and the maximum exponent i.e. $0111_2 = 7_{10}$

Thus the number $= -1 * 2^7 = -128$.

In summary, the range of numbers using the 10 bit format is -128 to +124

Smallest magnitude positive number

This is the number closest to zero on the positive side of the number line. It has the smallest possible normalised mantissa and the largest negative exponent:

0.100001000 or $0.5 \times 2^{-8} = \frac{0.5}{256} = 0.001953125$

Notice that, although a mantissa of 000001 is smaller, it is not normalised.

Smallest magnitude negative number

This is closest to zero but negative. It has the smallest possible normalised negative mantissa and the largest negative exponent:

1.011111000 or $-0.53125 \times 2^{-8} \approx -0.0020752$.

Notice that 1.00001 is actually larger in magnitude than 1.01111. It is $-\frac{31}{32}$ rather than $-\frac{17}{32}$.

Trying to represent numbers closer to zero than those given above results in an 'underflow error'.

How to interpret bit patterns

An alternative way of working with bit patterns is to move the decimal point as explained below.

Positive values

What is represented by 0.11010 | 0011?

The number is positive as the MSB is 0.

The exponent is decimal 3. Thus, move the decimal point three places to the right to give:

$$0110.10_2 = 4 + 2 + 0 + 0.5 = +6.5]0$$

(Note: an alternative is to think of the decimal point remaining fixed and the digits shifting three places to the left!).

Negative values

What is represented 100110 | 011?

The number is negative as there is 1 in the MSB.

Thus we need to revert to the original bit pattern. Thus, flip the bits and add 1.

Flip 100110 to get 011001.

Add 1 to get 011001 + 000001 = 011010

The exponent is $011_2 = 3]0$ hence, move the point 3 places to the right.

Thus, the number is as followed.

$$011010 = 0.11010 = 0110.10 = -6.5.$$

EXERCISE 4.10

Consider a system with a 4 bit mantissa and a 6 bit exponent.

1. What is the range of numbers that can be represented in this system?
2. What is the effect on the precision with which numbers can be held?
3. How is zero represented? What problems might this issue raise?
4. Write the following numbers in normalised form. Clearly indicate the mantissa and exponent in each: (i) 123.098 (ii) 0.00004567 (iii) -453.09888
5. Assume that we have a 16 bit format with 10 allocated to the mantissa and 6 to the exponent, both are 2's complement format.
Represent the following numbers. 11.125, -11.125
6. Using your chosen programming language, find out the method it uses to represent integers and floating point numbers. Find out also the range of integers and floating point values provided by the language.



©IB04.1.4 ADVANTAGES OF INTEGER AND FLOATING-POINT REPRESENTATIONS

2004

Integers have a finite range, but all values inside the range can be stored accurately. Whilst overflow errors occur these can be pre-empted by the programmer and trapped because the exact maximum and minimum can be easily determined given the number of bits used. Commonly, 32 bits are used to store single precision and 64 bits are used to represent double precision or long integer values. Using an unsigned integer we can store large positive integers using 32 bits.

Using 32 bits 2's complement the maximum integer is $n = 32$.

$$01111111111111111111111111111111_2(32-1)_{10} = 2147483647.$$

The smallest negative integer is -2^{31} . Whilst this is a reasonable range we can do better, but we may lose some accuracy.

Floating point representation greatly expands the range of numbers that can be represented. However, not all numbers can be represented in the allowable range. This can lead to truncation errors that in turn lead to inaccuracy and possible errors in output.

It is also an interesting fact that the gap between representable numbers is not constant.

The other obvious advantage of floating point over integer representation is that it allows the use of fractional values. This is clearly outside the scope of the integers.

Fixed point representation of real numbers suffers from similar limitations of range as integer representations.

EXERCISE 4.11

1. Assume that you were to use the 8 bit fixed representation for floating, with the decimal point being assumed to be between the 5th and 6th bit. What is the range of numbers that can be represented?
2. Investigate your chosen programming language to determine how it handles overflow and underflow errors. Do overflow errors cause run-time errors for floating point values? Are underflow errors reported as zero?
3. Write a program that accepts a standard integer and prints this out. Try to enter large values and see what is displayed. Explain the result.
4. Write a program to store a floating point number and then assign this value to an integer. Printout the percentage rounding/truncation error that has been introduced.
5. Write a function to accept a floating point value and an integer value that represents the number of decimal points to be rounded off. For example, 12.3456 and 3 would indicate rounding off to the third decimal place and 12.346 would be returned by the function.
6. Write a program to investigate the following looping algorithm. Vary the increment value to see if there are times when the test of the sum does not give the expected result.

Start

```
increment      0.001
sum = 0
counter = 0
limit = 6
```

```

        while (counter < limit) {
            sum = sum + increment
            counter = counter +1
        }
        if (sum == counter + increment) {Output 'yes'}
        else {Output 'No'}
    End

```

Explain the different results, assuming there are some.



REVIEW EXERCISE 4.12

1. Define the terms 'fixed point representation' and 'floating point representation'.
2. Describe the difference between normalised and un-normalised form. Give examples to support your answer.
3. Using fixed point representation, show how 234.5 could be represented using 10 bits, where the decimal point is located between the 3rd and 4th bit. The MSB bit is a sign bit.
4. Represent 234.5 as a normalised decimal floating point number. Clearly state the mantissa and exponent.
5. Write 234.5 in binary format and then convert it into normalised format. State clearly the mantissa and exponent.
6. Using the 10 bit representation described earlier, is it possible to store 234.5? If not what type of error would occur if you attempted this?
7. In order to store 234.5 what size mantissa and exponent would be needed?
8. For the newly sized mantissa and exponent suggested above what is the range of real numbers that can be represented?



©IB0 415-6
2004

'TRUNCATION, UNDERFLOW AND OVERFLOW ERRORS

A truncation error occurs when a real number is assigned to an integer variable. For example, assume x is an integer and we are able to assign 12.098 to the variable. The stored integer is 12. An error of 0.098 has occurred and this amount of accuracy has been lost.

Overflow integer errors can occur when you try to store a number that cannot be represented in the allocated number of bits. For example, if 8 bits have been allocated the maximum unsigned integer, i.e. positive integer, that can be stored is $1111\ 1111_2 = 1+2+4+8+16+32+64+128 = 255_{10}$. If we attempt to store a number bigger than 255 an overflow error will be reported i.e. 'number too big'.

Overflow integer errors can also occur using 2's complement. Assume we have a 4 bit 2's complement format. Assume the operation $0111_2 + 0111_2$ was undertaken. What would happen?

The resulting binary addition produces the bit pattern 1110_2 , which is, of course, negative! In this

case the bits from the carry have overflowed into the -8 place value position.

Overflow floating point errors occur when we try to use more bits than are available to store a number. Floating point overflow errors occur if numbers with either the mantissa or exponent are too large to fit into the allocated bits.

In the previous section we used a 10 bit format with 6 bits for the mantissa and 4 bits for the exponent. The allowable range of possible numbers ranged from -124 to +124. Numbers outside this range cannot be represented using this format.

In this example, if we tried to store $160.5_{10} = 10100000.1$. Written in normalized format we have 0.101000001×2^8 or $0.101000001 \times 2^{1000}$.

The mantissa requires 9 bits and the exponent instead of being +8 is in fact -8!

Overflow errors are reported as exceptions in languages such as Java and can be trapped by the programmer, otherwise the program may fail.

Underflow errors occur when an attempt is made to store a number that is too small. Such attempts are not reported as errors but default to zero and can cause run-time errors such as 'divide by zero' or worse, they cause important calculations to become zero.

In our 10 bit format we allowed 4 bits for the exponent. If we tried to store a number that in normalised format required the binary point to move more than 8 places to the left we would get an underflow error occurring because we can only represent up to -8.

For example: store the number $0.1 \times 2^{-10}_{10} = 0.0000000001$. (Note the exponent is decimal-10 not binary 10). In this case the exponent cannot be represented and would require an exponent of 5 bits in 2's complement to represent the left shift of the point. The mantissa could be represented.

EXERCISE 4.13

1. Define the terms underflow and overflow error and give examples to illustrate your definition.
2. Define the term truncation error.
3. Using 2's complement, how is it possible to get an overflow error.
4. Using 16 bits with a 10 bit mantissa and 6 bit exponent in 2's complement format, find a number that would cause an overflow error and one that would cause an underflow error. Make sure you clearly demonstrate and justify why the error would occur.
5. How does Java handle overflow and underflow for either reals or integers?
6. In Java, how would you recommend detecting underflow and overflow in a program for integers or reals?



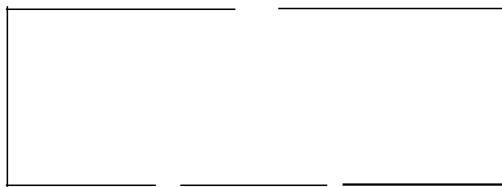
© IBO
2004 **4.2 BOOLEAN LOGIC**

© IBO
2004 **4.2.1 BOOLEAN OPERATORS**

Boolean logic is the basis of the design of the circuits used by computers. These circuits are constructed of inputs that enter a logic circuit to produce a single output. The logical operation of the circuits is governed by the rules of boolean logic.

Consider a simple circuit that involves a battery, light, switch and connecting cable.

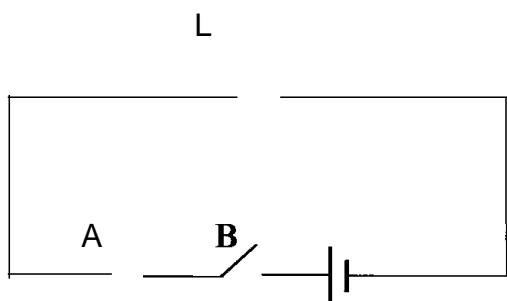
The light shines when the switch is closed. Conventionally a closed switch is represented by a 1 and an open switch by a 0. Further we denote the on state of the light to be represented by a 1 and the off state of the light to be represented by a 0. Both the switch and the light are 'two state' devices. They have only two states represented by a 1 or 0.



We can represent the logic of this situation by use of what is termed a truth table. The first column represents the possible state of the single input and the second column the state of the output i.e. 'state of the light'.

| Switch | Light |
|------------|---------|
| Open (0) | Off (0) |
| Closed (1) | On (1) |

Let us consider a case where there are two switches A and B. The circuit with the switches is shown in the diagram below. The light, denoted by L, is on only when both switches are closed i.e. $A = 1$ AND $B = 1$. All other combinations result in the light being in the off state. We refer to the logic displayed by this circuit as the 'AND condition'. Logic circuits are built using combinations of different standard conditions and we use special symbols to represent that logic. In this case we refer to the logic circuit as an 'AND gate'. It is termed a 'logic gate' because the two inputs must pass through the logic of the circuit. The logic can be represented by the following truth table, which also shows the special symbol used to show an AND gate. In the AND gate the switches are in series.



The AND Gate

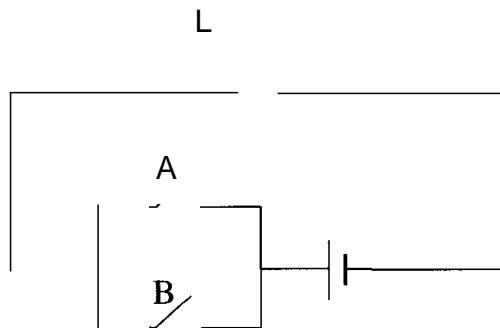
| A | B | L | AND gate symbol |
|---|---|---|--|
| 0 | 0 | 0 | |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | |
| 1 | 1 | 1 | |
| | | | A AND B written as A.B As a boolean expression we have L = A • B |

A truth table (shown as columns A, Band L) shows the possible combination of inputs and the outputs resulting from these inputs after they have passed through the logic of the gate.

The AND gate can also be written using the AND boolean algebra operator to give what is termed the boolean expression for the AND gate i.e. $L = A \cdot B$. The dot indicates AND.

The OR Gate

Consider the circuit below. In this circuit, the light is on if either A or B is closed or both are closed at the same time. This is a fundamental logic circuit referred to as the 'OR gate'. The switches are in parallel.



Its truth table is as follows.

| A | B | Output F |
|---|---|----------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

The symbol for the OR gate is:



The boolean algebra operator for OR is the $+$ sign. Thus the OR gate boolean expression is stated as $f = A + B$. This is read as f is true when A or B is true.

Note the use of the word 'TRUE' in the above sentence. We refer to the 1 state as representing TRUE and the 0 state as representing FALSE. Thus we get a true state from the OR gate in three ways and only one way when using the AND gate.

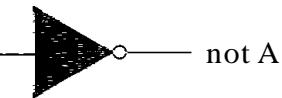
The NOT Gate

The NOT gate simply changes the state of an input. Thus if an input is TRUE the NOT gate outputs the reverse i.e. FALSE. If the input is FALSE the output is TRUE.

The truth table is as follows.

| A | Not A (written \bar{A}) |
|---|----------------------------|
| 0 | 1 |
| 1 | 0 |

The symbol for the NOT gate is as shown:



The boolean expression for a NOT gate is to use the bar over the letter denoting the input.

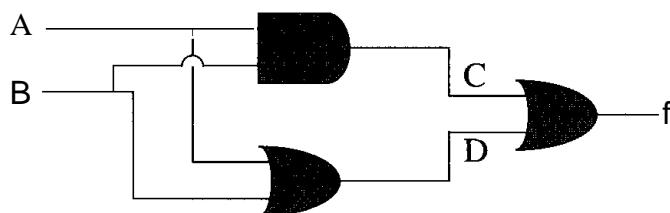
Thus \bar{A} denotes 'not A' in a boolean expression (Note: A' is also commonly used).

Drawing a truth table for a circuit.

We have three logic gates that we can use and if we use the same inputs into each we can put the three gates together in different ways to form 'circuits'. The inputs to each circuit are either on the A input line or B input line. We can run lines off the main input line by using a 'branch'. Thus we could form a circuit using an AND gate and OR gate leading into a single OR gate. By labelling the output of the first AND gate as C and the output of the first OR gate as D and labelling the output of the last OR gate as f, we can trace the logic of the circuit using a truth table.

The logic circuit diagram below shows the circuit for $f = (A \text{ and } B) \text{ or } (A \text{ or } B) \text{ or }$

$$f = (A \cdot B) + (A + B)$$



The truth table has two main input columns, A and B. The truth table then has two more columns representing the outputs from the AND gate as C, and from the OR gate as D. Both inputs from C

and D are fed into the last OR gate and the final column denotes the logical output of the entire circuit that comes out on the final line labelled f.

The truth table is as shown below:

| A | B | A-B C | A+B D | A-B + (A + B) C + D F |
|---|---|----------|----------|-----------------------------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

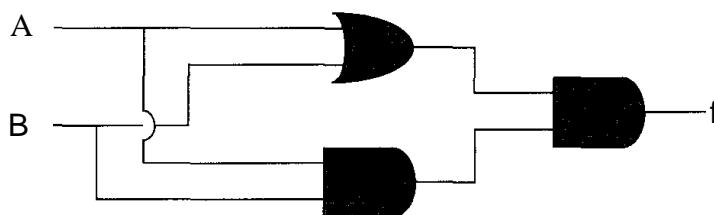
The logic circuit can also be described using boolean algebra to denote the output at line f as:

$$F = A - B + (A + B)$$

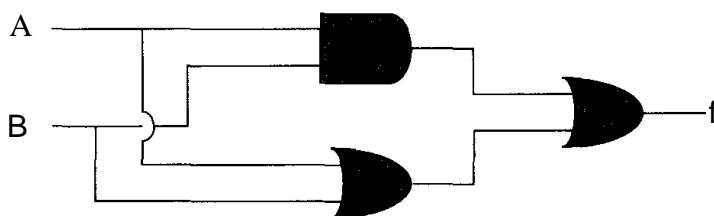
Thus f is true when the boolean expression is true and this occurs when either A or B is true. Thus the circuit behaves as the simpler OR gate circuit. This is an important result. Where two circuits have the same set of outputs for the same set of inputs we can use either circuit, as both circuits are said to be 'equivalent' or 'equal'. It makes sense to pick the simpler circuit with the minimum number of gates i.e. the 'minimised circuit'.

EXERCISE 4.14

1. Draw a truth table for the circuit shown below.



2. Draw a truth table for the circuit shown below.



3. For each of the circuits above, write down the boolean expression as shown in the example.
 4. For each circuit shown, can you determine a circuit that is the same but uses fewer gates?



Number Systems

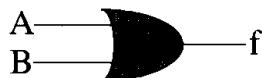
Further logic circuits

There are three other gates that are commonly used and we will consider each in sequence.

The exclusive OR gate XOR

The XOR gate produces a TRUE output only if one of the inputs is TRUE but not if both are true.

The XOR gate is denoted by the \otimes symbol and by the gate symbol as shown below.



The basic truth table for the exclusive OR is shown below.

| A | B | $A \otimes B$ |
|---|---|---------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

We can determine the boolean expression for a certain circuit by looking at the input patterns that produce a TRUE output. In the case of the 'exclusive OR' there are two patterns of inputs for A and B that produce a true output. The first pattern is not A and B. The second pattern is A and not B. Putting the two together we can say that we get a TRUE output when:

$$(\text{not } A \text{ and } B) \text{ or } (A \text{ and not } B) \text{ is true.}$$

Using boolean algebra symbols we can write the exclusive OR as the boolean expression:

$$f = \bar{A} \cdot B + A \cdot \bar{B}$$

The XOR is used to minimise circuit design by using the XOR gate to replace logic patterns that behave as f above.

EXERCISE 4.15

1. A circuit that is related to the XOR is the 'coincidence circuit'. Draw a truth table for a situation where a true value is output only when the inputs are the same i.e. $A = 0, B = 0$ gives 1, as does $A = 1$ and $B = 1$.



The NOT AND gate (NAND gate)

The 'NAND gate' is derived by inverting the output of the AND gate by applying the NOT operator.

The truth table for a NAND gate is shown below.

| A | B | A and B $A \cdot B$ | A and B $\bar{A} \cdot \bar{B}$ |
|---|---|------------------------|------------------------------------|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

The logic gate symbol for the NAND gate is shown:



In boolean algebra the not symbol is the bar and thus the boolean algebra expression for the NAND is shown as: $A \cdot B$

The NOT OR gate or NOR gate

The 'NOR gate' is the inverse of the 'OR gate'. Its basic truth table is shown below and the boolean algebra symbol is denoted as shown using the bar notation $A + B$.

| A | B | $A + B$ | $\bar{A} + \bar{B}$ |
|---|---|---------|---------------------|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

The logic symbol for the NOR gate is shown:



EXERCISE 4.16

1. Does $\bar{A} \cdot \bar{B} = \underline{A} \cdot \underline{B}$?

Step 1: Construct a logic circuit for $\bar{A} \cdot \bar{B}$.

Step 2: Construct a truth table for the circuit.

Step 3: Construct a logic circuit for $A \cdot B$

Step 4: Construct a truth table for the circuit.

Compare the final column of each truth table. What do you notice?

Number Systems

2. Apply the above method to determine whether or not the logic expressions $\bar{A} + \bar{B}$ and $A + B$ are equivalent.
3. Compare the outcomes of the above circuits to the circuits for the NAND and NOR gates. Do you notice any equivalences?



© IBO 2004 4.2.2 CONSTRUCTING BOOLEAN EXPRESSIONS

In the examples shown above, we have seen that boolean expressions can be written by collecting the combinations of inputs that produce a true output. We will see shortly that there exists a range of boolean logic laws that allow us to simplify these circuits.

Boolean expressions use only the boolean logic symbols and conventions. Some practice is required to become comfortable with this way of constructing boolean expressions. Some examples will help.

EXAMPLE

Write the boolean expression for the circuit (A AND B) or (A AND NOT B).

SOLUTION

This would be written as $(A \cdot B) + (A \cdot \bar{B})$. Note that the brackets can be removed.

EXAMPLE

Write the boolean expression for the circuit (A XOR NOT B) and (C NOR D)

SOLUTION

This would be written as $(A \otimes \bar{B}) \cdot (C + D)$ (note that the brackets are required here).

© IBO 2004 4.3 DERIVING TRUTH TABLES FROM BOOLEAN EXPRESSIONS

In this section we firstly introduce the notion of a third input to a logic circuit. In considering a third input we now have 8 possible input combinations to consider. There are two options for each input i.e. true or false and this gives $2 \times 2 \times 2 = 8$ (not $2 + 2 + 2 = 6$) different combinations of inputs.

Assume that we had a circuit described by the boolean expression f as shown below:

$$f = (A \cdot B) + C, \text{ which is read } (A \text{ and } B) \text{ or } C.$$

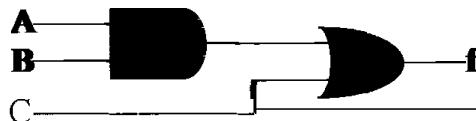
This expression has three inputs A, B and C. There are two gates i.e. AND and OR and the output comes from the OR gate.

A truth table can be constructed to show the logic of this circuit. Note that the layout of the Os and Is forms a pattern. You should copy this pattern. By doing this you will ensure that no combination of inputs is missed or repeated. The final column shows the output.

| A | B | C | A-B | A-B+C |
|---|---|---|------------|--------------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | I | 0 | 0 | 0 |
| 0 | 1 | I | 0 | 1 |
| I | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 0 | 1 |
| I | I | 0 | I | I |
| I | I | I | 1 | I |

We can also construct the logic circuit by noting that there are two inputs into an AND gate. The output from this gate and the input from C then flow into a final OR gate.

The circuit is shown below.



Notice also that there are 5 pairs of the three inputs that result in a TRUE (1) output. If we write these outputs combinations down, we get a complex logic circuit as the boolean expression. This long expression must be equivalent to the shorter boolean expression from which the truth table was derived. Note that each term in this type of boolean expression is referred to as a 'minterm'.

The longer expression is:

$$f = \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

and is known as a 'sum of products' form. This expression must be equivalent to: $f = (A - B) + C$

EXERCISE 4.17

Construct truth tables for the following boolean expressions

1. $f = (A + B) + (A - B)$
2. $f = A - B + A - C$
3. $f = A \cdot \bar{B} + B \cdot C + \bar{C} \cdot A$
4. $f = A \cdot B + B \cdot C + \bar{A} \cdot B$



© IBO 4.2.4 SIMPLIFYING BOOLEAN EXPRESSIONS
2004

The aim of circuit building is to build circuits that have the minimum number of gates. In the above situation the longer expression has six AND gates, three NOT gates and one OR gate. As compared to the shorter equivalent expression that has two gates. Obviously the latter is to be preferred.

The process of reducing the complexity of a circuit by reducing the number of gates is referred to as MINIMISATION.

There are two ways of approaching minimisation. The first way is to apply the rules of boolean algebra to minimise the boolean expression. The second alternative is to use Karnaugh maps.

The laws of boolean algebra.

Commutative laws

$$A + B = B + A$$

$$A - B = B - A$$

Associative laws

$$A + (B + C) = (A + B) + C$$

$$A - (B - C) = (A - B) - C$$

Distributive laws

$$A - (B + C) = A - B + A \cdot C$$

$$A + (B - C) = (A + B) - (A + C)$$

Tautology laws

$$A - A = A$$

$$A + A = A$$

$$A + A = I$$

$$A - A = 0$$

Absorption Law

$$A + (A - B) = A$$

$$A - (A + B) = A$$

Identities

$$0 - A = 0$$

$$0 + A = A$$

$$A + 1 = 1$$

$$1 - A = A$$

$$A = A$$

Complement

$$A + \bar{A} - B = A + B$$

EXERCISE 4.18

1. For each result above, show that the equivalence is true by drawing a truth table.
2. Show that the following equivalence is true by drawing a truth table.

$$A + B \cdot C = (A + B) \cdot (A + C)$$

3. Show that the following equivalence is true by drawing a truth table.

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

4. Show that $A + A - B = A + B$
5. Show that $A - (A + B) = A - B$



More difficult worked examples.

EXAMPLE

$$\text{Show that } \bar{A} \cdot B + A \cdot \bar{B} + \bar{A} \cdot \bar{B} = \bar{A} + \bar{B}$$

SOLUTION

Use the commutative law to rearrange to get: $A - B + A - \bar{B} + A - \bar{B}$

Use the distributive law to get: $\bar{A} - (B + \bar{B}) + A - \bar{B}$

Use the inverse law ($x + \bar{x} = 1$) to give: $\bar{A} + A - \bar{B}$

Use the complement law to give: $A + \bar{B}$.

A technique commonly used to add some extra terms relies on the fact that $A + A = A$. This often allows factorizing that leads to simplification.

EXAMPLE

Simply this expression. This has four minterms and each term has a single NOT component.

$$A \bullet B \bullet C + \bar{A} \bullet B \bullet C + A \bullet \bar{B} \bullet C + A \bullet B \bullet \bar{C}$$

SOLUTION

Let us add $A \bullet B \bullet C$ between the 2nd and 3rd terms and the 3rd and 4th terms and then factorise. We now have:

$$A \bullet B \bullet C + \bar{A} \bullet B \bullet C + A \bullet B \bullet C + A \bullet \bar{B} \bullet C + A \bullet B \bullet \bar{C}$$

We can now group each pair of terms and use the fact that $x + \bar{x} = 1$ to minimise.

$$B \bullet C \bullet (A + \bar{A}) + A \bullet C \bullet (B + \bar{B}) + A \bullet B \bullet (C + \bar{C})$$

This gives $B \bullet C + A \bullet C + A \bullet B$, which is an equivalent minimised circuit.

EXERCISE 4.19

- For each of the examples above, show that the minimised circuit is equivalent to the original circuit by completing a truth table.



De Morgan's Laws

In some of the previous exercises you explored De Morgan's Laws. Here we state them formally. De Morgan's Laws are useful results that can often be used to simplify boolean expressions.

$$\text{De Morgan's Laws: } A + \bar{B} = A \bullet B \text{ and } A \bullet \bar{B} = A + B$$

To apply De Morgan's laws, follow this simple algorithm

- Take a term e.g. $A \bullet \bar{B}$
- NOT the individual members of the term e.g. $A \bullet B$
- Change the operator i.e. \bullet to $+$, or $+$ to \bullet . i.e. $A + B$
- NOT the entire term i.e. $A + B$

EXAMPLE

Using De Morgan's Laws show that $f = \bar{A} \bullet \bar{B} + (\bar{A} + \bar{B}) = A \bullet B$

SOLUTION

Note: we have two separate compound terms, which we treat as single terms.

Step 1: NOT the terms to get $(A - B) + (A + B)$

Step 2: Change OR to AND to get $(A-B) - (A+B)$

This reduces to $A \bullet B + A \bullet B$ i.e. $A \bullet B$

Step 3: NOT the lot to get $A - B$

We can confirm this result by use of a truth table

| A | B | \bar{A} | \bar{B} | $\bar{A} \bullet \bar{B}$ | f | $A \bullet B$ | $A - B$ |
|---|---|-----------|-----------|---------------------------|-----|---------------|---------|
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | |
| 0 | | 1 | 0 | 0 | | 0 | |
| | 0 | 1 | 0 | 0 | 1 | 0 | |
| | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Note that the last column matches the output column f , which shows that the two boolean expressions are equivalent.

EXERCISE 4.20

1. Using truth tables confirm that De Morgans's Laws are true for both basic cases.
2. Use De Morgan's laws to simplify the circuit: $f = \bar{A} + \bar{B} - \bar{C}$.

**Equivalence circuits**

Consider the following circuit that uses a NOR gate and only one input, A.



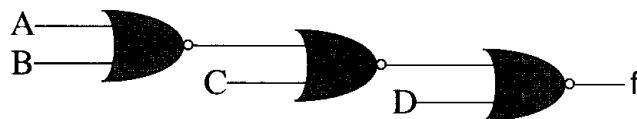
Number Systems

The truth table for this circuit is shown below.

| A | \bar{A} | $A\bar{A}$ | $\bar{A}\bar{A}$ |
|---|-----------|------------|------------------|
| 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

This circuit is equivalent to NOT A.

Consider this grouping of NOR gates with inputs A, B, C and D.

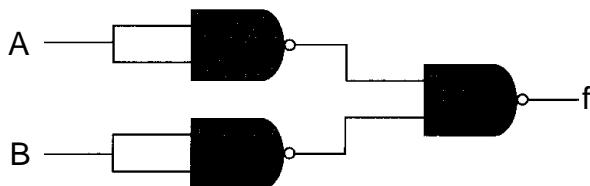


The truth table is as shown below. If you look at the last column you will see that it is equivalent to an AND gate.

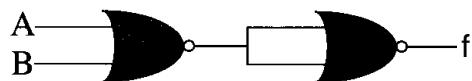
| A | B | C | D | f |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

EXERCISE 4.21

1. Construct a truth table for the following circuit and write down a circuit that it is equivalent to.



2. Use De Morgan's Laws to show that the equivalence result is true.
 3. Construct a truth table for the following circuit and write down a circuit that it is equivalent to.



Karnaugh maps

Karnaugh maps (hereafter termed K-maps) are a visual depiction of the logic of a circuit. They are useful because we can use them to group minterms together that have common elements which allows us to remove the other terms. This is done by grouping minterms into even number groups of cells by rows or columns, but not diagonally. We can group from the end of row one back to the start. The flat table is actually a cylinder folded out. When the groupings are made we look for common elements.

An example will help explain. In a three input circuit there are 8 possible inputs. Thus, in a K-map there are 2 rows and 4 columns, which gives 8 cells. The row labels are A and NOT A. The column headings are then B AND C, NOT B AND C, B AND NOT C, NOT B AND NOT C.

A basic K-map of three inputs (variables) is shown below.

| | B • C | $\bar{B} \cdot C$ | $B \cdot \bar{C}$ | $\bar{B} \cdot \bar{C}$ |
|-----------|-------|-------------------|-------------------|-------------------------|
| A | 1 | 0 | 0 | 0 |
| \bar{A} | 1 | 0 | 0 | 0 |

Note the placement of the Is and Os.

The corresponding truth table would look as shown below. Note the location of the 1s in the output column and their location in the K-map.

| A | B | C | Output | Miniterm |
|---|---|---|--------|---------------------------------------|
| 0 | 0 | 0 | 0 | $\bar{A} \cdot \bar{B} \cdot \bar{C}$ |
| 0 | 0 | | 0 | $\bar{A} \cdot \bar{B} \cdot C$ |
| 0 | | 0 | 0 | $\bar{A} \cdot B \cdot \bar{C}$ |
| 0 | | | | $\bar{A} \cdot B \cdot C$ |
| | 0 | 0 | 0 | $A \cdot \bar{B} \cdot \bar{C}$ |
| 1 | 0 | | 0 | $A \cdot \bar{B} \cdot C$ |
| | | 0 | 0 | $A \cdot B \cdot \bar{C}$ |
| | | | | $A \cdot B \cdot C$ |

From the truth table, the boolean expression is: $f = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$

From the K-Map we can circle the first column because B e C is common. Thus, the circuit can

Number Systems

be reduced to: $f = B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C}$. This result is consistent with the algebraic approach in that by use of boolean algebra we have:

$$\begin{aligned} f &= A \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} \\ &= B \cdot C(A + \bar{A}) \\ &= B \cdot C \end{aligned}$$

Other standard minimisations are shown with the following series of examples.

| | $B \cdot C$ | $\bar{B} \cdot C$ | $B \cdot \bar{C}$ | |
|---|-------------|-------------------|-------------------|---|
| A | 1 | 1 | 0 | 0 |
| A | 1 | 1 | 0 | 0 |

This shows $f = C$ as it is common to all cells.

| | $B \cdot C$ | $\bar{B} \cdot C$ | $B \cdot \bar{C}$ | $\bar{B} \cdot \bar{C}$ |
|---|-------------|-------------------|-------------------|-------------------------|
| A | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 0 | 0 |

This shows $f = A$ as it is common to all cells.

| | $B \cdot C$ | $\bar{B} \cdot C$ | $B \cdot \bar{C}$ | $\bar{B} \cdot \bar{C}$ |
|---|-------------|-------------------|-------------------|-------------------------|
| A | 1 | 1 | 0 | 0 |
| A | 0 | 0 | 1 | 1 |

This shows $f = A \cdot C + A \cdot \bar{C}$

| | $B \cdot C$ | $\bar{B} \cdot C$ | $B \cdot \bar{C}$ | $\bar{B} \cdot \bar{C}$ |
|---|-------------|-------------------|-------------------|-------------------------|
| A | 1 | 1 | 0 | 1 |
| A | 1 | 1 | 0 | 0 |

This gives $f = C + \bar{A} \cdot \bar{B} \cdot \bar{C}$

EXAMPLE

Use a K-map to represent the following truth table and from this show the boolean expression in its longest form and then use the K-map to simplify. Check by using boolean algebra that the minimised circuit is equivalent.

SOLUTION

| A | B | C | f | miniterm |
|---|---|---|---|-------------------------------------|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | | | $\bar{A} \bullet \bar{B} \bullet C$ |
| 0 | | 0 | 0 | |
| 0 | | | | $A \bullet B \bullet C$ |
| | 0 | 0 | 0 | |
| | 0 | 1 | | $A \bullet \bar{B} \bullet C$ |
| | | 0 | 0 | |
| | | | | $A \bullet B \bullet C$ |

$$f = \bar{A} \bullet \bar{B} \bullet C + \bar{A} \bullet B \bullet C + A \bullet \bar{B} \bullet C + A \bullet B \bullet C$$

As a K-map we have

| | $B \bullet C$ | $\bar{B} \bullet C$ | $B \bullet \bar{C}$ | |
|---|---------------|---------------------|---------------------|---|
| A | 1 | 1 | 0 | 0 |
| A | 1 | 1 | 0 | 0 |

From the K-Map we can circle the 1s and see that C is common. Thus the equivalent circuit is $f=C$.

If you look at the truth table you can see this is the case as we get a 1 if and only if C is true. Thus it is irrelevant what the other inputs are.

By applying the rules of boolean logic to the first equation we can show that the circuits are equivalent.

$$\begin{aligned}
 f &= AeBeC + AeBeC + AeBeC + AeBeC \\
 &= A e C e (B + \bar{B}) + \bar{A} e C e (B + \bar{B}) \\
 &= AeC + AeC \\
 &= C e (A + A) \\
 &= C \text{ which is shown in the K-map}
 \end{aligned}$$

EXERCISE 4.22

1. Use a K-map to simplify the following truth table.

| | $B - C$ | $\bar{B} - C$ | $B \bullet \bar{C}$ | $\bar{B} \bullet \bar{C}$ |
|-----------|---------|---------------|---------------------|---------------------------|
| A | 1 | 1 | 1 | 0 |
| \bar{A} | 1 | 0 | 0 | 0 |

- a. Write down the full boolean expression.
- b. Now simplify this expression using the K-map.
- c. Show that the full boolean expression can be reduced to the minimised form. (Hint: you may have seen this earlier!)



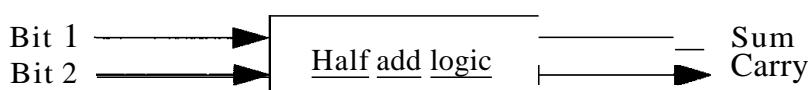
© IBO 2004 **4.2.5 CONSTRUCTING LOGIC CIRCUITS FROM BOOLEAN EXPRESSIONS**

The HALF adder

If we consider the adding of two binary bits we can see that there are four specific combinations. In base 2 we have the following:

$0+0 = 0$ or $0 + 1 = 1$ or $1 + 0 = 1$ or $1 + 1 = 0$ with a carry of 1.

Such a system can be described as a logic circuit with input for bit 1 (A) and bit 2 (B) and two outputs Sum (S) and carry (C). A block diagram is shown below.



From these rules we can determine a truth table as shown below

| Bit 1 | Bit 2 | Sum | Carry |
|-------|-------|-----|-------|
| A | B | S | C |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

The circuit is in fact two separate circuits combined.

The sum is an XOR and the Carry as an AND gate. Thus the boolean expression for the sum is given by:

$$f = \bar{A} \cdot B + A \cdot \bar{B} \quad (1)$$

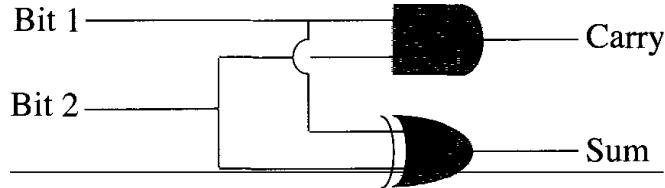
$$f = A \otimes B \quad (2)$$

The circuit for (I) is shown using two NOT gates, two AND gates and one OR gate. This circuit is simplified by using the XOR gate symbol as shown below.

The boolean expression for the carry is simply an AND gate with inputs from A and B.

Putting the XOR gate and AND gate together we have the logic circuit for the adding of two bits.

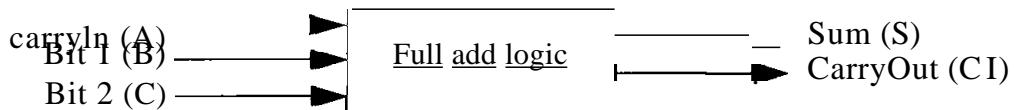
A diagram is shown below.



The FULL ADDER

The full adder must take into account the possibility of a carry coming in as well as going out.

Hence there are three inputs: bit1(B), bit2(C) and carryIn(A) and two outputs sum(S) and CarryOut(CI).



Number Systems

A truth table can be constructed as shown below.

| A | B | C | S | CI |
|---|---|---|---|----|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Reading from the truth table we can derive the boolean expression for both S and CI as:

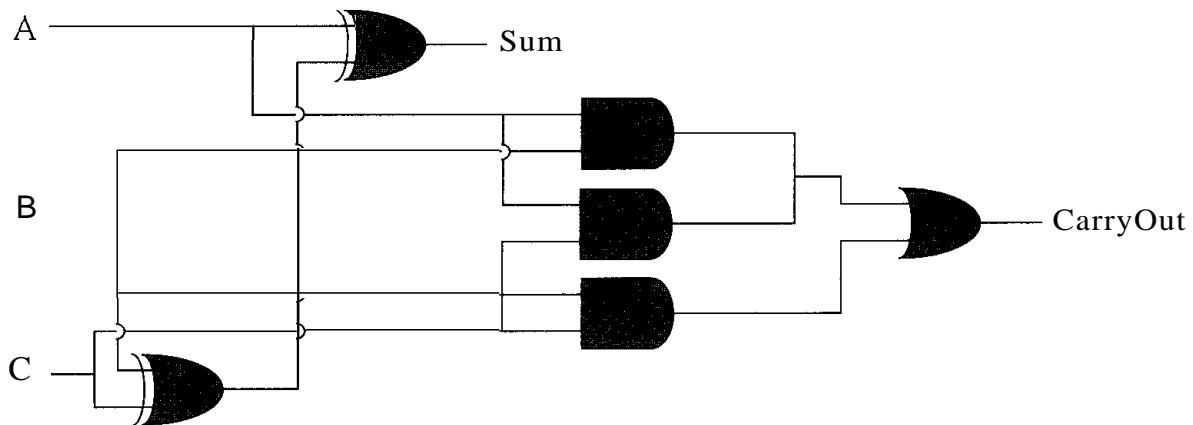
$$S = A \cdot B \cdot C + \bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot C + A \cdot B \cdot C , \text{ which reduces to}$$

$$S = B \otimes (C \otimes A)$$

$$CI = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C , \text{ which reduces to}$$

$$CI = A \cdot B + A \cdot C + B \cdot C$$

The logic circuit for each can now be drawn and is shown in the diagram below.



MINIMISATION PROOFS

Boolean Algebraic Proofs for the Minimisation of the Sum (S) and CarryOut (CI) boolean expressions.

CARRY CIRCUIT MINIMISATION

$$CI = A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C$$

Add in two more $A \cdot B \cdot C$ terms and group as shown:

$$CI = B \cdot C \cdot (A + A) + A \cdot C \cdot (B + B) + A \cdot B \cdot (C + C)$$

$$CI = B \cdot C + A \cdot C + A \cdot B$$

SUM CIRCUIT MINIMISATION

$$S = \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot C + A \cdot B \cdot \bar{C} \quad (1)$$

$$S = B \text{ xor } (C \text{ xor } A) = \bar{B} \cdot (C \cdot \bar{A} + \bar{C} \cdot A) + B \cdot (\overline{C \cdot A + A \cdot C}) \quad (2)$$

Step I: Taking minterms 1 and 3 from equation (1) gives

$$\bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot \bar{C} = \bar{B} \cdot (\bar{A} \cdot C + A \cdot \bar{C}), \text{ which corresponds to the first minterm in (2)}$$

Step 2:

Minterms 2 & 4 from (1) must be equivalent to the second term of (2)

Grouping the 2nd and 4th minterms from (1) gives:

$$\bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot C = B \cdot (A \cdot C + A \cdot \bar{C})$$

The B terms match.

Now apply De Morgan's Law to $(A \cdot C + A \cdot \bar{C})$

$\bar{A} \cdot \bar{C}$ gives $A + C$ and then not the lot to give $A + C$

$A \cdot C$ gives $\bar{A} + \bar{C}$ and then not the lot to give $\bar{A} + \bar{C}$

This gives $A + C + \bar{A} + \bar{C}$ and we apply De Morgan's Law again.

$(A + C) \cdot (\bar{A} + \bar{C})$ and NOT the lot to get:

$\overline{(A + C) \cdot (A + C)}$ which gives $\overline{(A \cdot C + A \cdot \bar{C})}$

Adding in the B terms we get the term we require $B \cdot (\overline{A \cdot C + A \cdot \bar{C}})$, which matches the required minterm (QED)

© IBO 2004 4.2.6 CONSTRUCTING BOOLEAN EXPRESSIONS FROM LOGIC CIRCUITS

This is the process of reversing what we have been doing.

A boolean expression such as $f = A \cdot B + (C + B)$ states the following:

There are three inputs A, B and C

One AND gate for A AND B

Number Systems

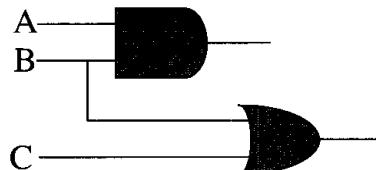
One OR gate for C OR B

One OR gate which accepts input from the above mentioned and gate and or gate.

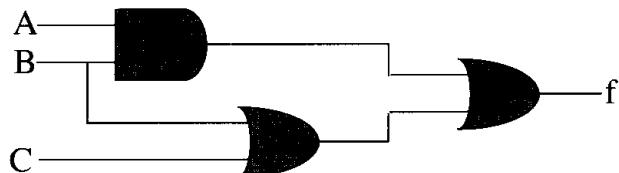
To build the circuit, start by drawing the first AND gate:



Now add the third input C and take a line from B and run them through an OR gate.



To complete take the outputs from both gates and run them through a final OR gate. The final circuit is shown below.



The above discussion shows how to derive a logic circuit from a Boolean expression. To construct a Boolean expression from a logic circuit you apply a similar deconstruction-construction technique.

In the logic circuit we note the following:

- (a) there are three inputs
- (b) Three logic gates: an AND, and two OR gates.
- (c) The second OR gate accepts inputs from the other AND and OR gate and produces the output on line f. We have two sub-circuits leading into the final OR gate.

This last point gives the basic template structure i.e. sub-circuit! OR sub-circuit 2

We now work out what each sub-circuit is made of.

Sub-circuit I is A AND B ie $A \cdot B$

Sub-circuit 2 is A OR B ie $A + B$.

Thus, filling out the template we have $A \cdot B \text{ OR } A + B$.

Finally we have $(A \cdot B) \cdot (A + B)$

© IBO 2004 **4.2.7 EXPLAIN THE FUNCTION OF SPECIFIC CIRCUITS**

EXAMPLE

A device is attached to a computer interface card that has three connections A, B and C. The device has a small fan attached and this fan is to be turned on when the connections from the interface card have the decimal values of 1, 3, 4, 5 or 7. These values are stated in the table below.

Draw a truth table and the full boolean expression for the above. From the K-map derive a minimised form and from this boolean expression, derive the logic circuit. Show also that the full boolean expression is equivalent to the minimised form.

SOLUTION

Truth table

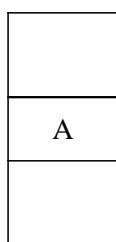
The inputs to the truth table form a 3 bit pattern Le. 0,0,0 represents decimal 0 and 1,1,1 represents decimal 7.

| decimal | A | B | C | f |
|---------|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1* | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 3* | 0 | 1 | 1 | 1 |
| 4* | 1 | 0 | 0 | 1 |
| 5* | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7* | 1 | 1 | 1 | 1 |

Full boolean expression:

$$f = (A \cdot B \cdot C) + (A \cdot \bar{B} \cdot C) + (\bar{A} \cdot \bar{B} \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot C)$$

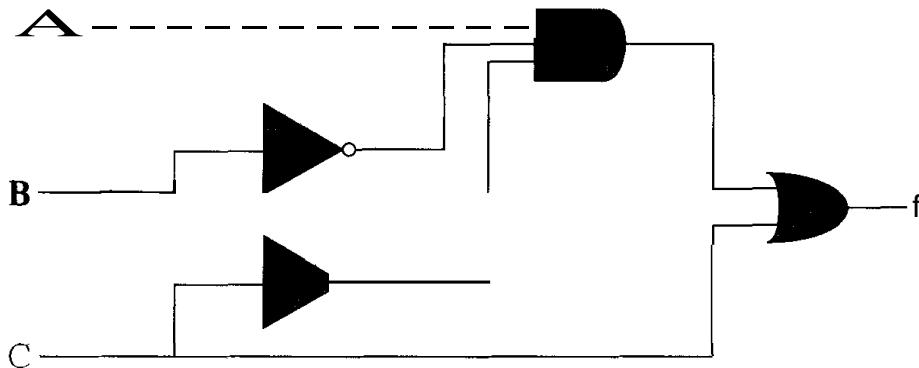
As a K-Map we have.



From this, we can see that the circuit minimises to $f = C + A \cdot \bar{B} \cdot \bar{C}$.

Number Systems

The logic circuit is therefore as shown below



The minimised circuit can be shown to be equivalent by the following process.

$$\begin{aligned}
 f &= A \cdot B \cdot C + A \cdot \bar{B} \cdot C + \bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot C \\
 &= B \cdot C \cdot (A + \bar{A}) + \bar{B} \cdot C \cdot (A + \bar{A}) + A \cdot \bar{B} \cdot \bar{C} \\
 &= B \cdot C + \bar{B} \cdot C + A \cdot \bar{B} \cdot \bar{C} \\
 &= C(B + \bar{B}) + A \cdot \bar{B} \cdot \bar{C} \\
 &= C + A \cdot \bar{B} \cdot \bar{C}
 \end{aligned}$$

EXERCISE 4.23

1. A house alarm is located in a hallway. It sounds if power is on and either movement is detected or a sound is detected in the hallway.
 - a. What are the inputs to this logic system?
 - b. How many gates are required?
 - c. Construct a truth table.
 - d. From the truth table, construct a full boolean expression.
 - e. Is the boolean expression in minimised form?
 - f. Construct a logic circuit.
 - g. Test that the logic circuit is correct by checking using a truth table.

2. A robot is controlled by two motors that drive two wheels. When the left motor is on and the right motor off, the robot turns right and the reverse to turn left. The robot has three sensors used to detect the proximity of things around it. Assume that the robot is moving forward. If the robot detects something blocking its path on the left it turns to the right and vice versa. If the robot senses something in front, it stops. Otherwise it moves forward.
 - a. Draw a truth table for the control of the robot by the motors.
 - b. Derive and simplify a boolean expression from this truth table.
 - c. Draw a logic diagram for one of the motors.



5

Chapter contents

- 5.1 Terminology
- 5.2 Static data structures
- 5.3 Dynamic Data Structures
- 5.4 Objects in problem solutions
- 5.5 Recursion
- 5.6 Algorithm evaluation

This chapter covers the material in Topic 5 of the **HL** IE Computer Science Syllabus. The chapter follows the syllabus in the exact order it is specified and begins with a discussion of terminology.

© IBO 2004 **5.1 TERMINOLOGY**

© IBO 2004 **5.1.1 DEFINITIONS**

Identifier: an identifier is a name given by a programmer to represent a variable, class name, method name, data type or any other element defined within the program.

Operand: an operand is a name given by programmers to represent named memory locations and objects that will be manipulated by the program. For example, a variable is an identifier that is also an operand as it names a memory location. For example $S + 45$ contains the operand S , which is a variable, and 45 . Another way to think of an operand is 'that to which an operation is applied' i.e. S and 45 have the addition operation applied to them.

Operator (unary and binary): operators indicate the action to be applied to operands in an expression. Thus, an operator is a character or string of characters which designate an operation. Operators can be classified e.g. arithmetic operators indicate the normal operations of arithmetic: $+$ (plus), $-$ (minus), $/$ (division), $*$ (multiplication); Boolean operators include: not, and, or, xor.

Unary Operator: special operator that requires only one operand e.g. negation in boolean expressions or $++x$ in C and Java programming languages.

Binary Operator: operator that requires two operands e.g. $+$ when applied to two operands $x + y$.

Actual parameter (Argument): a data value passed between a calling program segment and the called segment. When a method is called, you often give the method some data either as a constant or as a variable e.g. $\sin(x)$. The quantity inside the brackets is referred to as an argument, hence the term 'method's arguments'. Many methods require more than one argument and hence the term argument list e.g. $calc(x, t, y)$. The list x, t and y make up the argument list.

Formal Parameter: refers to the variable names in the method or class declaration header e.g. `public int calc(int a, int b, String v)`. In this case the variables a , b and v are parameters of the method `calc`. The parameters of the method get their values from the arguments of the method call. Parameter values can be passed-by-value. The values of the arguments are passed or copied to the parameters of the method, i.e. the argument values initialise the parameter variable.

In Java, the values of primitive data types such as `int` and `double` are always passed-by-value. Any changes made in the called method are not made to the original data values as they are completely separate memory locations.

In Java the value of an object's reference is also passed. In the IE and JETS this is interpreted as the equivalent of pass-by-reference.

The difference between pass-by-value and pass-by-reference is that in pass-by-reference the data values are not copied, rather the original locations are referenced. Hence, if these are changed, they are changed in the original locations.

In Java the value of the object reference is passed e.g. when passing arrays, Java classes or user defined objects. This acts like pass-by-reference. Primitive data types can't be passed using pass-by-reference in Java, however, wrapper classes such as `Integer()` can be used if this feature is desired.

In the following example we have a method called equation defined, that is called, and a value is returned and assigned to the variable p .

PARAMETER PASSING EXAMPLE

Line numbers have been added to aid the explanation.

```

1. int x=10, y=2, p=0;
2. int n[] = {23,34,12}
3. int p = equation(x,y,n)
4. String outline = "value of p    "+p+" Value of n[ y]    "+n[ y] );
5. output(outline)

6. int equation (int m, int b, int c [] )
{
7.     m = b + m;
8.     c[ b] = m + b;
9.     return m;
}
```

Explanation of the example.

At line 3 the method equation is called and the argument values of x, y and the array reference value are passed. Note that arrays are 'objects' and not 'primitives' in Java.

At line 6 the following happens:

The value of variable x is assigned (copied) to initialise the parameter m in the method. Thus m holds 10.

The value of variable y is assigned to initialise the parameter b in the method. Thus b holds 2.

The value of the reference of array $n[]$ is passed to the array variable parameter c . Thus $c[]$ references the values {23,34,12}.

At line 7 the method variable m is assigned the value $2 + 10$ i.e. 12.

At line 8 $c[2]$ is assigned the value $12 + 2 = 14$. As this is a reference the value of $n[2]$ back in the calling part of the program is also changed! The array values of $n[]$ and $c[]$ are {23, 34, 14}.

At line 9 the value of m i.e. 12 is returned and assigned to the variable p at line 3.

At line 5 the following is output: value of $p = 12$. Value of $n[y] = 14$.

Infix notation: An infix notation is a notation for representing operations where the operator is placed between the two operands. It is to operate on e.g. $a+s+(t*y)$. It is the normal way that equations in mathematics are written and evaluated.

Postfix notation also known as reverse polish notation. A postfix notation is where each pair of operands is followed by its operator. e.g. $xy+c^*$ is equivalent to the infix notation $(x+y)*c$.

Prefix notation: A prefix notation is where each pair of operands is preceded by the operator. e.g. $*+xyc$ is the prefix equivalent of $(x+y)*c$.

EXERCISE 5.1

1. Using your chosen computer programming language list some examples of an identifier.
2. In the following expressions, state the operators and operands used.

$c = (t+y)-4*a+b$

$d= ++c*(d-t)$

3. In the above expressions, state the binary and unary operators used.
4. In the following function call and definition, state the following:

- a) the function identifier name.
- b) the arguments.
- c) the formal and actual parameters.
- d) which variables are passed by value and which are passed by reference.

```
int fred (int a, int b, int d[])
{
    int c = 0;
    for (int i=0; i<d.length; i++)
        if (d[ i ] >a && d[ i ] <b)
    {
        c = c + 1;
    }
    return c;
}
```

The call to method is as follows: note $f = 10$ and $d = \{23, 67, 12, 45\}$.

`output ("Result = "+fred(f, 20, d))`

- e) What is output?
- f) Was it necessary to actually return the value of c ? If not why not?
5. In what format is this expression: $(a*c)-(d/23)$?
6. What type of format is reverse polish?
7. Change the expression $(a*c)-(d/23)$ to reverse polish format?
8. Write the following expression in postfix and then prefix format.

$x*y+(c-j)*g$
 $s+f-a*f+e/f$

9. Evaluate this expression, which is in postfix notation: $23\ 14\ +\ 12\ 12/\ +$
10. Convert the following from reverse polish to infix notation

$xy+c^*$



©IBO 2004 5.1.2 DEFINITION OF A STACK, QUEUE AND BINARY TREE.

Recall that a data structure defines the way the data is organised in the computer's memory so that it can be conveniently accessed and processed by the program. In Chapter I you were introduced to the notion of a LIST data structure i.e. an array. In a LIST data structure the data is stored in a sequence and is of the same data type.

In this section we introduce three separate LIST data structures, the stack, the queue and binary tree, which are all very useful and which behave in different ways.

STACK DATA STRUCTURE

Think of a simple pile of plates. Normally you place a plate onto the pile at the top and also remove a plate from the top.

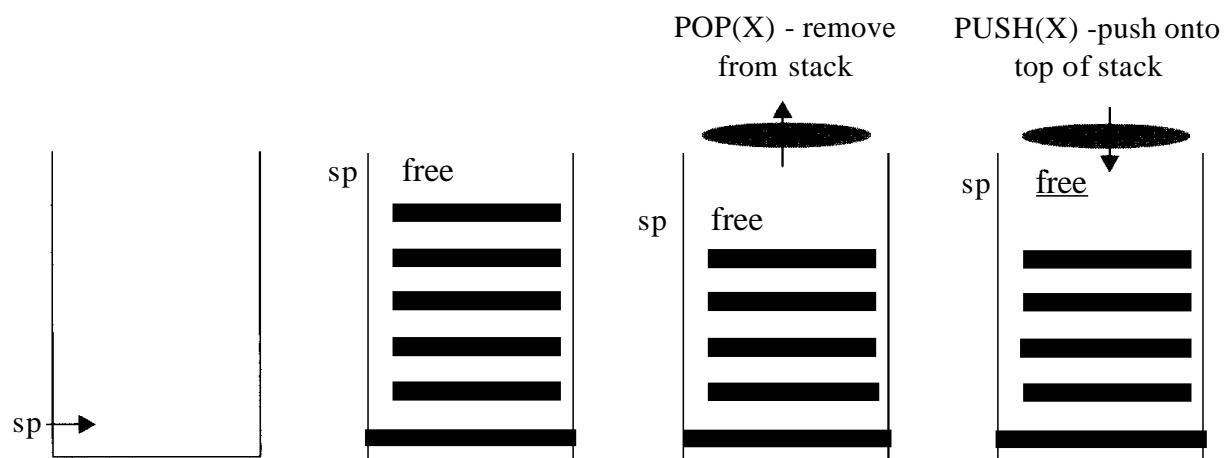
Another simple every-day example is that of a pile of coins. Take 10 coins and place them one on top of one another. You will notice as you add coins to the pile you do so by placing the next coin on top of the last coin i.e. on the TOP. If you remove a coin you can only do so from the TOP without either splitting the pile or possibly causing it to topple over!

Such a pile of coins or plates can be thought of as a STACK.

A stack is a data structure with the following characteristics:

- Data is pushed onto the top of the stack in the same way that plates are stacked on a table.
- Data is removed from a stack by taking off the top element. Data is said to be accessed in a Last In First Out (LIFO) manner. Continuing our plates stack example, we can only access the top plate and to get access to any other plates we must remove those plates that are stacked on top.
- A stack pointer variable is used to point to the next free space in the stack.
- Stacks implemented using an array have a fixed maximum size that cannot be exceeded.
- Stacks implemented as linked lists, which are dynamic are limited only by the available memory.
- It is illegal to try to PUSH onto a full stack and to POP from an empty stack.

Figure: 5.1- Stack (LIFO)

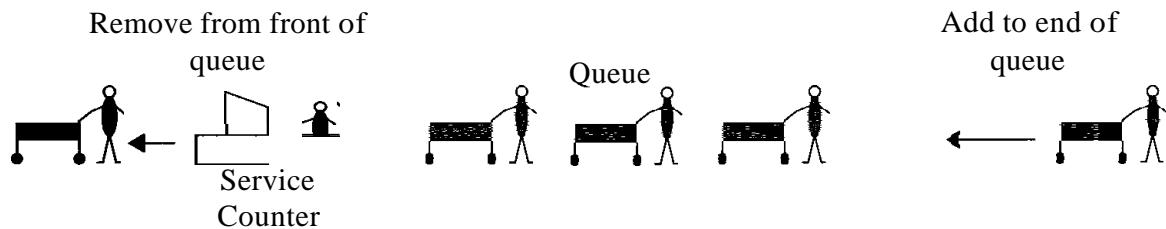


QUEUE DATA STRUCTURE

Next time you line up at a service counter of some kind, such as an ATM or customer service counter, you are joining a queue. The queue is also a LIST structure, but it has different characteristics from the stack. In a queue, items are added to the tail of the queue not to the top, as is the case with a stack. Things are removed, or serviced, from the head of the queue.

A queue is a data structure with the following characteristics:

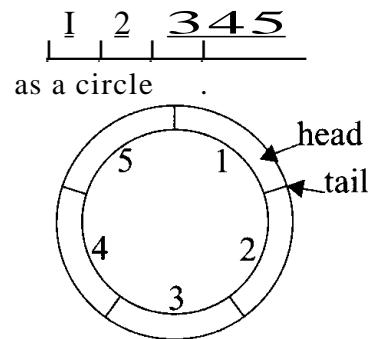
- Data is added to the end of the data list in the same way that you queue in line at a service counter.
- Data is removed from the front of the queue. We say that data is accessed from a queue in a First In First Out (FIFO) manner.
- A queue head and tail variable is used to point to the current head or tail of the queue.
Queues implemented using an array have a fixed size which cannot be exceeded, otherwise an error will occur.
- Queues implemented using a dynamic linked list are limited only by available memory.
- It is illegal to try to remove data from an empty queue or to add it to a full queue.



It is common to view queues that are implemented in a static array as a circular structure. Two variables are used to keep track of the head and tail of the queue.

The circular view of a queue requires that two properties are monitored.

1. If the head is made equal to the tail of the queue after an item is removed, the queue is empty.
2. If the tail is equal to the head after an addition, the queue is now full.



OPERATIONS ON STACKS AND QUEUES

We can add and remove from both a stack and a queue. But there are some efficiency processing problems associated with how we actually store the queue as a static array. At this stage this is the only way you know of storing a list, as an array.

We can implement a stack easily using an array and a pointer to the head or top of the list. As items are added we incur an insertion to the list and an update to the head pointer. As we remove, we incur an access to the list to get the data item and then a decrement to the head pointer. However, this is not the case with a queue if we use an array.

EXERCISE 5.2

1. Define the data structure referred to as a stack.
2. Define the data structure referred to as a queue.
3. We are going to undertake these operations on a queue: A = Add and R=Remove.

Assume the queue is an array and it contains only three (3) places and is called QUEUE[index]

Assume that we fix the head of the queue to be always the first element in the array. i.e. QUEUE[0]

Trace the following operations and draw the state of the queue at each stage.

A 12, A 13,A,23,R,R,

4. What inefficient processing occurs at each stage by implementing a queue as a simple linear structure? How might you overcome these problems?



DEFINITION OF A BINARY TREE

A binary tree stores data in nodes that allow branches to the left or right according to a greater than or less than type rule. For example, if storing integers in nodes of a binary tree, the left branch holds integers that come before the current integer and the right branch holds integers that are greater than the current integer.

A binary tree allows lists of data to be searched efficiently in a similar way to that of a binary search. It allows efficient searching of a list stored as a sorted array.

Binary trees are used in a range of applications such as:

Storing of a file index in RAM to enable a randomly organised file to be directly accessed using the key as an index into the relative record position of the record in the file.

Further details about Binary trees are covered in the dynamic data structures section.

© IBO 5.1.3 USES OF STACKS 2004

The primary use of a stack is to keep track of things in the order that they occur so that you can reverse back through them to get back to where you started.

A stack data structure is used by web browsers to keep a current list of the sites visited. If you view this list on your browser, it is a stack. The last site visited is on top of the list and the first site you visited will be last. When you click the back button you are causing a POP operation and the web address on top of the stack is taken off and accessed. When you click on a new site, the site you are leaving is PUSHed onto the 'sites visited' stack.

Some other common uses of stacks are:

1. Parameter storage and subprogram return addresses.

When a program calls another program or function, the called function can also call another function. When the first program is called, the current program needs to be temporarily suspended and its return address and the values of the parameters it is using kept so they can be used when

the called function returns. To do this, the return address and the contents of the parameters are placed onto a stack in memory. As subsequent calls are made, the next return address and set of parameters are PUSHed onto the stack.

When a function stops executing, the return address is POPped off the stack and the program continues. This chain of events is represented below. It shows program 1 calling function 1, which calls function 2, which then calls function 3.

The return address of the program is placed onto the stack and then, as the other functions are called, their return addresses are added to the stack.

```
Program 1
Call Function 1 //Push return address of Program 1 onto stack
    Call Function 2 //Push return address of Function 1 onto
    stack)
        Call Function 3 //Push return address of Function 2 onto
        stack)
            End Function 3 return
                //Pop function 2's return address from stack)
            End Function 2 //Pop function 1's return address from stack)
        End Function 1 //Pop return address of Program 1 from stack)
    Continue process in program 1
```

2. Interrupt handling

Interrupts work in the same way. When the CPU detects interrupt operations, the current state of the computer must be saved. The contents of variables and return address are placed onto the stack. When the interrupt has been handled, the computer returns to where it was by popping the data from the stack.

3. Evaluation of arithmetic expressions

One of the reasons that reverse polish or postfix notations are used is because they can be evaluated using a stack. An example of a reverse polish calculator is shown later in the text.

A simple example will demonstrate the point. Consider the list 12,23, + and assume this is in a stack with the + at the top

If we POP the top item we can determine that it is an operator. We now know that the next two POP operations must give us two numbers, which we will add. This is processed using the notion of a stack.

© IBO 2004 5.1.4 USES OF QUEUES

The primary use of a queue is to ensure that the service that is being queued for is given first to those that joined the queue first. When you queue in a customer service line you expect to be served in the order in which you have joined the queue i.e. the person in front of you will get served before you do.

Some examples of how queues are used in computer science are now briefly discussed.

1. **Keyboard queue:** the keyboard buffer into which characters are stored as they are pressed, operates as a queue. The first letter typed is the first letter sent. Thus, as subsequent letters are typed they are added at the tail of the buffer.

2. Printer queues: Printers on a LAN are a shared resource. However, the way they are managed is by queuing requests in a printer queue. A print queue stores the requests as they arrive and hence operates as a 'first request printed first' queue. The print queue on your PC also operates this way. If you ask for four things to be printed and view the print queue it will show the names of the print jobs in the order they were requested with the first one requested being on the top or head of the list.
3. Customer queue simulations: Have you ever wondered why supermarkets have the number of check operators they do at a particular time or why the traffic lights change at certain time intervals? Check outs and traffic lights are good examples of queues. The aim is to minimise the wait time in the queue, but also to minimise the number of queues needed, to keep costs down. However, the gap between arrivals to join the queue varies. The aim is to maximise service standards but minimise cost. At times the arrival rate is low per time interval and at other times the rate is high i.e. 'off peak time' and 'peak time'. To work out how often to change lights and how many counters to operate in supermarkets, computer programs are written to simulate the arrival rates and service times for members in the queue. In this way the optimal number of open counters required, and the optimal time interval for lights can be estimated.

EXERCISE 5.3

1. Explain how a stack could be used when a program that contains procedures and functions is executing.
2. Assume that there are three procedures: PROC1, PROC2, PROC3. Within PROC1 a call is made to PROC2 and within PROC2 a call is made to PROC3. Explain how a stack can be used to ensure that when PROC1 calls PROC2 the program is able to keep processing when these series of calls have been completed.
3. A system programmer has been asked to write a utility program that will allow people to see what jobs are waiting to be printed. Explain why a certain type of data structure would be more suitable than other possibilities.



QIB0 5.1.5 DISCUSS THE FEATURES AND APPROPRIATE USAGE OF BINARY TREES.

Binary trees can be used for a range of purposes. Some of these are listed below.

- Storing index keys to enable indexed access to a file.
- Storing mathematical expressions to enable evaluation using reverse-polish.
- Storing file directory structures to enable searching.
- Storing decision trees.
- Storing any set of data that needs to be searched efficiently.
- Store a physical list of data so that the data can be retrieved in sorted order.

The key feature is that the data can be accessed using one of three traversal techniques. These are outlined in section 5.3.12.

EXERCISE 5.4

1. Develop a list of further applications of binary trees.
2. Explore the use of binary trees in graph theory.
3. Select one application of a binary tree you have discovered and discuss the reasons why the binary tree is the appropriate data structure for the application.



© IBO 2004 5.2 STATIC DATA STRUCTURES

The most commonly used static data structure is that of the array, which was introduced in Chapter 1. A static structure allows lists of data of the same type to be stored and accessed via an index that specifies the position of the data in the list. Thus, an advantage of storing data in a static array list is that individual data elements can be accessed directly without reference to other items in the list provided that the index position of the required data item is known. However a disadvantage is that the length of the list cannot be extended during the running of the program. This means that static arrays have to be dimensioned to take into account the likely maximum size of the list prior to program execution.

A range of search and sort operations can be performed on static array lists of data. We have already discussed a range of sorting techniques and searching techniques in Chapter 1.

It is important that lists can be sorted efficiently. In this Chapter we will firstly consider a very efficient sorting technique called the 'Quick Sort'.

We will then consider the problem of how to access items in a list if the index position is not directly known. The method used is to convert a key, such as a name, into an index position, which can then be used to retrieve the desired data item. This technique is referred to as 'hashing'.

© IBO 2004 5.2.1 - 5.2.2 QUICK SORT

The quick sort algorithm presented here uses recursion. Readers are advised to cover section 5.5 on recursion before attempting this section.

The quick sort operates very efficiently for large lists. It applies a 'divide and conquer' problem solving strategy.

The basic idea is this:

- Pick the middle value and call this the 'pivot' .
- Start at each end of the list by using a left and right pointer.
- Move those values that are less than the pivot to the left of the pivot.
- Move those values that are greater than the pivot to the right of the pivot.
- Stop. At this point the pivot is in the correct place. And the list is now partitioned into two. Pick the left hand partition and apply the procedure again and again until sorted.
- Move the right pointer to the midpoint and retain the left pointer. Recalculate the mid point and repeat. Repeat until only one value is left in the partition.
- Pick the right hand partition and apply the procedure again and again until the list is sorted. Repeat as above.
- List is sorted.

An algorithm for the quick sort is shown below.

Points to note are:

Mid point is chosen as the PIVOT value

There are two recursive calls one for the left and one for the right of the pivot.

A trace using the algorithm is shown on the following page. The array used is:

10,4,8,1,7 these values move through this sequence

7,4,8,1,10

7,4,1,8,10

1,4,7,8,10

```

void quickS (int start, int finish, int [] array)
{
1   int pivot, left, right, temp;
2   left = start;
3   right = finish;
4   pivot = array[ (left+right)/2]
5   while (right> left)

6       while (array[ left] < pivot) {left = left+1;}
7
8       while (pivot < array[ right] ){ right = right-1;}
9
10
11      if (left<=right)
12      {
13          temp           array[ left];
14          array[ left] = array[ right];
15          array[ right] = temp;
16          left = left + 1;
17          right = right - 1;
18      }
19      if (start<right) quickS (start, right, array);
20
21      if (left<finish) quickS (left, finish, array);
}

```

TRACE OF THE QUICK SORT ALGORITHM

Note: A trace of a quick sort is difficult because of the recursion. When you trace, be aware of when recursive calls are made.

Advanced Data Structures and Algorithms

In the table below:

- (a) S=Start, F=finish, P=pivot, L=left, R=right, T=temp, D[L]=data[leftJ],
D[RJ]=data[rightJ], D[] = contents of the array
- (b) The condition flags the result of the while or if condition check as true or false

| Line | S | F | P | L | R | T | D[L] | D[R] | D[] | Condition |
|---------------------------------|---|---|---|---|---|---|--------|------|-----------------|-------------|
| call | I | 5 | | | | | | | 10,4,S, ,1,7 | |
| 2,3,4 | | | S | I | 5 | | | | | |
| 5 | | | | | | | | | | True |
| 11 | | | | I | 5 | | | | | True |
| 12 | | | | | | | 1 0 | | | |
| 13 | | | | | | | 7 | | | |
| 14 | | | | | | | | 10 | | |
| 15 | | | | 2 | | | | | | |
| 16 | | | | | 4 | | | | 7,4,S, ,1,10 | |
| IS | | | | | | | | | | Loop line 5 |
| 5 | | | | | | | | | | True |
| II | | | | 3 | 4 | | | | | True |
| 12 | | | | | | S | | | | |
| 13 | | | | | | | 1 | | | |
| 14 | | | | | | | | S | | |
| 15 | | | | 4 | | | | | | |
| 16 | | | | | 3 | | | | 7,4,1, S,IO | L<S,R>S |
| IS | | | | | | | | | | Loop line 5 |
| 5 | | | | | | | | | | False |
| 19 | 1 | 5 | | 4 | 3 | | | | | True |
| Call quicksort(1,3 ,data) | | | | | | | | | | |
| call | 1 | 3 | | | | | | | | |
| 2,3,4 | | | 4 | 1 | 3 | | | | | |

| | | | | | | | | | |
|--------------------------|---|---|---|---|---|---|---|----------------|---|
| 5 | | | | | | | | | True |
| II | | | I | 3 | | | | | True |
| 12 | | | | | 7 | | | | |
| 13 | | | | | | 1 | | | |
| 14 | | | | | | | 7 | | |
| 15 | | | 2 | | | | | | |
| 16 | | | | 2 | | | | | |
| 18 | | | | | | | | | Loop line 5 |
| 5 | | | | | | | | | False |
| 19 | 1 | 3 | | 2 | 2 | | | | True, call |
| quicksort(1,2 ,array) | | | | | | | | | |
| call | 1 | 2 | | | | | | | |
| 2,3,4 | | | I | I | 2 | | | | |
| 5 | | | | | | | | | True |
| 11 | | | I | I | | | | | True |
| 12 | | | | | 1 | | | | |
| 13 | | | | | | 1 | | | |
| 14 | | | | | | | I | | |
| 15 | | | 2 | | | | | | |
| 16 | | | | 0 | | | | 1,4,7, 8,10 | |
| 18 | | | | | | | | | Loop line 5 |
| 5 | | | | | | | | | False |
| 19 | | | | | | | | | False (note L=2 F=3 after recursion! !) |
| 21 | | | | | | | | | True Call |
| quicksort(2,3 ,data) | | | | | | | | | |
| call | 2 | 3 | | | | | | | |
| 2,3,4 | | | 4 | 2 | 3 | | | | |
| 5 | | | | | | | | | True |
| 11 | | | 2 | 2 | | | | | True |
| 12 | | | | | 4 | | | | |

| | | | | | | | | | |
|-------------------------|---|---|---|---|---|---|---|----------------|--|
| 13 | | | | | | 4 | | | |
| 14 | | | | | | | 4 | | |
| 15 | | | 3 | | | | | | |
| 16 | | | | 1 | | | | 1,4,7, 8,10 | |
| 18 | | | | | | | | | Loop line 5 |
| 5 | | | | | | | | | False |
| 19 | | | | | | | | | False (note: L=4, F=5 after recursion) |
| 21 | | | | | | | | | True Call |
| quicksort(4,5 ,data) | | | | | | | | | |
| call | 4 | 5 | | | | | | | |
| 2,3,4 | | | 8 | 4 | 5 | | | | |
| 5 | | | | | | | | | True |
| 11 | | | 4 | 4 | | | | | True |
| 12 | | | | | 8 | | | | |
| 13 | | | | | | 8 | | | |
| 14 | | | | | | | 8 | | |
| 15 | | | 5 | | | | | | |
| 16 | | | | 3 | | | | | |
| 18 | | | | | | | | | Loop line 5 |
| 5 | | | | | | | | | False |
| 19 | | | | | | | | | False |
| 21 | | | | | | | | | False |

We are finished. This is a difficult trace and you need to be very careful when checking the path of the recursive calls. You might like to code the algorithm and put output statements in appropriate places to output key values and check the above trace.

EXERCISE 5.5

1. Use the quick algorithm shown above to perform a trace to show how the following list would be sorted: 6,23,4,56,2.
2. Find another version of the quick sort and perform a trace to show how it operates.



©IBO 2004 **5.2.3 CONSTRUCTING HASH TABLES**

As mentioned in the introduction to this section an advantage of a static array is that the data elements can be accessed directly via the index. However, if you do not know the index, you need to search the array. Using a linear search is slow and to use a binary search the list needs to be constantly sorted, which takes up computer resources. One way to overcome this problem is 'hashing'.

Generating hash codes

Data stored in a 'hash table' is accessed by applying a mathematical function to a data value stored in the table so as to get its address or index in the table. The table is an array data structure, thus the value returned by the hash function is the array index position.

Mathematically we say 'index =hash(key)'.

Let's say we wish to store a set of eight 'three letter product codes' in an array. The data in the array would act as a look up table. To gain access to the codes we could use a linear search, which would take on average $\frac{8}{2} = 4$ array accesses to locate the desired name and the worst case would

be 8 accesses. An improved way would be to somehow calculate the array index from the code. Whilst there is some processing overhead in calculating the index it is usually faster than a sequential search, especially as the list increases in length.

One way to generate the index value would be to convert the code to the ASCII sum of its letters. By then performing modulo division using a divisor of 8 we would get remainders that range from 0 to 7. These remainders would act as indexes for an array with 8 elements.

The following algorithm produces the hash table. The table is shown following the algorithm.

```
public class hash
{
    public static void main(String args[] )
    {
        String [] c ={ "asd", "wer", "ert", "rty", "tyu", "erf", "ghj", "yui" }
        new public hash(c);

        public hash(String codes[] )
        {
            int ht=0;
            char ch;
            String st;
            output("Code\tHash Total\tHash Value(key)");
            for (int i=0; i<codes.length; i++)
            {
                st = codes[ i] ;
                for (int j=0; j<3; j++)

                    ch = st.charAt(j);
                    ht = ht + ch;
                }
                output (codes[ i] +"\t"+ht+"\t"+ ht%8);
            }
        }
```

```
ht = 0;
```

The output from this algorithm is as follows:

| Code | Hash Total | Hash Value(key) |
|------|------------|-----------------|
| asd | 312 | 0 |
| wer | 334 | 6 |
| ert | 331 | 3 |
| rty | 351 | 7 |
| tyu | 354 | 2 |
| erf | 317 | 5 |
| ghi | 313 | 1 |
| yui | 342 | 7 |

The table shows the hash totals and the hash keys. You can check these by either using the program or using the ASCII code table. We will work out one code as an example.

The code 'asd' has the hash total = $97 + 115 + 100 = 312$.

This then converts to the hash key of $312 \bmod 8 = 39$ remainder $0 = 0$.

What we want is for the hash function to spread the index values out evenly over a range of values so that there are no clashes i.e. two codes having the same hashed value. In this example the hash table requires at least 8 locations. However, we normally allocate more spots. Why? The reason is that the hashed values need not be unique. It is entirely possible that the hash values of two different product codes could be the same. This occurs when the remainders are equal. This creates a clash or collision of hash keys because we cannot store two values in one index location in the array. As an aside, you can store two numbers in the one location. For example, assume that values from 0 to 99 need to be stored. They could be stored as 99 99 i.e. 9,999. The first number would occupy the first two digits and the second the last two!

In the above example the product codes 'rty' and 'yui' have the same hash keys.

There are three ways to handle such clashes or collisions.

1. **Overflow area:** we divide the table up into two sections. The first or main section holds the data items as normal. The second section is called the 'overflow area' and data values that result in clashes are stored here in serial order. When a data value is required and a clash occurs, the overflow area is searched sequentially until the desired data value is found.
2. **Chaining:** here the array values point to a possible chain of data values that all hash to the same index. As a new data value is determined that clashes with an existing value, it is chained on the end of the list pointed to by the index. This is a dynamic data structure which is dealt with later in this chapter. A two dimensional array could also be used to hold the possible codes that hash to the same row number, but would waste a considerable amount of space.
3. **Probing:** as the array contains more spots than is required there will be free spots in

amongst the used sections. When a clash occurs we simply look through the array until we find a free spot and insert the data item there. In future, when a clash occurs, we start a sequential search from the index where the clash occurred until we locate the required data item. We could also, by default, start at the top of the list.

A comment about performance - hash tables require more memory allocated to them than there are data values to fill them. Thus more memory is required than a sequentially searched array. Well hashed tables evenly allocate the data values over the remainder space. It has been found that prime divisors do this best. A well hashed table should have a minimum number of data values that clash. If a hash table has a number of clashes it needs to be re-hashed using a different hashing algorithm or larger divisor. Often it is necessary to simulate the requirements before a hashing method is decided upon.

In the above example if we chose to divide by 51 we would avoid clashes and generate these hash key values: 6,28,25,45,48,11,7,37. Thus, to avoid clashes, we would need to allocate an array of dimension 51.

The algorithm below shows how the hash table could be constructed. A use of the hashed table of product codes is to allow validation of an input code by looking in the hashed array to check that it matches a valid code.

A sample algorithm to demonstrate some of these points is shown below.

```

import java.io.*;
public class hashIT{
    public static void main (String args[])
    {
        String [] c = {"asd","wer","ert","rty","tyu","erf","ghj","yui" }
        new hashit(c);
    }
    hashIT(String codes[]
    {
        String hash [] = new String[ 51] ;
        int ht=0;
        for (int i=0; i< hash.length ; i++)
            hash[ i] ="";
        for (int i=0;i<codes.length;i++)
            for (int j=0;j<codes[i] .length();j++)
            {
                ht=ht+codes[ i] .charAt (j) ;
                hash[ ht%51] =codes[ i] ;
            }
            output (codes[ i] +" "+ht+" "+ht%51);
            ht=0;
        }
        ht=0;
        String pcode="asd";
        for (int j=0;j<pcode.length();j++)
    }
}

```

```

        ht=ht+PCODE.CHARAT(j);
    }
    OUTPUT(ht%51+" "+HASH[ ht%51] );
    IF (PCODE==HASH[ ht%51]
    {
        OUTPUT("Valid Code");
    }
    ELSE
    {
        OUTPUT("Invalid Code");
    }
}

```

This algorithm produces the following output.

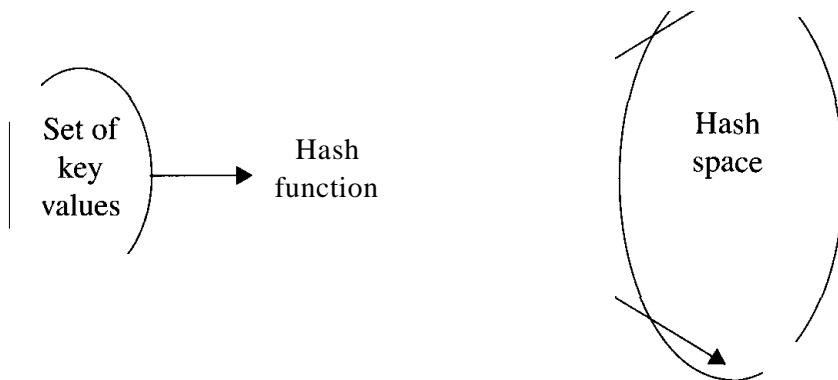
```

asd 312 6
wer 334 28
ert33125
rty 351 45
tyu 354 48
erf 317 11
ghj 313 7
yui34337
6 asd
Valid Code

```

The hashing process can be thought of as mapping the set of keys onto a set of hash keys or hash space. The diagram below shows this concept using the above example. It shows the set of product codes being mapped onto a hash key space where each product code maps to only one hash key.

Figure: 5.2 - Hash Space



EXERCISE 5.6

1. You are required to store the surnames of the people in your class in an array. To enable an efficient lookup of a persons surname to see if they are in your class you are required to setup a hash table using the surnames (you might need to add initials).

- Step I: Calculate the hashtotal for each person in your class.
- Step 2: Experiment with different divisors to minimise the possibility of clashes.
- Step 3: Write a program to store the names in a hashed array (you might find it easier to read the names from a sequential file).
- Step 4: Write a procedure or function which accepts a surname and then returns a suitable value that can be used to decide if the person is in the class or not!



©IBO 5.2.4-5.2.5 STACKS 2004

As mentioned above, stacks are 'Last In First Out' data structures that can easily be implemented using a static one dimensional array.

The basic operations are PUSH(data item) onto the stack and POP(data item) a data item off the stack. A stack pointer is used to point to the next available free space and has an initial value of 0.

The stack has a maximum size set to the length of the array.

A stack can be in three states:

Empty. Thus we cannot POP and the stack pointer is at 0.

Part full: can PUSH and POP

Full: Thus we cannot PUSH and the stack pointer is equal to its maximum value.

In the following functions the array is termed 'stack' and the stack pointer is 'sp'.

PUSH algorithm

This algorithm is reasonably straight-forward. As the stack pointer indicates the free spot, we add the data item at that indexed position and then increment the stack pointer. Before we do the push, we must check that the stack is not full i.e. check to see if the stack pointer is equal to the maximum length of the stack.

```
void push(int data)
{
    stack[ sp++ ] =data;
```

NOTE: Most languages support the unary operator ++ and this can be used to simplify the code for the push to simply stack[sp++]. Used in this way the current value of sp is used and then it is incremented. ++sp works the other way, i.e. sp is incremented and then used, which would not be useful here.

POP algorithm

The key idea is that the stack pointer points to the next available free spot. If it is 0, the stack is empty. If it is equal to the Maximum, the stack is full and we can pop, but before we pop we decrement the stack pointer. This allows access to the item to be returned and we can then set the item to the null value (e.g. 0) to indicate 'not a valid stack item'.

The basic pop algorithm is shown below. It returns the value of the location.

```
int pop()
```

```
return stack[ --sp]
```

The pop operation could be achieved by using a unary operator - to simplify the code, i.e. `returnValue = stack[--sp]`. This decrements the sp first and then is used as the index into the array.

A trace of this algorithm using a stack of 3 available spaces (i.e an array with 4 spots), is shown in diagram 5.4.

Note: In most languages such as C, C++ and Java the array index values begin at 0. Thus the length of the array in physical terms is always one more than the maximum allowable index value. When the last item was placed on the stack, the stack point (sp) was still incremented i.e. it is now equal to the length of the array. We can use this fact to pick up if the stack pointer points past the end of the stack, causing an overflow error. This is done by asking if the stack pointer is equal to the maximum length of the array i.e. one more than the actual number of index values. If it is, we generate an error and do nothing. The stack pointer only gets to be equal to the array length when the stack becomes full. As we trap this value before push, our implementation works to prevent overflow.

In most implementations, functions are included which can be used to return the empty or full state of the stack. Such functions are declared to be boolean and thus return true or false states of the stack related to whether the stack is full or empty.

Lastly, let us discuss the initial state of the stack. The stack array should be initialised to a set value, though this is not absolutely necessary. The important initialisation is to set the stack pointer to 0.

An algorithmic implementation of a standard stack is shown below. It uses the notion of two boolean functions to return the full or empty status of the stack. These are tested prior to attempting either a pop or push operation. The implementation uses imperative or standard programming structures and the stack point (sp) and the array `stack[]` can be thought of as being declared as global variables.

```
public class Stack
{
    int [] stack;
    int max;
    int sp;
    public static void main(String args[])
    {
        new Stack(3);
    }
    public Stack(int size)
    {
        init(size);
        int data = 0;
        push(10); push(5); push(3); //stack is full!
        if (stackFull())
            System.out.println("Stack Full");
        showStack();
        if (!stackFull())
    }
```

```

        push(21); //try to add to full stack
    else
        System.out.println("Can't PUSH StackFull");
    data = pop(); data = pop(); data = pop(); //stack is empty
    if (stackEmpty())
        System.out.println("Stack is empty");
    showStack();
}
void showStack()
{
    for (int i=0; i<sp; i++)
        System.out.print(stack[ i ] + " .. ");
    System.out.println("");
}
void init(int size)
{
    max = size;
    stack = new int[ size ];
    for (int i=0; i<size; i++)

        stack[ i ] = 0;
    }
    sp = 0;
}
void push(int data)
{
    stack[ sp++ ] = data;
}
int pop ()
{
    return stack[ --sp ]
}
boolean stackFull()
{
    if (sp==max)
        return true;
    else
        return false;
}
boolean stackEmpty()
{
    if (sp==0)
        return true;
    else
        return false;
}
}

```

The output of this algorithm is as shown below.

Stack Full

10 .. 5 .. 3 ..

Can't PUSH Stack Full

Stack is empty

Advanced Data Structures and Algorithms

The program uses two methods to check the status of the stack: stackFull and stackEmpty. The stack pointer is set to point to the first element in the array i.e. $sp = 0$. If on a POP operation $sp = 0$, we can conclude the stack is empty. This is because on a PUSH operation the element to be added to the stack is inserted at the position pointed to by the stack pointer i.e. in the first case the element is inserted into the first position. The stack pointer is then incremented to point to the next free spot i.e. $sp \leftarrow sp + 1$ and $stack[sp] \leftarrow x$.

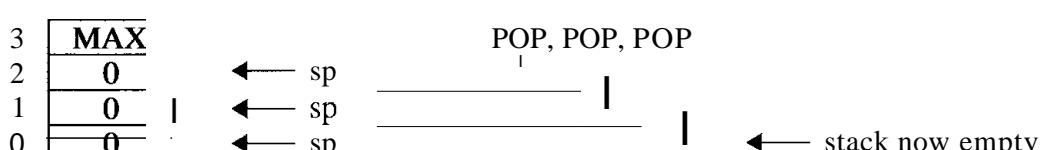
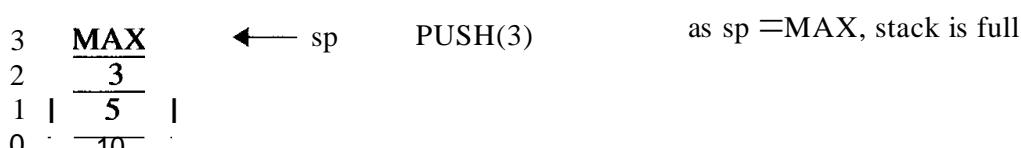
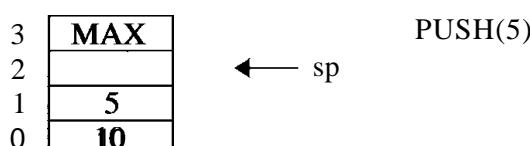
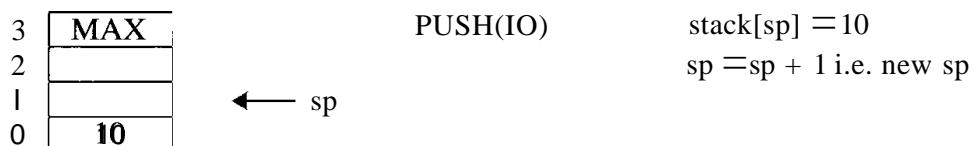
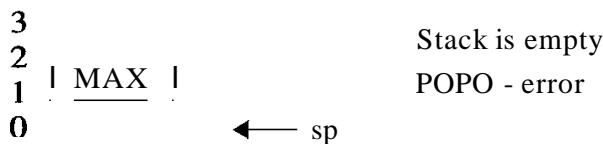
The pop operation uses the reverse of this idea. When an element is removed from the stack the stack pointer is decremented and then used as the array index i.e. $stval = stack[sp]$ and then we decrement the stack pointer i.e. $sp \leftarrow sp - 1$.

In this way the stack pointer is always pointing to the next free spot. Thus if it equals one more than the maximum length of the stack, the stack must be full. In this case we can POP but we cannot PUSH. If we POP the stack pointer is decremented first and points to the last element in the stack. We can now PUSH because we use the current position to insert and then we add one to the stack pointer. If we test at this stage, the pointer value indicates the stack is full.

A trace of the algorithm is shown in figure 5.3. It shows the initial state of the stack and then proceeds to show the state of the stack as the values 10, 5 and 3 are PUSHED onto the stack to make it full. Three successive POP operations are then performed resulting in the stack being empty.

Figure: 5.3 - Trace using a 3 item stack

Initial condition



Stacks are commonly used to implement reverse polish calculations. The following example is taken from the IB support material and shows how a stack can be used to implement a reverse polish calculator.

EXAMPLE

Write a simulation of a reverse polish calculator. The data is to be input either directly or via a file and is assumed to be either a valid operator or a real number.

SOLUTION

If $1\ 4\ +\ =$ was entered the output would be 5. Both 1 and 4 are placed onto the stack and the $+$ calls PUSH($\text{pop}()$ $+$ $\text{pop}()$). The effect of this call is to assume that two numbers are on the stack and the first is removed and the second removed and added. The result is then PUSHed back onto the stack.

The $=$ operator causes the error flag to be set to 0 and the value of answer to be set to the value on the top of the stack i.e. $\text{answer} = \text{pop}(\text{stack})$.

The algorithm implementation is shown below.

```
import java.io.*;
class Rpn

    int sp=0;
    int error = -1;
    public static void main(String args [] )
    {
        new Rpn () ;
    }
    public Rpn ()
    {
        double stack [] = new double [10]
        char term='.';
        double answer=0, number=0;
        String messages [] = new String [4]
        messages[ 1]      "Division by zero";
        messages[ 2]      "Stack underflow";
        messages[ 3]      "Memory overflow";
        String inLine;
        while (error<0)
        {
            inLine = input("Enter expression");
            term = inLine.charAt(0);
            if (term == '=')

                error = 0;
                answer = pop(stack);

            else if(term      '+')push(stack, pop(stack)+pop(stack));
            else if(term      '-')push(stack,-1*pop(stack)+pop(stack));
            else if(term      '* ') push(stack, pop(stack)*pop(stack));
            else if(term      '/')
```

```

        number = pop(stack);
        if (number==0)
            error = 1;
        else
            push(stack, pop(stack)/number);

        else push(stack, Double.valueOf(inLine) .doubleValue());
        for (int i=0;i<stack.length;i++)
            output (stack[ i]);
    }
    if (error ==0)
        output("The answer = "+answer);
    else
        output ("Error: "+messages[ error] );
}

void push (double stack[ ] , double value)

if (sp==stack.length-1)
    error = 3;
else
    stack[ sp++]=value;
}
double pop(double stack[])
{
    double n=0;
    if (sp==0)
        error = 2;
    else
        n=stack[ --sp]
    return n;
}

```

EXERCISE 5.7

1. Define the term 'stack' and the operations POP and PUSH.
2. What errors can occur when using a stack?
3. What is the purpose of the stack pointer?
4. Draw a sequence of diagrams to show the contents of a stack array called PILE [] if the following operations are performed. The maximum size of PILE is 3. Show the stack pointer and assume that errors are detected for appropriate conditions. Use this sequence of operations:

PUSH 10, PUSH 23, POP, PUSH 34, PUSH 21, POP, POP

5. Implement the simple stack program to manipulate characters. Write out a suitable set of test data with expected results and then test your program. One such program might reverse a string. For example, input APPLE and display it as ELPPA.

6. Re-write the reverse polish calculator to use a class for the stack and another for the calculator. Test that this works. What advantage does this method bring?



© IBO 2004 **5.2.6-5.2.7 QUEUES**

Queues are First In First Out data structures that can be implemented using a one dimensional array.

Data items are removed from the beginning of the queue. Thus if we used a simple array we would remove the item the first time from the first position. It would then make sense to move the other items up one spot. By doing this we would simplify things as all we would need to do is to keep track of the tail of the queue. BUT, imagine the extra processing that is required in constantly moving the data items. We therefore do not use this method and usually represent a queue as a circular data structure with a separate index pointer for the head and tail of the queue.

We also use two boolean flags, 'qfull' and 'qempty', to keep track of the state of the queue. As with stacks, an underflow errors occurs if you try to remove an item from an empty queue, and, an overflow error occurs if you try to add data past the maximum size of the queue.

The example below shows a queue with four spots. It is shown as a circular representation of the array. Note that the initial state of the queue is to have:

- qfull set to false
- qempty set to true
- head set to 0
- tail set to 0

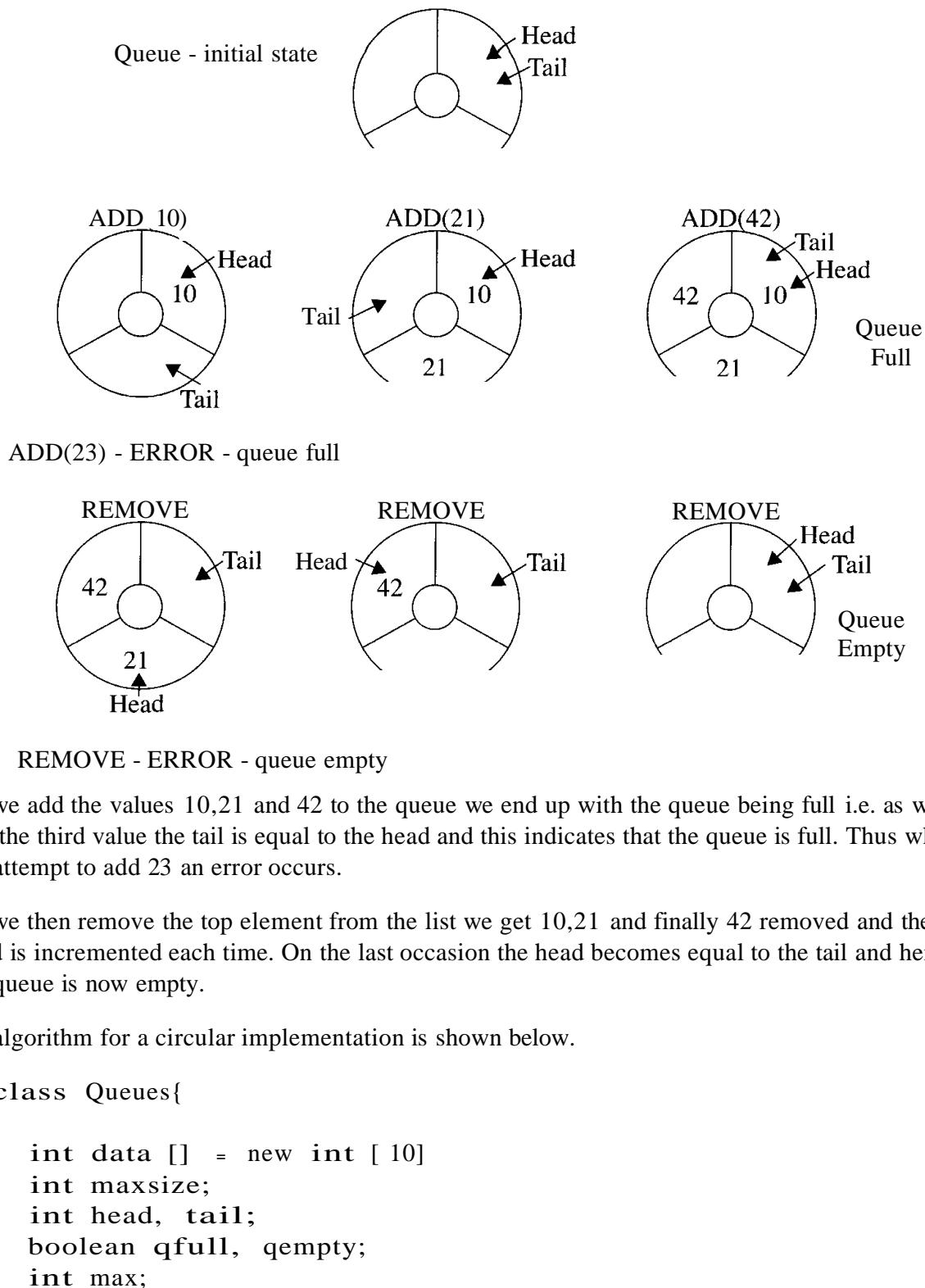
This prevents the algorithm from trying to remove from the queue when it is empty. If we add to the queue we increment the tail pointer by one. As we add to the queue the tail pointer can become equal to the maximum size of the queue. If so, the tail is set back to 0 and a test is made to see if it has now become equal to the head. If it is, the tail value remains the same but the queue full flag is set to true i.e. qfull =true.

As we remove items from the queue the head of the queue is incremented. As we increment we check that the value has not gone past the maximum. If it has, we set it to 0. Irrespective we must check to see if the new head value is equal to the tail value. If it is we have just made the queue empty and must set the qempty flag to true i.e. qempty =true.

An example and diagram will help explain how this operates. Assume the queue has three spots and to add an item to the queue we use ADD(item) and to remove an item we use REMOVE. In the diagram the following operations are traced and the contents of the queue and the values of head and tail are shown:

ADD(10), ADD(21), ADD(42), ADD(23) note: error, REMOVE, REMOVE, REMOVE
REMOVE note error.

Figure: 5.4 - State of Queue



REMOVE - ERROR - queue empty

As we add the values 10, 21 and 42 to the queue we end up with the queue being full i.e. as we add the third value the tail is equal to the head and this indicates that the queue is full. Thus when we attempt to add 23 an error occurs.

As we then remove the top element from the list we get 10, 21 and finally 42 removed and the head is incremented each time. On the last occasion the head becomes equal to the tail and hence the queue is now empty.

An algorithm for a circular implementation is shown below.

```

class Queues{

    int data [] = new int [ 10]
    int maxsize;
    int head, tail;
    boolean qfull, qempty;
    int max;

    static void main (String [] args)

        new Queues();

    public Queues()
    {
        initQueue();
    }
}

```

```

int response, item, queueValue;
output("1: add 2: remove 3: print 4: quit if");
response = inputInt("Enter Choice");
while (response !=4)

    if (response == 1)

        item = inputInt("Input value to add onto queue ");
        add(item);
    }
    if (response == 2)
    {
        queueValue=remove();
    }
    if (response == 3)
    {
        printQueue();
    }
    output("<1>: add <2>: remove <3>: print <4>: quit ");
    response = inputInt("Enter Choice");

void initQueue()

for (int i=0; i<data.length; i++)
{
    data [ i] = 0;
}
maxsize=data.length;
head=0;
tail=0;
qfull=false;
qempty=true;
}

void printQueue()
{
    if (!qempty)
    {
        for (int i=0; i<data.length; i++)
        {
            output (data[ i] );
        }
    }
    else
        output ("Error: queue empty");

void add(int x)

if (qfull)

```

```
    output ("Error: queue FULL");

else

    data[ tail] =x;
    qempty=false;
    if ((tail+1)==maxsize-1)
    {
        tail=0;
        if (tail==head)
        {
            qfull=true;

else

    tail=tail+1;
    if (tail==head)
    {
        qfull=true;

int remove ()

    int qval;
    if (qempty)
    {
        output ("Error: queue empty");
        qval=-1;

else

    qval = data[ head];
    qfull=false;
    data[ head] =0;
    if ((head+1)==maxsize)
    {
        head=0;
        if (head==tail)
        {
            qempty=true;

    }
else

    head=head+1;
    if (head==tail)
```

```

qempty=true;

return qval;

```

We can simplify these algorithms by a neat use of the modulo (integer remainder) function.

EXERCISE 5.8

1. Define the nature of a circular queue data structure.
2. What are the initial conditions of a queue?
3. What errors need to be detected when operating a queue?
4. What is the purpose of the head and tail variables?
5. Implement the simple queue program using a separate class for the queue. Develop a set of test data and expected results and test your implementation.
6. A clerk is positioned at a service counter where customers queue to return faulty purchases. As customers come into the complaints department they enter their surnames on a computer terminal. You are required to write a program to store the names of the people who queue at a service counter using a queue. The program should be able to retrieve the top name from the queue and display this to the clerks so that they can address the customers correctly and allow queuing customers to be added to the end of the queue. It should also be possible to display the members in the queue and the length of the queue.
7. Use the internet or other source to look up the implementations that use the mod function and implement the algorithm in full.



© IBO 2004 5.3 DYNAMIC DATA STRUCTURES

Arrays store data in a group of memory locations that cannot be extended once the program has been compiled and executed. Hence arrays are referred to as static data structures. To overcome the problem of needing to know in advance the likely maximum length of a list, we can use a chaining mechanism by linking memory locations together to represent a list that is made up of linked memory locations. Each memory location in the list points to the memory address of the following member of the list. Thus the memory locations are said to be linked together. Such data structures are referred to as 'linked lists' and as they are not bounded like arrays, they are said to be 'dynamic data structures'.

© IBO 2004 5.3.1 DEFINITION OF AN OBJECT REFERENCE

In Java, links are implemented by using reference variables that contain the memory address of the object being referenced. Thus a class can be defined that has a reference variable that can be used to reference (point to) other objects of the same class type.

A class from which list objects can be created is shown below. Objects created from the class have a data segment and a reference variable that allows other objects of the same type to be referenced. The data segments can be a collection of variables of different, or the same, data type

Advanced Data Structures and Algorithms

i.e. to form a record. For example we could declare a class Node, which holds an integer age and string name variables along with a reference variable called next that is itself of type class Node, as shown below.

```
public class Node
{
    int age;
    String name;
    Node next;
```

Note: this a very simple class definition. The variables are directly accessible using standard objectName.variable dot notation; no accessor methods or edit methods are required to access the data. The data parts are public and not private to the class. This all simplifies implementation and aids understanding of the fundamentals of the process.

Java allows reference variables to be created to 'point' to existing objects.

Using the above Node class we can create a head and tail reference.

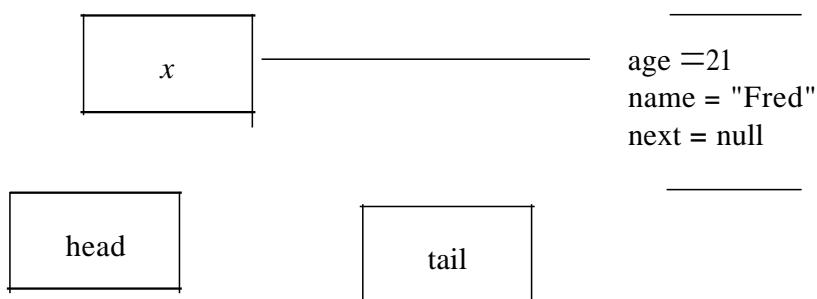
```
Node head null;
Node tail null;
```

Currently both head and tail do not reference any object, they are set to null.

If we now create an object of type Node called x and assign some data to age and name we can now assign head and tail to reference the new object x.

```
Node x new Node(); //create and instance of the object
x.age 21;
x.name "Fred";
next null;
head x; //head references the object x
tail x //tail references the object x
```

Diagrammatically the situation is as follows:



An object reference variable holds the memory address of another object, which can be of the same class.

Using the Node class above we could have a list of nodes linked together to represent a set of peoples' names and ages.

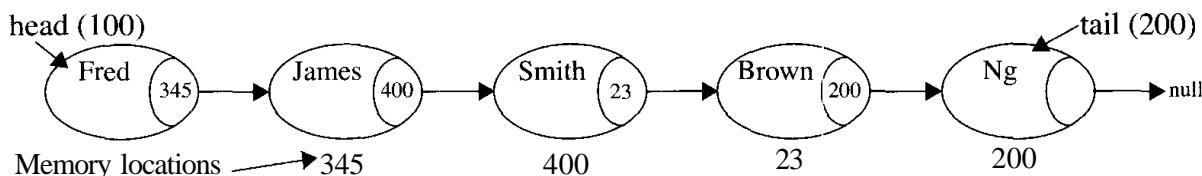
Each node in the list is comprised of a name and age as the data part and a pointer to the next element in the list. This list is a simple example of a linked list. Let us assume that we have five people and the data is represented as a list as shown below.

[(Fred,21, 345), (James, 23,400), (Smith, 12,23), (Brown, 34, 200), (Ng, 23, null)].

The first element of each record is the name, the second the age and the third the physical memory address. Assume that Fred occupies location 100. Head points to this value.

We can draw this set of nodes as a linked list as shown below. The first node has the name Fred and its next pointer has the value 345 to indicate that the next element in the linked list is James.

Figure: 5.5 - Linked List



Showing the memory location

Notes: allows the pointer to indicate the address.

Separate Head and Tail pointers

allow the list to operate as a stack or a queue.

References allow lists of data to be structured in various ways. An example of a simple linked list was shown above. The advantage is that the length of the list need not be known and hence the data structures constructed using pointers are known as 'dynamic data structures'. We will also see in the next section that there are several other advantages. Data can easily be inserted without the need to constantly move data items around in the list. Data can also be kept in sorted order by inserting the data into the list at the appropriate point. The overhead is that the list needs to be searched each time to locate the desired insertion point. Elements in a dynamic list cannot be accessed directly via an index as they can in an array.

©IBO 5.3.2 ALGORITHMS THAT USE REFERENCE MECHANISMS 2004

Java provides a very convenient referencing mechanism and you have been using it whenever you pass objects to a class or method or return an object from a class or a method.

References in Java are passed to and returned from objects and methods in the normal way i.e. as actual parameters and assigned to formal parameters. The mechanism is a pass-by-value one. What is passed is the value of the memory address of the object.

A number of algorithms that use the reference mechanism are discussed in the following sections.

©IBO 5.3.3-6 LINKED LISTS: SINGULAR, DOUBLE AND CIRCULAR 2004

A linked list is a list of nodes. Each node must contain a reference variable to the next node in the list and the last node in the list has its reference variable set to NULL to indicate the end of the list. By default, a head reference also needs to be created to indicate the start of the list and, optionally, a tail can be used to indicate the end if this is thought necessary.

Linked lists can be of three types. The following discussion relates to setting up a singly-linked linked list. The other structures are briefly considered after this section.

Singularly linked lists are linked lists where each node has a reference to the next node. A separate reference indicates the start of the list and the last node is set to NULL to indicate the end of the list. The list can only be traversed in one direction from the beginning to the end. It is

Advanced Data Structures and Algorithms

normal to use a head reference and to add items to the head of the list i.e. the list acts like a stack. You could also allocate a tail pointer to allow items to be added at the tail. Otherwise the list would have been searched to find the end of the list before a new node could be added to the list.

To create a linked list you need to follow these steps; the diagrams shows what is happening at each stage. We will use the Node class as used above.

Create your Node class declaration i.e. what is to be contained in a Node object created from the Node class.

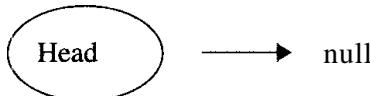
```
public Node
{
    int age;
    String name;
    Node next;
```

Create an instance of the class Node and call it head and set it to reference null.

```
Node head = null;
```

What we now have is an object called head of type Node that initialised to reference nothing i.e. null. This starts this list. Head itself does not contain meaningful data.

Figure: 5.6 a



Add an item to the list

Create another variable of type NODE, call this *n* and set head to point to *n*.

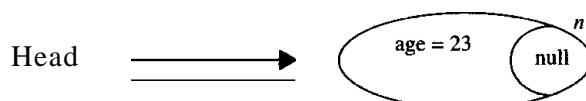
Now you can add data to NODE *n* i.e. *n->age* = 23, *n->name* = "fred" and set *n->NULL* to indicate the end of the list.

```
Node n = new Node();
n.age = 23;
n.name = "fred";
n.next = null;
```

We need to set head to reference: *n* i.e. head = *n*;

At this point the list consists two objects referencing the object *n* as the head of the list i.e. the objects *n* and head.

Figure: 5.6 b



To add another new item to the list follow these steps.

Create a new Node object called *m*.

Set head to point to *m* and *m* to point to *n*.

Assign data to *m* and set its next pointer to point to NULL.

```
Node m = new Node();
m.age = 34;
m.name = "Jane";
m.next = head;
head = m;
```

Notice the logical order of the last two statements. The new object is inserted at the front of the list between the head and the object *n* that is referenced by head.next. To start the insertion process *m*.next is set to what head is referencing i.e. head.next. Thus *m* now references *n*, but head.next is still referencing *n* also. To complete the insertion we make head.next now reference the new object *m* i.e. head.next =*m*.

The current state of the list is shown below. It now has two members of the list. The first has a reference that is set to reference the second node. The head references the address of the first node. In this case the new item is added at the front of the list

Figure: 5.6 c



Operations on Linked Lists

As with static lists we can perform a range of standard operations on a dynamic list structure.

Standard operations include the following:

- Add an item to the list at the head or tail.
- Insert an item into the list at the *nth* location.
- Search the list to find occurrence of item *x*.
- Count the number of items in the list.
- Delete an item from the list.
- Output the contents of a list to the screen, printer or disk file.

Before looking at a specific program to manipulate linked lists it is appropriate to look at some of the above operations in a less technical way.

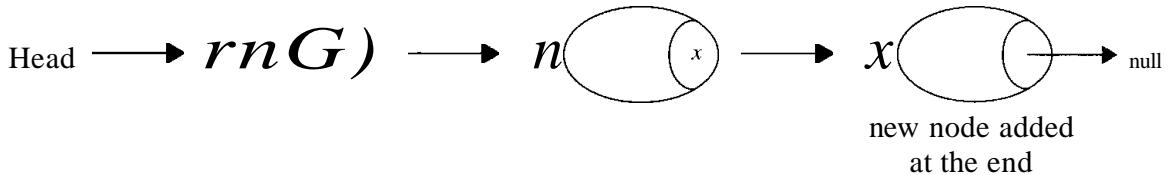
Consider the operation of adding an item to the end of the list as shown above. We would need to follow these basic steps.

```
Node x = new Node();
x.name = "Smith";
x.age = 12;
(Locate end of list, call it elist)
elist = x;
x.next = null;
```

Advanced Data Structures and Algorithms

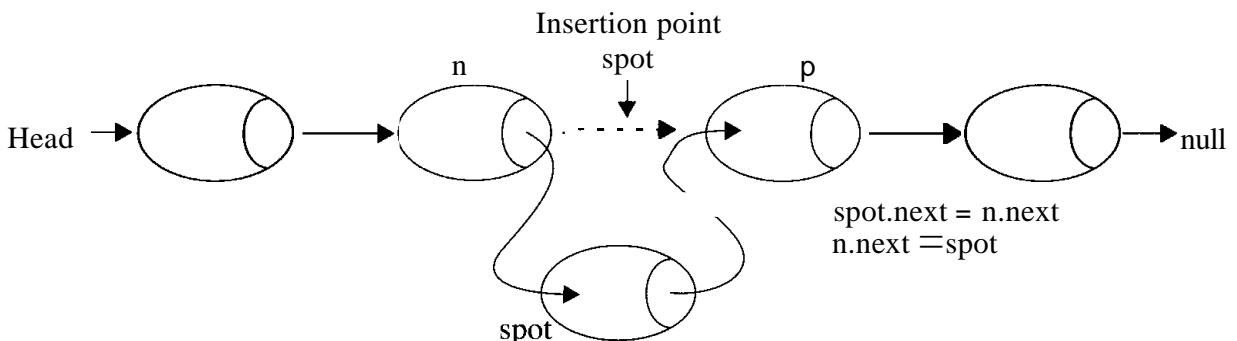
The new list looks as shown below. We are assuming that a locate function exists and this gives us the memory location of the last node as `elist`. Using this we can now set the last node to point to the newly created node i.e. `elist = x`. Thus the new element is appended to the end of the list. All that remains is to set the last node to point to null or `x.next = null`.

Figure: 5.6 d



We now turn to looking at **inserting a node** into the middle of the list. Again, we assume that there exists a function to locate the spot after which we wish to insert. This is called 'spot'! That is, we are inserting the new node after the node `spot`. The diagram below shows the sequence of events.

Figure: 5.6 e



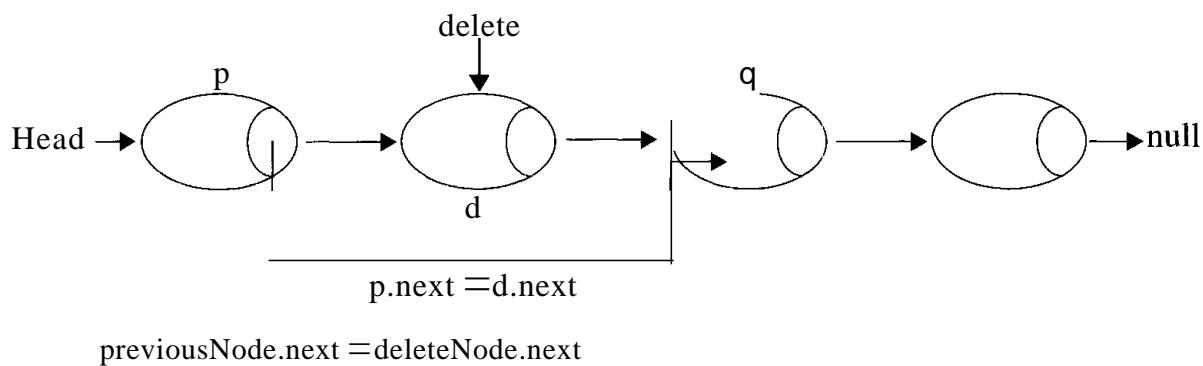
The basic sequence is listed below

- Make new node called *n*.
- Locate insertion point called *spot*.
- Set *n.next = spot.next*.
- Set *spot.next = n*.

In other words, what we need do is change the current location to point to the new location, and change the new location to point to what the current location pointed to!

Let us finally consider deletion of a node within the list. Let's assume the location to be deleted is called 'deleteNode' and the previous node is called 'previousNode' i.e. `previousNode.next` points to the node we wish to delete.

The basic sequence is shown in the following diagram. We need to make the `previousNode` point to the node pointed to by the node to be deleted. This is accomplished by a single statement as shown below in the diagram.

Figure: 5.6 f

In the next section, algorithms for dynamic stacks and queues are presented. These show the basic operations of a stack. The algorithms show how to add and delete at the head for the stack and at the tail for the queue. For many students it may be appropriate to look at these algorithms before proceeding with the algorithm described here.

EXAMPLE: OPERATIONS ON A LIST TO INSERT IN ORDER

An algorithm to demonstrate how to add, delete, search and output the contents of a list is shown following. The node used is declared within the program and the variables can be accessed directly as shown.

The key point of the algorithm is that the add operations is done to ensure the list remains in numeric sorted order.

Some brief descriptions of the methods are now provided to help you understand the algorithm.

The add function (Java method) accepts the data item to be added as an integer and a newNode created. The data is assigned. The data is then inserted into the list in numeric order. A check is made to see if the list is currently empty. If it is, the data is added at the head and the tail set to point to the head, otherwise the data is checked to see if it should be inserted before the current head or if it comes after the current tail. Otherwise the insertion point is located within the list and the data inserted at that point.

The delete function accepts the data item to be deleted. We assume that the items are unique i.e. the data could be a unique index to a file. Note that you cannot delete from an empty list. Also, if we are deleting the last item in the list we set both head and tail to null.

To indicate we found what we want, we use a boolean variable found, which is set to false to start with. To store the previous node we use a variable tempNode. The variable currentNode is used to start the search. To locate the node to be deleted we need to search through the list. Thus the currentNode is set to point to the head.

To search the list we can use a while loop, as shown below. The while loop executes whilst we have not found a match AND the next pointer is not equal to null. The latter condition would indicate that the end of the list has been found without finding a match.

```

while (! found) && (currentNode.next) != null)
{
    if (current.data == inData)

        output("found" + currentNode.data);
    found = true;
}

```

```

else
    tempNode = currentNode;
    currentNode = currentNode.next;
}
}

```

In the above algorithm, if we find what we want, we set found to true and this causes the while loop to fail. Otherwise we keep the location of the currentNode and store this in the variable tempNode and we update currentNode to reference to what the present value of currentNode.next points to. This allows the program to traverse the linked list until the value of currentNode.next is null.

When the while loop is exited we need to determine what action to take. If we found what we want and there is only one element in the list, we set both head and tail to null.

If we found what we want and it is at the end of the list, we set the tail to point to the previous element in the list. This is stored as tempNode. We need to set tempNode.next to point to null.

If what we found is in the middle of the list, we make the previous element point to what the element we are about to delete points to.

The interface is very simple and allows integer data input until -999 is entered as the sentinel value. There is no data validation to check for duplicates.

The complete program is shown below.

```

class Node

    int data;
    Node next;

public class Llist
{
    Node head    null;
    Node tail    null;

    public static void main(String args[])
    {
        new Llist();

        Llist()
        {
            int n;
            head = null; tail=null;
            n=inputInt("input nil");
            while (n!= -999)
            {
                add(n);
                display();
                n=inputInt("Enter ");
            }
        }
    }
}

```

```

item = inputInt("Input data to delete");
delete (item) ;
display();
}
void add (int d)
{
    Node newNode = new Node();
    Node current = new Node();
    current = head;
    if (head==null)
    {
        head=newNode;
        tail=newNode;
        newNode.data = d;
        newNode.next=null;

else if (d < head.data)
{
    newNode.data=d;
    newNode.next=head;
    head=newNode;
}
else if (d > tail.data)
{
    newNode.data=d;
    newNode.next=null;
    tail.next=newNode;
    tail = newNode;

else
{
    while (current!=tail)

        if (d>current.data && d<current.next.data)

            newNode.data    d;
            newNode.next   current.next;
            current.next   newNode;
            current = tail; //forces finishup

        else current = current.next;

    }
}

void delete (int d)

boolean found = false;
Node current = new Node();
Node tempNode = new Node (); tempNode=null;

```

```

current = head;

while (!found && current.next !=null)

    if (current.data==d) found=true;
    else

        ternpNode=current;
        current=current.next;

if (found &&head==tail)

    head=null;
    tail=null;
}
else
    if (found && current==tail)

        tail=ternpNode;
        ternpNode.next=null;

    else
        if (found) ternpNode.next=current.next;
    if (!found) utput("data in list");
}
void display ()
{
    Node current = new Node();
    current = head;
    while (current != null)
    {
        output (current.data+" .. ");
        current = current.next;
    }
    output ("End List\n");
}

```

EXERCISE 5.9

1. Alter this example to only add at the head of the list i.e. get rid of the tail pointer.
2. Change the program to pass the head pointer as an argument and not act as a global variable.
3. Modify the program to add the new piece of data so that the list is sorted in numeric order.
4. Dossier preparation program. Random access files on a disk cannot be displayed in some kind of sorted order because the order is random, unless of course the file has been written in sorted order. Assume that a file has been created to store details of cars in a car pool. The actual structure of the files record is not relevant except that it must store the

registration number of the car, car type, date purchased and cost of purchase. For the purposes of this exercise we will not actually access the file. We will revisit this problem in chapter 8.

Assume that an index file needs to be used to allow access to the data records using the registration number. The index file therefore needs to store a pair of values i.e. the registration number and the physical record number e.g. 123ERT, 34 indicates that the required record is stored in record position 34.

Write a program that accepts as input the registration number and an integer that represents the record position, and adds this detail to a linked list in sorted order by the registration number. The program should be able to locate and display the record position of a particular registration number, allow a new pair to be added and to allow a registration number to be removed from the list.



©IBO 5 **3.6 OTHER STYLES OF LINKED LISTS**

There are two other styles of linked lists. Each is now briefly considered.

A double linked list is a linked list where each node has two pointers. One points forward and the other backwards. The list can therefore be traversed in either a forward or backwards direction.

A suitable node description for a double linked list is shown below:

Class Node

```
int age;
String name;
Node forward;
Node backward;
```

To set the forward pointer you use the same method as shown above. To set the backward pointer, you need to temporarily store the location of the preceding node and when you make the new node, make its backward pointer point to the previous node.

The other linked list structure is the circular list. This is a linked list that has the last node point to the first node. This list does not use a null pointer value for the last node to point, but points the last node to point at the start of the list.

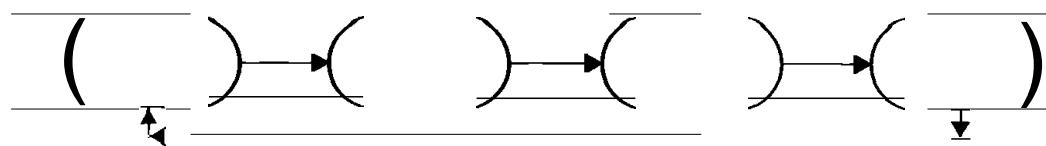
A diagram of each structure is shown below.

Figure: 5.7 - Linked Lists

Doubly linked list - allows movement backwards and forwards through the list



Circular List



EXERCISE 5.10

1. Implement a doubly linked list using the node definition as shown above. Include a way to move back or forward through the list after locating an item and insert data items in sorted order.
2. Construct a series of diagrams to show how the following data would look if it was stored in a simple linked list, a doubly linked list and a circular linked list. Assume that the data is to be stored in sorted order and that a head pointer is to be used. The data represents animals' names and the numbers of the memory locations. Data set (zebra:123, elephant:12, rabbit:345, horse:34:whale:23).



© IBO 2004 **5.3.7-8 DYNAMIC STACK**

The implementation of a dynamic stack is comparatively straightforward and easier to understand than the more complicated program discussed in the previous section.

To operate a dynamic stack we use a similar algorithm to the static version. The stack pointer would need to point to the last location added, in this case we will simply add the new data item to the head of the list and remove from the head.

There would be no need to check on overflow errors as the list length is now unbounded. However we should not try to pop past the end of the list.

An example of the operation of a dynamic stack is shown in the following algorithm.

The program defines the node structure to have two data parts i.e. data1 and data2 and a link reference called next. The head is declared as an object to reference the head of the stack. The initial status of the stack is to set the head to null. This indicates that the stack is empty.

The main program calls a sequence of pushes and pops and displays the status of the stack.

```

class Node

    int d;
    Node next;
    Node(int dIn, Node n)
    {
        d=dIn;
        next=n;

    }
public class StackDyn
{
    Node head = null;
    public static void main(String args[ ])
    {
        new StackDyn(1;

        StackDyn (I
    {

```

```

//add some values, insert is at head
add(21); add(2); add(1); add(121); add(21);
display(head);
int r = pop(head);
output("POPed value "+r);
r = pop(head);
output("POPed value "+r);
r=pop(head);
output("POPed value "+r);
output("Current status of stack");
display();
}
void add(int x)
{
    Node n = new Node(x, null); //make the node object
    n.next = head;//make new object point to what head did
    head = n;

int pop()

if (head == null)
    return -1;
else

    int c = head.d;
    head = head.next;
    return c;

void display()

Node c = head;
while (c!=null)
{
    output(c.d+" . ");
    c = c.next;
}

```

EXERCISE 5.11

1. Design some test data that will be entered into the program and determine a set of expected results.
2. Using the test data, perform a desk check (trace) on the program to check that the expected results actually occur.
3. Convert the previous reverse polish calculator to use the above dynamic stack data structures.



© IBO 2004 **5.3.9-10 DYNAMIC QUEUE**

A dynamic queue operates in the same way as a static version except that we do not need to worry about the queue becoming full in terms of the physical limitations of an array structure. If we wish to bound the queue, then we need to keep count of the number of active nodes.

We maintain a head to reference the first node in the queue. Items are removed from the front and added on using the end of the list. Items are added by searching for the end of the list, a tail pointer could also be maintained to facilitate this operations.

A major advantage offered by a dynamic queue is that the head node can be easily removed by moving the start or head pointer to point to the next node.

An example of a dynamic queue is shown below.

The front of the queue is referred to by the head and items are removed from the front of the queue i.e. like the pop() operation for a stack. However, items are added to the tail of the queue.

In this program a tail reference is not implemented, rather the end of Queue is searched for each time and the new node added at the end of the list i.e. the tail.

```
class Node

    int d;
    Node next;

    Node(int dIn, Node n)
    {
        d=dIn;
        next=n;

public class QueueDyn
{
    Node head;
    Node tail;
    public static void main (String args[ ] )
    {
        new QueueDyn ();

        QueueDyn ()
        {
            //add some values, insert is at head
            add(21); add(2); add(1); add(121); add(221);
            display();
            int r = remove();
            output ("\nvalue removed from Queue "+r );
            r = remove(); r=remove();
            output("Current status of Queue");
            display();
        }
    }
}
```

```

int remove ()

    Node n = head;
    head = n .next;
    return n.d;

void add(int x)

    Node n = new Node (x, null);
    if (head==null)
    {
        head = n;
        tail = n;
    }
    else

        tail.next = n;
        tail = n;

    }

void display ()

    Node c = head;
    c = c.next;
    while (c!=null)
    {
        output(c.d+" .. ");
        c = c.next;
    }

```

EXERCISE 5.12

1. Design some test data, and create some expected results which can be used to check the above program to see if the program outputs what is expected.
2. Use a dynamic structure to write a program to solve the customer complaints department queuing problem from the earlier section.



5.3.11-14 BINARY TREES

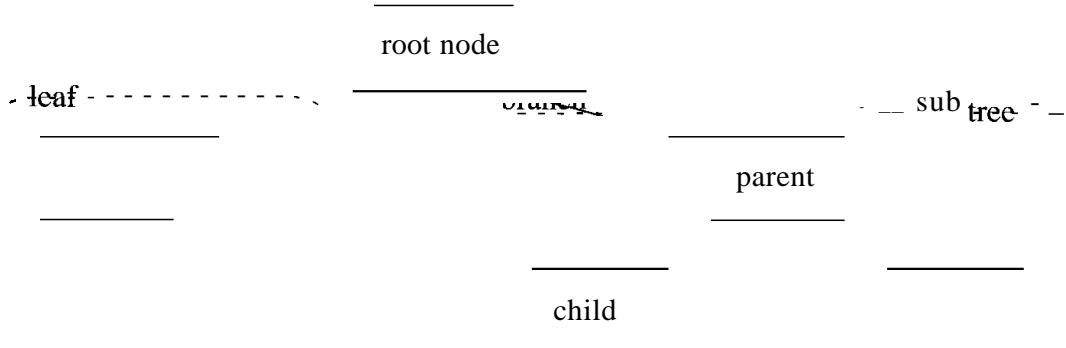
A linked list allows the list to grow dynamically, but unlike a static list, it cannot be searched efficiently using a binary search. The only way to search a linked list is to use a linear search. We can, however, use a special list structure called a binary tree to enable data in a list to be searched using the concept of the static array binary search technique.

General tree data structures are common in computer science. A good example is the structure of the folders or directories stored on a computer disk. The top level is commonly referred to as the root level and from this, many sub folders can be created. Each of these sub-folders can also point

to many lower level sub-folders. Within each folder a number of files can be stored.

Binary trees are a special tree structure. A diagram of a binary tree is shown below. The tree is made up of a series of nodes in which data is stored and left and right pointers are used. Each node can have at most two sub-nodes coming from it, referred to as the 'right child' and the 'left child'. These are pointed to by left (less than) and right (greater than) pointers. The top node is referred to as the 'root node', hence the term 'root node of the tree'. The lines linking one node to another node are called 'branches'. The links to nodes that have no further nodes are referred to as 'leaves' of the tree. The logical structure resembles a tree, whereas the physical structure is a sequence of memory locations with pointers.

Figure: 5.8



The general node structure is as shown below:

```
class NodeName  
  
    int dataPart;  
    NodeName leftPointer;  
    NodeName rightPointer;
```

Binary trees are very useful for storing dynamic lists of data which can be accessed in sorted order. They can be searched efficiently using a binary search technique. Trees structured in this way are known as 'binary search trees' and have the property that the left node is less than the parent node and the right node is greater than the parent node. Data items that are equal can go on either side provided, the algorithm used does this in a consistent manner.

EXAMPLE

Consider the problem of storing the following data set and then being able to search the set efficiently: Mary, Zork, Andrew, John, Ng, Claire, Jane, Brian.

SOLUTION

The general node structure would be as shown below and this can be shown diagrammatically.

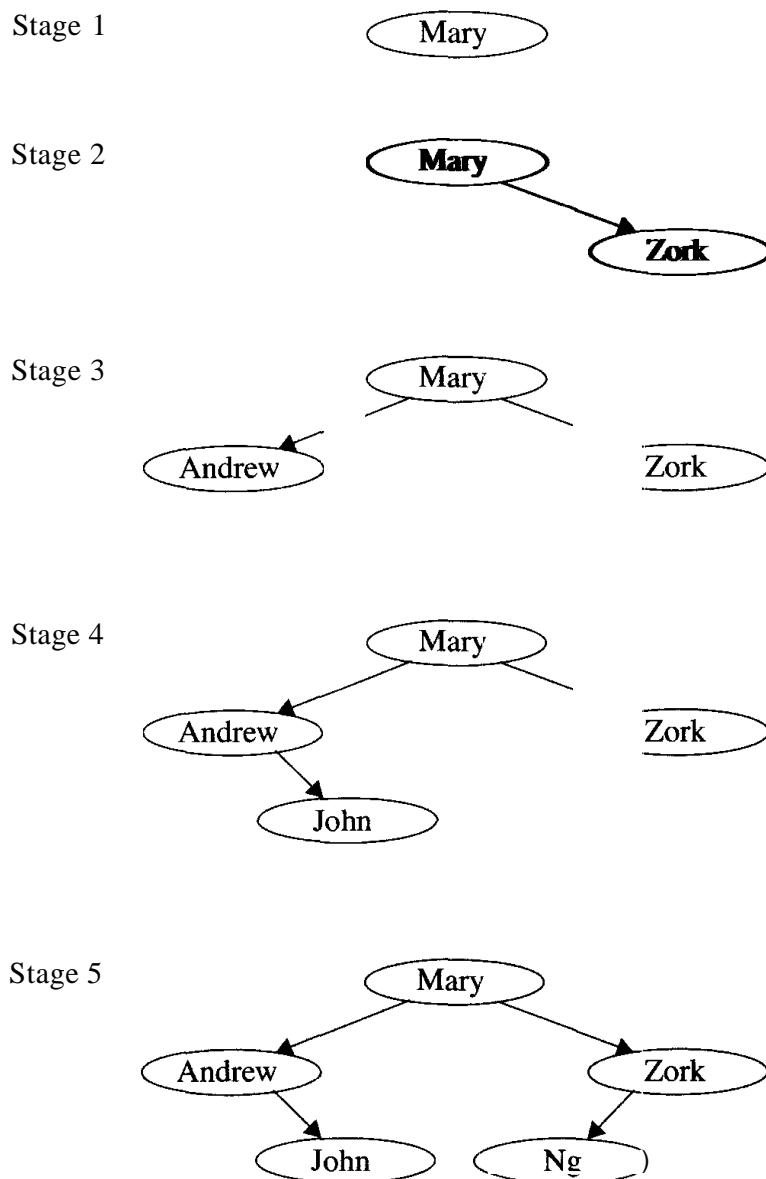
```
Class NameTree  
  
    String name;  
    NameTree left;  
    NameTree right;
```

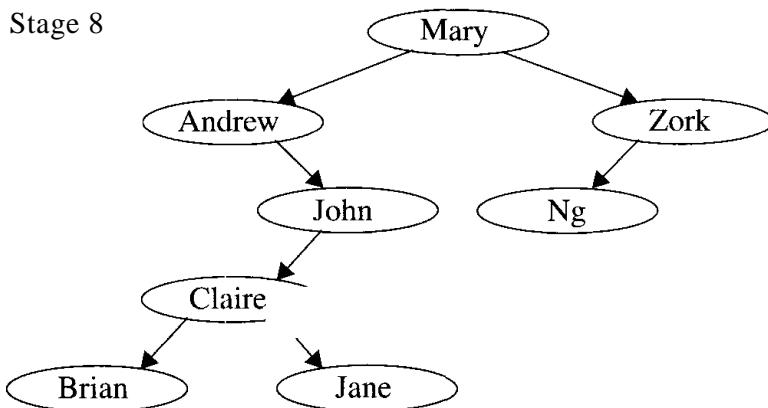
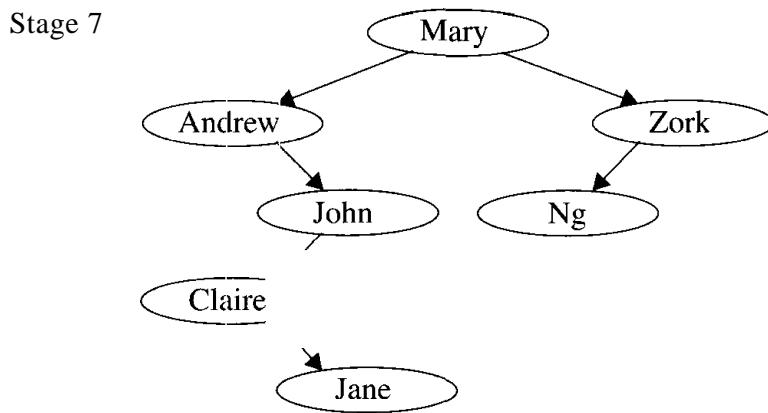
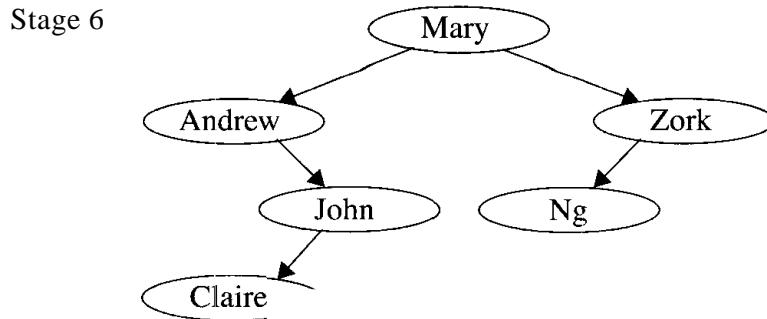
The placing of these names into a binary tree is shown in the diagram below. The diagram shows the stages of the tree as each node is added.

Mary is placed into the root node. The next name is placed on the right and Andrew to the left. John is placed to the left of Andrew and Ng to the right of Zork. Claire is placed to the left of John and Jane is placed to the right of Claire but on the left of John. Finally, Brian is placed to the right of Claire.

In each case we have placed the data item to the left or right of the above node value depending on the alphabetic order. Note that the tree needed no re-ordering or sorting as would be the case with a static array.

Figure: 5.9Figure 5.10





To search the tree we use the logic of deciding which branch to follow by asking the question 'is the data item alphabetically equal or to the left or right'. If it is equal the match has been found. If it is to the left and the left pointer is not null, move to the node pointed to and repeat the questions, otherwise move to the right node and repeat etc. To search a tree we use a recursive approach. This technique is discussed in more detail later. However, in simple terms, it allows a function to call itself and the full problem is solved by solving smaller ones.

EXERCISE 5.13

1. Draw a diagram to place the following numeric data into a binary tree. The numbers could represent a record number of data in a random access file.

45,26,11,40,56,30,44,34,112.

2. In general terms, describe what needs to be done to insert a new piece of data. i.e. search for the spot and insert the data into that spot.
3. What advantage does this structure have over a simple linear list? Can you quantify this advantage?



An algorithm to show both creating and searching the above numeric example is shown below.

The node created has the following structure

```
class Bnode
{
    int data;
    Bnode left;
    Bnode right;
}
```

The program operates by using a very simple interface to allow integer data to be input and inserted into the tree. A sentinel value of -999 is used to terminate input. The program then prompts the user to search for a data value.

```
class Bnode{
    int data;
    Bnode left;
    Bnode right;
};

public class btree{

    Bnode root= new Bnode();
    public static void main (String args[] )
    {
        new btree () ;

    public btree ()
    {
        int temp;
        root = null;
        temp = inputInt("Input data ");
        while (temp != -999)
        {
            Bnode newNode = new Bnode();
            newNode.data = temp;
            insert (root, newNode);
            display(root) ;
            temp = inputInt("Input data ");

            find = inputInt ("Input data value to find");
            int found = sTree(find);
            if (found == find) Output("found");
        }
    }
}
```

Advanced Data Structures and Algorithms

```
else Output("not found");

boolean isEmpty()

if (root == null)
    return true;
else
    return false;

int sTree(int d)

int returnFalse = -1;
Bnode current = root;
while (current!=null)

    if (current.data==d)
        return d;
    else if (current.data<d)
        current = current.right;
    else
        current = current.left;

    return returnFalse;

void display(Bnode current)

if (current !=null)
{
    display(current.left);
    output(current.data);
    display(current.right);

void insert(Bnode current, Bnode newNode)

if (root == null)
    root = newNocle;
else

    if (newNode.data<current.data)

        if (current.left==null)
            current.left=newNode;
        else

            current=current.left;
            insert (current, newNocle);
```

```

)
else

    if (current.right==null)
        current.right=newNode;
    else

        current=current.right;
        insert (current,  newNode);

```

Before we look at how the insert, print and search modules work it should be possible to work out what the tree will look like and, in general terms, how these operations work.

EXERCISE 5.14

1. Follow the logic of the algorithm and draw a diagram of the tree produced by the program. You should be able to do this even without knowing how insert and print and search actually work.
2. Perform the search operation i.e. sTree(23, root) and explain how you would do locate 23.
3. Assume the input from the above example is placed into a tree and we are going to use the output functions from the algorithm. Trace through how one of the output functions operates. One of them will return the list in sorted order.



'tRAVERSING A TREE

To insert or search or print out the contents of a tree we need to look through the tree. This process is called 'traversing' the tree. When you look at the tree you can locate what you want, but the computer program needs to follow an algorithm. We always start the traversal of a tree at the root node.

Consider the searching operation as shown by the function below. Note there are no duplicates allowed.

```

int sTree (int d){
    int returnFalse = -1;
    Bnode current = root;
    while (current!=null){
        if (current.data==d) return d;
        else if (current.data<d) current = current.right;
        else current = current.left;

    return returnFalse;
}

```

The value of the root is passed to the start and these are both of type tnode i.e. a tree node. The value being looked for is passed to the variable *d*.

If the tree is empty we abort the search and return -1 to indicate nothing is found. This is done by using a while loop with the condition *if start not equal to null* to continue to loop.

Inside the loop *if d is equal to the node data item* we stop and return 1. This means that control is returned to the calling program. Otherwise we need to ask "do we look down the left or right branch from the current node" and set start to either the current start->left or start->right value.

If the data value *d* is greater than the current value we reset start to be the value of the right branch pointer. Otherwise we set it to the value of the left branch pointer.

Now, the important bit. If we get to a position where either start->left or start->right are null this means that what we are looking for logically cannot be in the list because we have got to that part of the list by asking is what we want greater or less than the desired amount. If there is no further part of the tree to search, what we want must not be in the list and importantly, it obviously cannot be in any other part of the list either.

Let us now consider the **insert operation**.

The function to insert is shown below. The node *d* is the element to be inserted and the node *t* is the root of the tree.

The function uses recursion. In this case we can think of this process as saying "check the root first". Now look down the left or right branches. If the element has to be inserted into the left branch we check to see if there is anything down the left branch. If not we just make the left branch point to the desired node and return. Otherwise we want to call the insert function again and change the root to the next left branch node and repeat asking the question. We keep doing this until we insert the node.

The insert algorithm is not a general insert i.e. it does not cope with adding in the first element to the ROOT position because the algorithm does this explicitly. A general insert algorithm is shown below:

```
void insert (Bnode current, Bnode newNode){

    if (root == null) root = newNode;
    else
        if (newNode.data < current.data)
            { if (current.left==null) current.left=newNode;
              else
                  (current=current.left; insert (current, newNode);}
        }
    else
        { if (current.right==null) current.right=newNode;
          else
              (current=current.right; insert (current, newNode);}
```

This algorithm operates in the same way but will insert the new node into the root if the current

root is null, thus it is a more general algorithm.

Let us now consider the print or display or complete tree traversal operation. This operation is not the same as the search operation because in this case we want to visit every node in the tree. In the exercise above you were asked to trace the workings of the output functions. One of these output the list in sorted order. The method display(Bnode current) performs this.

```
void display(Bnode current){
    if (current !=null)
        { display(current.left) ;
                output(current.data);
                display(current.right);
        }
```

If you did this successfully you needed to look down the left branch repeatedly until you reached the last left node that had no child. This must have the numerically smallest value i.e. 11. The next value is the node immediately above i.e. the root of the sub-tree, 23. We then needed to look down the right branch and repeated the previous sequence looking for the smallest element in the right sub-tree which is 40.

This traversal is known as the 'inorder traversal' and in this case will give the data back in sorted order -11 ,23,40, 45,56,112.

An inorder traversal follows the sequence 'look down the left branch, then the root, then look down the right branch'.

There are two other traversals that we meet in the next section. They are 'preorder' where we visit the root and then the left and the right sub-trees of each sub-tree and the 'postorder' where we visit the left and right sub trees and then the root of each sub-tree.

EXERCISE 5.15

1. Create a binary tree program to store the surnames of the people in your class and allow the user to ask if a name is in the list. You can add a menu to allow the operations to be requested. What will you do if there are two people with the same name?
2. Add a function to allow the names to be displayed in sorted order.
3. Higher level dossier problem. Consider the previous file index problem i.e. the car registration file index. It was not possible to search the linked list efficiently other than by use of a linear search. The search function in the above example effectively implements a binary search, and we know that this is much more efficient.

Again the node is the same except that it now has to have a left and right node pointer. Hence our node would be:

```
class IndexPointer
    String rego;
    int recordAddress;
    IndexPointer left;
    IndexPointer right;
```

Assume that you will enter the index item values from the keyboard.

Your program needs to be able to accept a registration number and then search the binary tree and display the recordAddress to the screen.



Using trees to evaluate arithmetic expressions

Another common use of a tree is to evaluate arithmetic expressions.

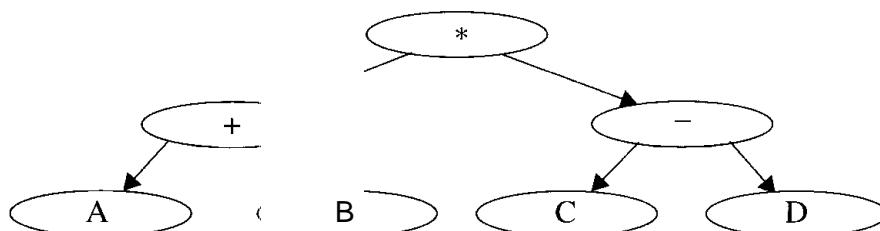
We have already seen that reverse polish notation can be evaluated using a stack.

For example, the expression $(A-t-B)*(C-D)$ can be written using post-order or reverse polish notation as $AB+CD-*$. As shown in a previous example, a stack can be used by popping two data values and then applying the operator that is popped next and repeating this until the list is finished.

A tree can be also used for this calculation.

To store the expression $(A+B)*(C-D)$ we would place the $*$ in the root position. To the left would go the $+$ and then to right and left of the $+$ would go A and B respectively. The $-$ would be placed to the right of the root node and then C and D would be placed to the left and right of the minus (-) node. A diagram of the resulting tree is shown below.

Figure: 5.10



To traverse (read through) the entire tree in a particular order we can use the post-order traversal to retrieve the expression in reverse polish notation or postfix notation.

Post-order traversal is accomplished by using the following algorithm strategy:

Visit the left sub-tree until a **null** pointer is reached and retrieve the data item.

Visit the right sub-tree until a **null** pointer is reached and retrieve the data item.

Visit the root of these left and right pointers and retrieve the data item.

By applying this strategy we would:

Look down the left sub-tree and retrieve A.

Look down the right sub-tree and retrieve B

Look at the root and retrieve the + sign.

Our list would be AB+.

We return to the root and look down the right sub-tree.

Look down the left and retrieve C.

Look down the right and retrieve D.

Look at the root and retrieve-.

Our list retrieved would now be AB+CD-.

Look at the root and retrieve *.

Our list retrieved would be AB+CD-*.

This list could be evaluated as we go or now treated as a stack.

There are two other ways to traverse a tree: pre-order and inorder.

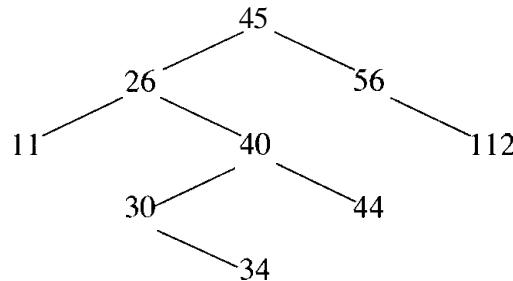
Inorder allows the data to be retrieved to give infix notation i.e. $(A+B)*(C-D)$. To do this we visit the left sub-tree, then the root, then the right subtree.

Pre-order allows the data to be retrieved in prefix notation. i.e. $*+AB-CD$. To do this we visit the root, then the left sub-tree, then the right-sub tree.

There is one operation that we have not covered. That is the **DELETE node operation**.

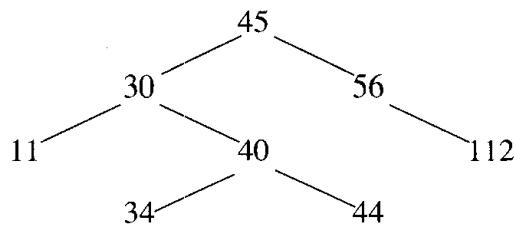
DELETE OPERATION

Consider the diagram shown below. To delete the node with the data value of 26 we could redraw the tree as shown. In this case the node has two children i.e. a left and right. So, what do we do? We look down the right branch for the child with the minimum value i.e. 30.



The node with 30 must have only a right branch because it is the minimum in that sub-tree. We now set the node with 26 in it to have the data value of the minimum i.e. 30. But what do we do with the right hand branch of the current node with 30 in it? We make the node that points to this, point to the right and child of 30 as its left hand pointer. Now 40 points to 34 on the left and 44 on the right! And, we can do this by calling the processes again to treat 30 as the new node to be deleted in that part of the sub-tree i.e. treat 30 as the root.

We now have.



But how exactly do we do that? We use recursion, though this is a bit tricky! The functions are shown below.

```

Bnode delNode(int d, Bnode current)
{
    if (current==null) return current;
    if (d<current.data)
        current.left=delNode(d, current.left);
    else if (d>current.data)
        current.right=delNode(d, current.right);
    else if (current.left !=null && current.right !=null)
    {
        current.data=locateMinValue(current.right) .data;
        current.right=delNode(current.data, current.right);
    }
    else
        if (current.left!=null) current=current.left;
        else current=current.right;
    return current;
}
Bnode locateMinValue(Bnode current)
{
    if (current==null) return null;
    else if (current.right==null) return current;
    return locateMinValue(current.left);
}
  
```

This works by finding the item and then looking down the right branch to find the minimum. The data value of the minimum replaces the found item. We then need to delete the minimum and, if there is a right branch, we then repeat the process until what we delete has no left or right child.

EXERCISE 5.16

1. Higher level dossier question. Modify the index binary tree example in the previous set of exercises to include the ability to delete a node from the index tree.
2. Word lists often need to be stored in sorted order to allow efficient searching of the list to check for matches e.g. a dictionary. Create a simple dictionary program that reads a set of 10 words from a sequential file and stores the words in a binary tree. Add the ability to search the tree and report back if an input word is in the list.
3. An organisation operates a stock purchase service via the Internet. A user logs in by usc of a username and password. The customer can then enter the stock code (e.g. IBM) of the

company and the quantity of the shares they wish to sell. A sample entry would be 'IBM, 20' which indicates that the customer wishes to sell 20 IBM shares. Instead of executing each share transaction as it is recorded, the program allows the customer user to enter a number of share trades in anyone session. Thus, the customer can add and delete shares traded from anyone session. A binary tree is to be used to store the share trades and the program needs to be able to allow the addition and deletion of share trades. Write a program to simulate the operation of this share trading software.

4. The local zoo needs to be able to search a list of the animals they have in the zoo to find out the age of the animal to assist with feeding. Make up a list of 10 animal names assuming that there is only one of each animal. The ages are stored in an array called AGES. The age of each animal will be stored in the array location that corresponds to the animal's name hashkey value. You will need to explore ways to set up the hashed array so that no clashes occur. A second dynamic data structure, i.e. a binary tree structure, is going to be used to search on the animal's name. The search will operate by the user entering the animal's name and the program displaying the age of the animal. The binary tree will need to be created by reading the array at the startup of the program. The tree can be searched and, if a match is found, the hashtotal is calculated, and the array accessed to find the age. To keep things simple, just hard code the animal's age at the appropriate place in the array. Note: this last question introduces the idea of using an index. The array is simulating a direct access file structure and the binary tree is providing a way to search the index using the primary key to get the record position.



LOGICAL AND PHYSICAL REPRESENTATIONS OF DYNAMIC DATA STRUCTURES

Dynamic data structures can be represented logically using diagrams as shown above. However, in the computer, the physical structure is maintained using a sequence of pointers. In the preceding examples of binary trees and linked lists, we saw that a logical diagram could be drawn that showed the structure. In the case of the binary tree this structure was very different from the physical representation, which is only a list of memory addresses. The programmer adds the structure by the use of pointers.

Linked lists can also be represented using two sets of arrays. One array holds the data and the other holds the pointer. The example shows a sequence of numbers in random order, but a linked list is created using a link array of the data array addresses.

Figure: 5.11

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|--------|------|--------|------|-----|--------|------|-------|
| Mark | Zork | Andrew | John | Ng | Claire | Jane | Brian |
| link[] | | | | | | | |

| | | | | | | | |
|---|---------|---|---|---|---|---|---|
| 4 | -l(end) | 7 | 0 | 1 | 6 | 3 | 5 |
|---|---------|---|---|---|---|---|---|

head = 2

The physical structure is simply a list of sequential array memory locations. The logical structure is the linked list: Andrew->Brian->Claire->Jane->John->Mark->Zork.

© IBO 2004 **5.4 OBJECTS IN PROBLEM SOLUTIONS**

© IBO 2004 **5.4.1 FEATURES OF AN OBJECT**

In normal computer programs data is defined and stored separately from the functions that process the data. Data types such as 'integer' or 'string' define how the data will be stored in memory and allow reference to the data using a variable name. User defined data types such as for a record structure or pointer structure allow related data of different data types to be collected together. The data can be structured in a variety of ways such as arrays, linked list, files etc.

Functions or procedures can then be defined to operate on the data. Functions and procedures can communicate with each other by passing data to and from each other or by returning a particular value that is either output or assigned for further use. Functions are distinguished from procedures by the fact that they return a value, whereas a procedure performs some processing but does not return a value.

Libraries of functions are usually available to allow programmers to access standard processing functions or procedures. The workings of the code in these library functions is hidden from the program developer.

An object allows the programmer to develop or access computer code that combines both data and operations into one unit. The object can be given data that acts as an initial input but, from that point on, the operation of the object is hidden from the programmer. The theory is that, by using objects, we should be able to increase reusability of code and reduce errors.

The data items of an object are referred to as 'data members'. Data can be completely hidden from the programmer so that it is impossible to access the data values, to see them or change them, without the use of an accessor method or edit/change method. Data provided to an object is often treated this way and is assigned to data members within the method thus hiding the data, i.e. made private to the object.

Functions defined within the object are referred to as 'member functions' or 'methods'. These can call other objects and their operation is hidden from the programmer. Member functions can be used to report data values from within the function or to provide internal processing.

Objects are described using algorithms or computer code that can be thought of as providing a blue print for the operation of the object. In this section on OOP we will refer to the description of the object as the CLASS description. The class is a bit like the plans of a building. But the object can only come into use when the programmer asks for the object to be constructed using the class definition. When the object has been constructed, memory is allocated and the programmer can use the functionality of the object. Objects therefore need to contain a special member function called a 'constructor'. The constructor executes to build the object and make available its functionality. Whilst an object is in use it takes up memory and thus when it is no longer required this memory needs to be reclaimed. A separate function called a 'destructor' needs to be called to do this and usually operates hidden from the programmer.

Let us consider a general example of a data list of real numbers ('doubles' in Java) and the standard operations you may wish to undertake on the list.

Design

Object Name **RealList**

Constructor:

RealList(data list, number of items in the list) accepts data

```

array and stores in local
array that is hidden.
class methods
print(): Print list
average(): Calculate and return average
max() or min(): Calculate maximum and minimum
sort()
displaySort(): Display sorted order

```

We could, of course, define many more member functions or methods.

To use the object we would have to create a new version of the object by calling the constructor and giving it the array of data. For example, we could say something like this in our program: `realList ages = new realList(data, n)`. This calls the constructor and passes the data list to the object i.e. an instance of the object has been created and can now be used.

To access the methods, we would use the name of the object, which is 'ages', and call the required member function (method) e.g. `ages.print()` would display the data or `output(ages.max())` would output the max value in the array.

A full discussion of the basis for using objects is beyond the scope of this book, however, there are many excellent references and students are encouraged to explore these. The notion of ABSTRACTION is important. The above object is an abstraction, or generalised view, of a list of real numbers. It is not particular to a specific problem or list. The programmer is free to re-use the abstract nature of the object in anyone of a number of particular ways.

Such an object could be used to process and type a list of real numbers. For example: temperatures, distances, heat rates, stock fluctuations etc.

We use the process of abstraction all the time when we make databases in which we define the generalised view of the entity (e.g. 'student' in a school's database). In this use of the term 'abstraction', we view the student in a general way, but only in a way useful to solve the database problem. We are not interested in a number of 'things or characteristics' of the students other than their personal details, subjects or marks.

Once the data had been supplied to the object it cannot be accessed directly if the data in the object has been declared to be private. Information could be gained from the object by asking it questions i.e. using the methods available in the object. The questions would take the form of calling member functions that were available to the programmer. But, only the results of the functions are available. The programmer cannot get at the code in any way.

© IBO 2004 5.4.2 ENCAPSULATION

The above object example shows the concept of encapsulation because both the data and the member functions or methods are combined into a single unit referred to as 'ages'. The primary purpose of encapsulation is that it hides data and information about how the object works. The programmer simply needs to use the functionality provided and can concentrate on what has to happen, not how it is to happen.

© IBO 2004 5.4.3 BASIC FEATURES AND ADVANTAGES OF INFORMATION AND DATA HIDING

An object contains both data and the ways to manipulate this data. The user or client of the object need only know what the object can do to be able to use it. In the example above the list object is an abstraction of the basic features of all lists. The user of the object can therefore use the object

at a more abstract and less detailed level to solve their problem.

In OOPS terms, information hiding (as this encapsulation of code and data is known in computer science) makes for highly portable, easily modifiable and safer software that is less prone to errors. Large applications may be easily maintained since objects may be updated, recompiled, tested and used as required.

© IBO 2004 **5.4.4 POLYMORPHISM**

Polymorphism is a term used in a number of related ways. The use of the + operator for adding real numbers and integers and concatenating strings employs the same syntax. This is an example of polymorphism known as 'overloading' that is available in traditional imperative programming languages.

The objects behave according to the data types presented, i.e. the pattern of the data types, without the programmer needing to perform any extra steps.

Objects can have a number of different constructions that all have the same name, but which may have different parameters. The generalised list example uses an array.

The array may be full, in which case we need not pass a value for n and create the object by the call RealList ages = new RealList(ages) or we may have used 0 to n elements, in which case a call could be RealList ages =new RealList(ages, n). Alternatively, we may wish to treat a sub-list, in which case we have RealList ages =new RealList(ages, low, high).

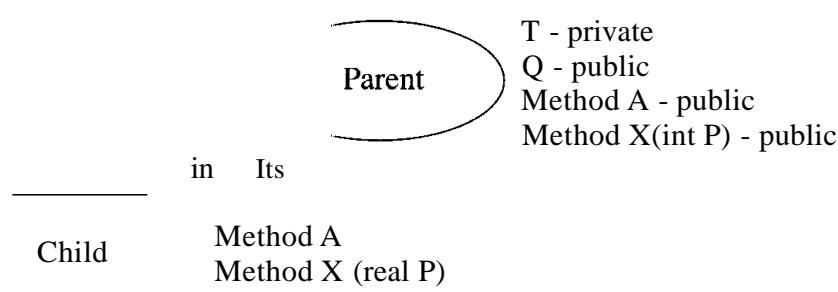
© IBO 2004 **5.4.5 INHERITANCE**

Inheritance in object-oriented programming is the ability to derive new classes from existing definitions of classes. A derived class or subclass inherits the data and functions or methods of the base class or superclass, and may add new variables and functions or methods. New methods or functions that have the same name override those in the superclass.

The major advantage is that the programmer is able to extend the functionality of an object or customise objects without needing to understand how the original class definition of the class was structured.

The diagram below shows the basic concept of inheritance. The parent class defines two variables, one private and one public. The child class inherits the characteristics of the parent and hence can access the public data but not the private data. It can also access the method Y from the parent. The methods X(int P) of the parent and X(real P) of the child can be selected depending on the data type of the argument provided by use of the polymorphic overloading feature of OOPs. The method A in the child over-rides the same method in the parent because the data types of the arguments are the same.

Figure: 5.12



EXERCISE 5.17

1. Define the terms 'class' and 'object'.
2. Describe in general terms the process of creating an object from a class.
3. What is meant by the term encapsulation?
4. What is meant by the term polymorphism?
5. What is meant by the term inheritance?
6. Use the diagram of a child and parent class to answer the following:
 - a) What data is private to the parent?
 - b) What methods are available in the child?
 - c) Which method can be overloaded?
 - d) Which method can be overridden?



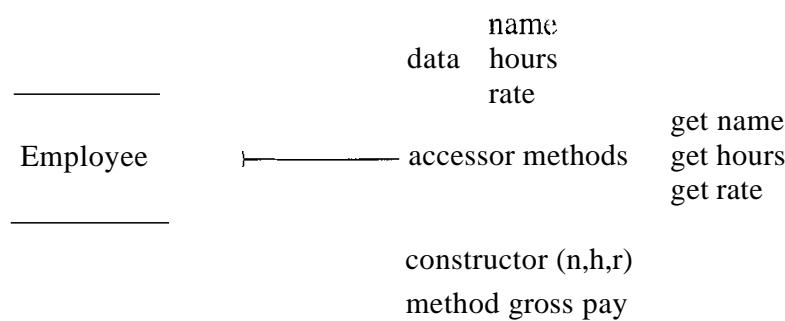
© IBO 2004 **5.4.6 ALGORITHMS WITH OBJECTS**

An example of how to define and use objects is now discussed.

Pay calculation

Assume that a normal employee has their gross pay calculated by multiplying the hours worked by the hourly rate. An object for this could be described by the following class diagram design. The diagram shows the data constructor i.e. EMPLOYEE and the range of accessor methods to retrieve the data and a method to calculate the gross pay. Obviously the pay could be directly calculated without the need of a method.

Figure: 5.13



The class definition derived from this design is shown below. Note: imperative programmers often feel that object definitions are long and complex because accessor and edit/change methods have to be provided. That is, the traditional assignment statement is replaced in many cases by calls to the object's methods if the data has to be accessed. This is a feature of OOP because encapsulation is used to protect data from accidental change.

```

public class Employee
{
    private String name;
    private double hours;
    private double rate;
  
```

```
public Employee (String nameIn, double hoursIn, double rateIn)
{
    this.name = nameIn;
    this.hours = hoursIn;
    this.rate = rateIn;

    String getName()
    {
        return name;

    double getHours()

        return hours;

    double getRate()

        return rate;

    double grossPay()
    {
        return hours * rate;
```

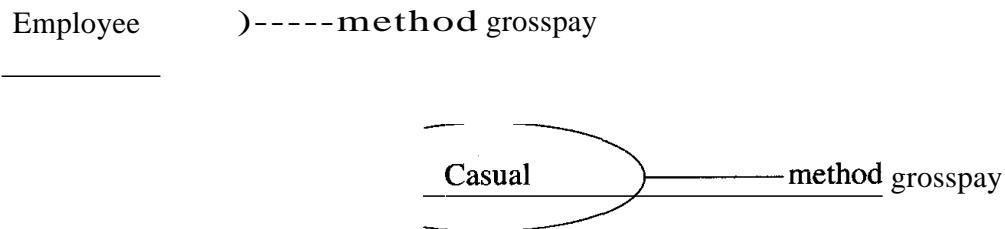
We could now use this object to calculate a person's pay. An algorithm to do this is shown below.

```
public class DoPay
{
    public static void main(String args [] )
    {
        Employee emp = new Employee("Fred", 34, 23.55);
        outputDouble (emp.grossPay());
```

The line `Employee emp = new Employee("Fred", 34, 23.55);` creates a new instance of the class `Employee` i.e. it creates or instantiates the object of type `Employee` by calling the constructor `Employee()` and passing the data to the object. The data cannot be directly accessed other than by use of the accessor methods because the instance data variables are declared to be private. The `grosspay()` method can be called directly using the dot notation by specifying the object type i.e. `emp.grossPay()`.

What if we also had a casual category of employee who received a 10% loading on their hourly rate?

We could write a separate program or we could extend the current class and derive a new class called `casual`. A design for this class is shown below and the algorithm description follows. The key feature is the addition of a new `grosspay()` method which will over ride the `grosspay()` method in the `Employee` class.

Figure: 5.14

The algorithm derived from this design is as follows. Note: Casual inherits the features of Employee but we need a way of creating the object Employee and giving it the necessary data. That is, the data name, rate and hours is held in Employee, not Casual. Java does this by the use of the super(parameter list) command as shown below. The child class instantiates the parent class and does this automatically. The super command ensures that the appropriate constructor is called. Note, super() must be the first line in the constructor.

```

public class Casual extends Employee
{
    private double loading;

    public Casual(String nameln, double rateln, double hoursln, double loadingln)
    {
        super(nameln, rateln, hoursln);
        loading = loadingln;
    }

    public double grossPay()
    {
        return loading * getRate() * getHours();
    }
}
  
```

To use this object we would employ an algorithm such as that shown below.

```

public class DoPay
{
    public static void main(String args[])
    {
        Casual emp = new Casual("Fred", 34, 23.55, 1.10);
        outputDouble(emp.grossPay());
    }
}
  
```

The key here is that the grosspay() method is called from the Casual object not the Employee object, however, the format for the call is exactly the same. This is an example of polymorphism using overriding of the parent method by the method in the child.

EXERCISE 5.18

1. Implement the above set of algorithms.
2. A school has established a policy whereby every student from age 12 to 15 years is expected to do one hour of homework per night. However, students older than 15 are expected to do 3 hours homework per night. Write a program that enables a user to enter a person's name and age and then to display how many hours of homework s/he should be doing.
3. A car hire company operates three sorts of cars: 4 cylinder, 6 cylinder and 4WD. The daily rates are \$50, \$80 and \$150 respectively. The company records the name of the hirer, the car and the number of days hired. Use OOP techniques to define a class for a standard car and then use inheritance to define another set of child classes to enable the rental to be calculated.



ADVANTAGES AND DISADVANTAGES OF OBJECTS

Advantages

- Faster development.
- Increased quality of software, more robust.
- Easier maintenance.
- Implementation details are hidden.
- Enhances reusability and modification.

Disadvantages

The fact that an object hides information about how it operates can cause problems for the programmer. In teaching examples, the programmer often develops the object, but in real situations, the programmer will probably be part of a team. An object may use a particular method that is inefficient or worse and which operates incorrectly for some types of data or ranges of data. Without access to the object code the programmer is unable to rectify these types of problems.

Many computer languages do not support objects either directly or via some form of interface. To move from a non-object orientated environment would be costly in both money and time.

Very simple programs can take longer to construct using objects.

There is still a lack of a standarised approach as to how to implement objects.

© IBO 2004 5.5 RECURSION

We have already used recursion in the discussions about the use of the binary search, quick sort and tree traversal.

© IBO 2004 5.5.1 DEFINITION OF RECURSION

Recursion is the process of repeatedly calling a function from within the definition of the function itself.

Recursive functions are derived from recursive relationships. For example, factorial n can be defined in terms of the number n itself as factorial $n! = n * (n-1) * (n-2)$ etc. This could be written as $n! = n * (n-1)!$.

Recursive functions must have a condition defined that needs to be met to terminate the recursive sequence of calls to the function. In the factorial example it is important to stop when n calls have been made otherwise zero will be included in the multiplication or the process will not stop and cause an error as available memory is exhausted.

When the recursive process is terminated, the computer evaluates each call in sequence until the final calculation is performed. This process is best thought of as backing out of the nested set of calls and evaluating as you go.

Let us use the example of calculating factorial n .

$n! = n * (n-1) * (n-2) * \dots * 1$, which is $n^* (n-1)!$ etc.

The stopping condition is to stop when n is less than 1 i.e. 0.

A mathematical (non-Java) recursive definition is described below:

```
factorial (n : integer)

if n <= 1 then factorial = 1
else factorial = n * factorial (n-1)
```

The recursive function has a name and its parameter is the value of $n!$. The 'if statement' checks to see if the termination condition is true. If it is, then the value of the function is set to 1 and this causes the recursive sequence to end and the backing out evaluation sequence to begin. Notice that when factorial is set to 1 there is no recursive call made and the sequence ends.

If the termination condition is not met, another recursive call is made and the value of n is decremented by 1, but the value of n is included in the multiplication sequence.

Let us apply the algorithm to calculate factorial 4 i.e. $4!$.

The calls would look like this:

```
factorial(4)
4 * factorial (3)
4 * (3 * factorial (2))
4* (3* (2* factorial (1)))
4* (3* (2* (1)))
```

Advanced Data Structures and Algorithms

The sequence of calls terminates and the calculations are now performed and the value 24 is returned as the value of the function.

Two Java methods of implementing this recursive calling process are shown below. The first uses iteration and the second uses recursion.

```
int factorial (int n)

int f = 1, i = 1;
while (i <= n)

    f = f * i;
    i=i+1;

return f;
```

The function accepts the value n which is used to stop the loop. A second variable (i) is used to multiply with and a third variable f , originally initialised to 1, is used to accumulate the value of the factorial. The value of the factorial (j) is returned when the counter i exceeds the value of n .

A function that uses recursion in Java to compute factorial n is shown below.

```
public class Recursion
{
    public static void main (String args[])
    {
        new Recursion();
    }
    public Recursion()
    {
        System.out.println("factorial 4      "+factorial(4));
    }
    int factorial (int n)
    {
        if (n==1)
            return n;

        else
            return n*factorial(n-1);
    }
}
```

The terminating condition is to check when n is equal to 1.

If this is the case the value of n is returned, otherwise the value of n is multiplied by what is returned by the recursive call.

factorial(4) calls the function and passes 4.

4*factorial(3) is then called, which in turn calls

4*3*factorial(2), which then calls 4 * 3 * 2 * factorial(1). This terminates the calling

cycle and the computation $4 * 3 * 2 * 1$ is evaluated to give 24.

The traversing of a binary tree is done using recursion (see previous example page 304). Also, to look through a linked list, recursion can be used.

©IB05 5-2 ADVANTAGES AND DISADVANTAGES OF RECURSION

Why would you consider using recursion? Many problems have elegant solutions using recursion. Recursive algorithms require less variables and often many fewer lines of code than other methods. There is a class of problems which have no other way of being solved.

However, recursion can take longer than other methods. Each call has a time overhead and this may lead to a recursive procedure actually being slower than its iterative equivalent.

Recursion can also be tricky to follow. This can cause problems for maintenance programmers. Thus it is particularly important to clearly document how recursion is being used so that other programmers can understand your code.

©IBO 5.5.3-5 USING RECURSION

At this point the reader is asked to go back and review the use of recursion in the binary search, quick sort and tree traversals as examples of recursion. Three examples are now worked through to show how recursion can be used.

EXAMPLE 1: FIBONACCI NUMBERS

Fibonacci numbers form the Fibonacci sequence 1, 1, 2, 3, 5, 8, 13, 21, 34, ... and have the general definition given by

$$\begin{aligned}F_n &= 0, \text{ if } n=0 \\&= 1, \text{ if } n=1 \text{ or } 2 \\&= F_{n-1} + F_{n-2}, \text{ if } n>2\end{aligned}$$

This gives the recursive relationship $\text{Fib}(n) = \text{Fib}(n-1) + \text{fib}(n-2)$.

SOLUTION

A recursive algorithm for this relationship is defined below.

```
int fib(int n)
{
    if (n<2)
        return 1;
    else
        return fib(n-1) + fib(n-2);
```

EXAMPLE 3: ACKERMAN FUNCTION

Ackerman's function is defined as:

$$\begin{aligned}A(0, N) &= 1 \\A(1, 0) &= 2 \\A(M, 0) &= M+2, \text{ if } M > 1 \\A(M, N) &= A(A(M-1, N), N-1), \text{ if } M > 0 \text{ and } N > 0\end{aligned}$$

SOLUTION

As a function, this would be represented as follows:

```
int ack(int m, int n)

if (m == 2)
    return 1;
if (n == 0)

    if (m == 1)
        return 2;
    else
        return m+2;
}
else
    return ack(ack(m-1,n), n-1);
}
```

EXAMPLE 4: EXPONENTIAL TO A POWERING Xⁿ.

Write a program to evaluate x^n using recursion.

SOLUTION

An algorithm to achieve this is shown below.

```
int power (int x, int n)
{
    if (n == 0) return 1
    else return x * power(x, n-1)
```

EXAMPLE 4B: USING RECURSION TO COUNT THE NUMBER OF VALUES IN A LINKED LIST.

Using the previous linked list example, the following method can be used to count the number of elements in the list.

SOLUTION

An algorithm to achieve this is shown below. The method is called by pass the head node object

as the argument i.e. count(head). Head is assignment to the object current in the method. If current is null then the recursive process terminates by returning 0, otherwise the 1 + count(current.next) is called. The reference to the next node in the list is passed back. This process repeats until the null terminating reference is encountered. Each time a recursive call is made 1 is added!

```
int count(Node current)
{
    if (current == null) return 0;
    else
        return 1+count(current.next);
```

EXERCISE 5.19

1. Define the term 'recursion'.
2. Discuss the advantages and disadvantages of recursion.
3. Implement each of the above examples of recursion.
4. Perform a trace on the raising to power recursive solution.



5.6 ALGORITHM EVALUATION

5.6.1-2 EFFICIENCY OF ALGORITHMS

The time it takes a computer to execute a particular algorithm depends on the speed of the processor and the way the algorithm operates.

Thus, it is difficult to give precise times, but we can generally use some form of approximation to show the relative dimension of the time required, or RAM space needed, to execute an algorithm. We are interested in obtaining measures for two aspects of an algorithm's efficiency: impact on the time taken to process and the amount of RAM used by the algorithm.

For instance, if we consider the linear search, the time it takes to find something in the list is dependent on, or proportional to, the length of the list. Thus any consideration about the speed efficiency of a linear search needs to be judged in terms of the length of the list (n).

The speed (efficiency) of a linear search when applied to a list of size n has these possibilities:

- Best case is 1 comparison.
- Worst case is n comparisons.
- Average is $\frac{n}{2}$.

As n gets very big it does not matter whether we talk of n or $\frac{n}{2}$. And for this reason we say that

the time efficiency of a linear search is 'order n '. This means that the upper bound or worst case on the speed of processing is n .

Contrast this to the binary search. In a binary search the best case is 1 and the worst at maximum case is $\log_2 n$. We therefore say that the binary search has a time efficiency of 'order $\log_2 n$ '. For very large lists this is a large speed improvement when compared to the linear search. Thus the binary search is to be preferred, but, remember a binary search only operates on a sorted list and the time taken to sort must also be taken into consideration.

The efficiency of sorting algorithms can be compared by considering the number of comparisons and the number of swaps.

The bubble sort and selection sort both have two loops and these are related to n . Hence we can initially say that both will have a time efficiency of n^2 i.e. each inner loop step is repeated $n \times n$ times. Let us confirm this by looking at the number of comparisons made for both.

The first loop of the bubble sort makes $(n-1)$ comparisons and the next loop $(n-2)$ and so on until only one element is left.

Thus the number of comparisons is the sum of the series $(n-1) + (n-2) + (n-3) + \dots + 1$. This sums to

$$\frac{n(n-1)}{2}$$

$$\frac{n(n-1)}{2} = \frac{n^2 - n}{2} = \frac{n^2}{2} - \frac{n}{2}.$$

As n gets bigger the $\frac{n^2}{2}$ term has only a small effect thus we can

say that the number of comparisons tends to $\frac{n^2}{2}$. As n gets big n^2 and $\frac{n}{2}$ are very large and we can

thus use n^2 as the efficiency indicator.

Thus the efficiency of bubble sort and selection sort are said to be of order n^2 .

Remember that the bubble sort is useful for lists that are nearly sorted. You can pick up if the list is sorted by the fact that, if no swaps occur, then the list is sorted.

If we take into account the number of swap operations, we get a similar picture.

In the bubble sort there are 3 swaps each time we interchange values. Thus, in the worst case,

there would be 3 swaps for each comparison $\left(3\left(\frac{n^2}{2} - \frac{n}{2}\right)\right)$ and for the average case this would be divided by 2 to give $\left(\frac{3}{2}\left(\frac{n^2}{2} - \frac{n}{2}\right)\right)$.

As n tends to get big this would approach $\frac{3n^2}{2}$ which we can estimate as n^2 . The selection and insertion sorts are similar.

The quick sort is much faster than both these. Each time we perform a section of the quick sort on a partition we perform n comparisons. We also perform $\log_2 n$ divisions of the list. Thus the number of comparisons is equal to the number of comparisons performed per sub-list or partition. In mathematical terms we have an efficiency of $n \log_2 n$.

So far we have considered approximations by looking at the case where input is large because the list of data is long. We have also looked at the average and worst case situations.

Another rule of thumb worth mentioning is, when considering the efficiency aspects of the operation of a computer, consider only those operations that take the longest time or that operate most frequently.

The Big O notation

We have already mentioned the most significant results, but not in terms of the use of the 'Big O' notation.

The Big O notation allows us to use a mathematically based approximation system for stating the efficiency estimates. Its aim is to provide an estimate of the upper bound for time performance or memory usage.

Mathematically this can be stated in the following way.

*If f and g are time measures. Then f is $O(g)$ if there exists a number k such that the following holds
 $f(n) \leq kg(n)$*

This means the time complexity function is bounded as n gets big or approaches infinity.

Let us apply this to the bubble sort example.

We know that the time complexity function is $f(n) = \frac{n^2}{2} - \frac{n}{2}$. To use the Big O method we need to determine the upper bound of the complexity function $f(n)$ by solving the following inequality.

$$f(n) \leq kg(n)$$

$$\frac{n^2}{2} - \frac{n}{2} \leq kg(n)$$

The left hand side of this inequality is always going to be less than n^2 . Thus $k=1$ and $g(n) = n^2$. Thus we can say that the function f is $O(n^2)$, i.e. order of magnitude n^2 . This means that if the list doubles the length of time taken to sort the list increases four fold. The Big O method gives us an upper bound of the dimension of the 'fastness' given the parameter n , which is usually the length of the list of data.

It is often the case that time functions can be represented as polynomials. In determining the Big O function for larger more complex polynomials we can ignore constants and ignore lower-order terms. An example will help explain.

Consider a time function $f(n) = 4n^5 + 3n^4 + 2n^3 + n^2 + 5$.

This equation states that the time for the program to run is related to the length of the data set by the function $f(n)$, which in this case is the 5th order polynomial.

We know that $f(n) \leq 4n^5 + 3n^5 + 2n^5 + n^5 + 5n^5$, just by increasing the powers of the terms.

Thus $f(n) \leq 15n^5$, by collecting like terms. This satisfies our condition.

Hence we can say that f has the time complexity of $O(n^5)$.

Efficiency comparison of various ways to calculate Fibonacci numbers

In the recursive implementation of the Fibonacci sequence we have a situation where many redundant functional computations take place. In fact, in the worst case, we have that $fib(n) \sim O(2^n)$ repeated calculations. As functions use stack space in memory to store return addresses, a recursive solution is also a heavy user of memory.

It would be better if we calculated each element of the sequence only once.

We can improve the efficiency of the calculation by storing the terms of the sequence in an array as shown in the algorithm below, which has time complexity of $O(n)$. It also requires memory allocation directly related to the size of the list n i.e. $O(n)$.

```
void loadFibValues(int size)
{
    int [] fib = new int[ size ]
    fib[ 0] = 1;
    fib[ 1] = 1;
    for (int i=2; i<fib.length; i++)
        fib[ i] = fib[ i-1] + fib[ i-2] ;
```

To get the value of the 10th Fibonacci number you would access the index position 9 (remember we start at 0) of the fib array Le. $fib[10]$.

This is a much faster implementation but does require $O(n)$ memory space.

Summary of complexities

- $O(1)$: constant time not related to the length of the data set.
- $O(\log 2n)$: log time and is fast.
- $O(n)$: Linear time.
- $O(n \log 2n)$: $n \log n$ time.
- $O(n^2)$: quadratic time, polynomial complexity.
- $O(n^3)$: cubic time, polynomial complexity.
- $O(nn)$: not computable.

Infeasible or exponential time complexity measures include $O(2^n)$, $O(3^n)$, $O(n!)$. Algorithms with such characteristics shown can be avoided as it is likely that as n gets bigger a solution will not be possible.

Summary of common O complexities

- Linear Search $O(n)$
- Binary Search $O(\log 2n)$
- Bubble Sort $O(n^2)$
- Selection Sort $O(n^2)$
- Quick Sort $O(n \log 2n)$

In terms of algorithms involving loops the following rules of thumb apply:

- If there is a single loop the time complexity is related to n i.e. $O(n)$.
- If there are two loops the time complexity is $n * n$ i.e. $O(n^2)$.
- If there are three loops the time complexity is $n * n * n$ i.e. $O(n^3)$.

EXERCISE 5.20

1. State the efficiency of the various searches presented.
2. State the efficiency of each of the sorting algorithms presented.
3. For the algorithm presented below estimate the time complexity and storage space requirements i.e. what is the dimension of the array?

```
for (J = 0 to N) do
  for (K = N to 2N) do
    A[ K]      K+J
```

4. Show that an algorithm with a time function of $f(n) = 10n^3 - 4n^2 + 2n + 4$ has an order of magnitude estimated time of $O(n^3)$.
5. Verify, by drawing a tree structure to show the calculation, that a recursive Fibonacci function performs repeated calculation.



© IBO 2004

5.6.3-4 ORGANISING AND EVALUATING DATA STRUCTURES AND ALGORITHMS TO SUIT PROBLEMS

When selecting data structures, we need to be aware of the type of data to be stored, how it will be manipulated and how it will be stored in RAM and on disk. It is important to ensure that the chosen data structure facilitates efficient storage, access and processing.

Arrays are static structures but they allow direct access to data elements via the array index. Arrays have to be sized to account for the worst case to avoid out of bounds overflow errors, which means that large amounts of space may not be used. Addition to, and deletion from, the list facilitate a number of 'shuffling' actions that take time. A time measure is proportional to the length of the array or, to be more precise, the length of that part of the array affected. Hence we can say that an insertion or deletion has an order of complexity proportional to the length of the list i.e. $O(n)$, and $O(n)$ to locate the insertion point. Arrays allow data to be sorted and very efficiently searched by use of the binary search. Data in arrays can also be processed sequentially.

Data stored in a dynamic structure such as a linked list has no memory space limit placed on its length other than the available memory. Data can be easily inserted and deleted in basically one operation. It is certainly not related to the length of n i.e. it has a time complexity of $O(1)$. But the list must be searched for the location of the item in the list. This is also the case for elements in an array, unless some hashing algorithm is used to locate its index position directly. Thus both have $O(n)$ complexity in terms of locating the item. Dynamic structures are very efficient of memory because they grow and shrink to suit and do not need to be sized for the worst case scenario.

Data in an array can be sorted and searched using a binary search which has time complexity of $O(\log_2 n)$. However, data has to be kept in sorted order and, by using a quick sort, we can perform these operations in $O(n \log_2 n)$.

Data in a linked list can also be kept in sorted order but cannot be searched other than by using a linear search that has time complexity of $O(n)$. However, if searching was required, a binary tree dynamic structure could be used. Depending on need, this could be a separate structure i.e. doubling memory. A binary tree can be searched in $O(\log_2 n)$ time.

Consider the problem of working out which sequence of numbers has the greatest sum in a sequence of n positive and negative numbers.

For example, consider the list 1,6,-8,9,1. The possible sub-lists are enumerated below.

1=1
 1+6=7
 1+6+-8=-1
 1+6+-8+9 = 8
 1+6+-8+9+1 = 9

6
 6+-8 = -2
 6+-8+9 = 7
 6 + -8+9 + 1 = 8

-8
 -8+9 = 1
 -8+9+1=2

9
9+1 = 10
1

By this exhaustive process we have the maximum subset being $9+1=10$.

A slow way to solve this problem is to use three loops as shown below. The outer loop controls the starting position, the next loop controls how long each sub-set of numbers is and the innermost loop allows the numbers in the current sub-set to be totalled and compared to the maximum.

```
int maxSubset (int seq[])
{
    int max = 0;
    int sum = 0
    for (int start = 0; start<seq.length; start++)
        for (int end = start; end<seq.length; end++)
        {
            sum=0;
            for (int j = start; start<=end; j++)
                sum = sum + seq[j];
        }
        if (sum> max) max = sum;

    return sum;
```

The outer loop repeats n times and the inner loops repeat approximately $\frac{n}{2}$ times each.

We can apply the efficiency analysis as $n * \frac{n}{2} * \frac{n}{2} :: \frac{n^3}{4}$.

In terms of the Big **A** notation we have $\frac{n^3}{4} \leq n^3$, hence we can say that the algorithm has a time efficiency of $O(n^3)$.

We can improve this algorithm by noting that we can remove the second loop. It is not needed. All we need do is control a start position and then exhaustively look at each sub-set from that point. To do this we need only have an inner loop.

```
int maxSubset (int seq[])
{
    int max = 0;
    int sum = 0;
    for (int start = 0; start<seq.length; start++)
        for (int end = start; end<seq.length; end++)
    )
        sum =0;
        if (end == start) sum =0;
```

```

    else sum =sum + seq[ end]
}
if sum > max then max = sum

return sum;

```

This gives an efficiency improvement reduced to $O(n^2)$. We should also note that both the solutions above have a memory efficiency of $O(n)$ i.e. the data has to be stored as a list in memory.

It is important to try and concentrate on macro optimisation like this rather than fiddling with reducing the number of comparisons performed or if statement tests.

The principle of dynamic programming can be applied to reduce the time complexity even further. Dynamic programming works on the principle of solving large problems in terms of the solution to earlier smaller problems.

Look at the exhausted list of all possibilities. One way we could approach the problem is to keep a running total of the sequences and store the maximum as we go. At each stage we could test to see if the addition of a new value reduces the sum to less than zero. If it does we can ignore it as a possible candidate. The maximum to this point has already been stored so we can set our running total back to zero and continue. If the new total is positive, but reduced, we have no way of knowing if the next number added will make the sequence exceed the current maximum, hence we keep going. At the finish of each step we test to see if the maximum needs to be changed.

The new algorithm has time complexity of $O(11)$, which is a major improvement. An algorithmic implementation is shown below. Assume the array $n[]$ holds {1,6,-8,9,1};

```

int maxSubset(int seq[])
{
    int max = -999; prev =0;
    for (int i = 0; i < seq.length; i++)

        if (n[ i] + prev) > 0      prev = n[ i] + prev;
        else      prev = 0;

        if (prev > max)  max = prev;
    endif

    return sum;
}

```

The trace of this algorithm shows the following sequence of events.

```

prev=0
i=0 n[ 0] = 1
1 + 0 > 0 thus prev = 1
max = 1

i=1 n[ 1] = 6
1 + 6 > 0 thus prev = 7 (1+6)

```

max = 7

i=2 n[2] = -8
 -8 + 7 < 0 thus prev 0
 max 7 (stays put)

i=3 n[3] = 9
 9 + 0 > 0 thus prev 9 (9+0)
 max 9 (changes)

i=4 n[4] = 1
 1 + 9 > 0 thus prev 10 (1+9)
 max = 10

Thus the maximum sub-sequence is 10. The new algorithm has reduced the time complexity to $O(n)$ and in fact there is no need to store the numbers in the sequence.

EXERCISE 5.21

1. You are required to store up to 10,000,000 items in an appropriate data structure. You are required to add and delete items and search the data structure often. What are the complexity implications for different data structures e.g. linked list, array, binary tree etc.?
2. Study the efficiency issues associated with the following situation and recommend a possible course of action. It is necessary to store between 2,000,000 and 4,000,000 records in a file. Data records are frequently being added, deleted and edited.
3. Study the efficiency issues associated with the following situation and recommend a possible course of action. A file that contains 10,000,000 records is to be organised as a sequential access file. Over time, it is realised that 90% of the time only 100 records in the file are accessed.
4. Why would be it more important to make macro improvements to an algorithm process than to spend a lot of time making small micro level changes?
5. Contrast the efficiency of the two Fibonacci programs given previously, by comparing the recursive and iterative version that loaded the values into an array. The Java System class provides a method to record the current time: long startTime=System.currentTimeMillis(); Use this method to record the time taken by the two methods.
6. Use the currentTimeMillis() method to compare the efficiency of the various sorting and searching methods you have been given.
7. Create an array of random integer values and dimension the array to 1,000,000. Compare the time taken to locate the last value using a linear search as compared to a binary search.



6

Chapter contents

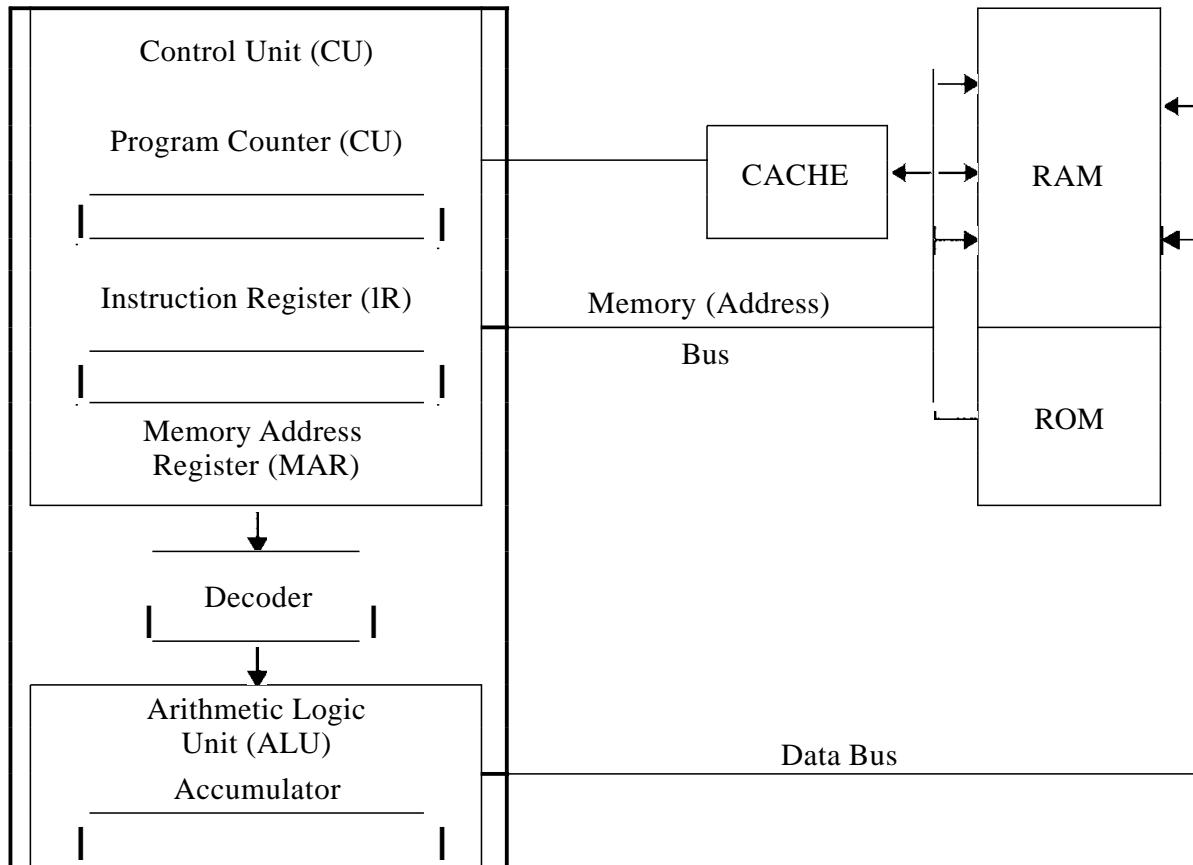
- 6.1 CPU Configuration
 - 6.2 Disk Storage
 - 6.3 Operating Systems and Utilities
 - 6.4 Further network fundamentals
 - 6.5 Computer/Peripheral/Communication
-

© IBO 2004

6.1 CPU CONFIGURATION

A basic model of a processor (central processing unit) is shown in the diagram below. The model shows the control unit, arithmetic logic unit and primary (or main) memory, more commonly referred to as RAM or random access memory.

Figure: 6.1



The CPU is the central part of the computer that performs most of the calculations. On a PC the CPU is a single chip or microprocessor housed on the 'mother board'. The mother board houses the processor unit or chip, RAM, cache, memory and data bus and connectors to the external bus and to other controller cards and I/O devices. On larger computer systems the CPU may be spread over a board containing more than one chip. The processor unit contains the control unit and arithmetic logic unit.

The control unit controls the operation of the CPU. It coordinates the retrieval of instructions from memory, decodes these and then executes them.

The arithmetic logic unit (ALU) performs all arithmetic operations, such as addition and subtraction and comparisons like tests for equality of the contents of memory locations.

Memory is divided into two separate parts: read only memory (ROM) and random access memory (RAM). The RAM of the computer is where the current data and executing program instructions are stored. Each location of RAM has an address and contents. The instructions and data are stored as the contents. Instructions are made up of two parts: the opcode and the operand. The contents of memory are transferred into the CPU registers via the data bus and the memory addresses are accessed via the memory address bus. RAM is a volatile storage area requiring an electrical current to maintain its state.

ROM is used to store permanent operating system instructions that are required to boot and

operate the computer. They cannot be changed. The code to boot the computer is stored in ROM and executes automatically when the computer starts. The code that operates for each interrupt is also stored in ROM in fixed positions.

© IBO 2004 **6.1.1/2 FUNCTIONS OF THE ACCUMULATOR, INSTRUCTION REGISTER AND PROGRAM COUNTER**

The control unit has a number of registers. A register is a location that temporarily stores data or memory addresses that are going to be used by the current cycle of the computer.

The basic registers in the control unit are:

- The program counter (PC) holds the address of the next instruction in the program sequence. It is assumed to be the next instruction and is automatically incremented, unless the executing instruction modifies the contents of this register via a jump or branch instruction. The automatic incrementing is a fundamental part of the design, or architecture, of the chip.
- The instruction register holds the opcode of the instruction that is about to execute i.e. the type of instruction e.g. ADD, MULT or STORE.
- The memory Address register holds the operand (memory address) of the data to be used by the instruction or the location to which data will be written by the instruction that is about to execute (Note: not required in HL).

The ALU has a special register called the 'accumulator'. This register holds the ongoing total of any calculations being performed.

The instructions stored in a computer are in machine code format and are a sequence of Is and Os. Machine code is code that is in a form ready to be executed directly by the hardware. In general, the process of compilation converts the higher level language source code into machine object code. Compilation creates an entirely new object code program and an interpreter converts source code a line at a time. A machine code instruction is made up of the opcode and operand. Machine code can be written using an assembler language that uses a mnemonic system to allow programmers to refer to an instruction opcode using mnemonics such as ADD to indicate the add instruction. Operands can be referred to by use of normal variable type names. There is a one to one correspondence between a single machine code instruction and an assembler instruction.

A simple assemble language could have these opcodes:

- LOAD memory location X: gets the contents of memory location X and overwrites what is in the accumulator.
- ADD memory location X: adds contents of X to accumulator.
- MULT memory location X: multiplies current contents of accumulator by X.
- STO memory location X: moves the contents of the accumulator to the contents of memory address X.

Using these sets of instructions we could write an assembler program to implement the logic of this assignment statement $Y=P + (A*B)$. Assume $P=2$, $A=3$ and $B=4$.

Our assembler code would look like this:

```

1000    LOAD  A
1001    MULT  B
1010    ADD   P
1011    STO   Y

```

Note that memory addresses have also been added. You can assume that these are the last four bits of longer 16 or 32 bit addresses.

The program would be loaded into RAM and the PC set to hold the address of the first instruction. The steps followed to execute the program are explained below.

© IBO 2004 6.1.3 FUNCTION OF THE INTERRUPT REGISTER

Interrupts are the way a processor is able to handle the demands made of its processing time. There are two types of interrupts: hardware interrupts and software interrupts. A processor is basically able to do one thing at a time.

When a processor is running a program, a number of other things can happen that may require the processor to stop what it is doing to service these requests. This is what happens when you are typing on the computer and the phone rings. You stop typing and answer the phone. When you have finished, you resume typing. The phone ring is acting as an interrupt.

Hardware interrupts are linked to the physical architecture of the computer and usually are designed to allow devices to communicate and gain the 'attention' of the processor when the device has a problem or wishes to send or receive data. A printer, for example, may wish to report an 'out of paper' error or a modem may wish to send a stream of data.

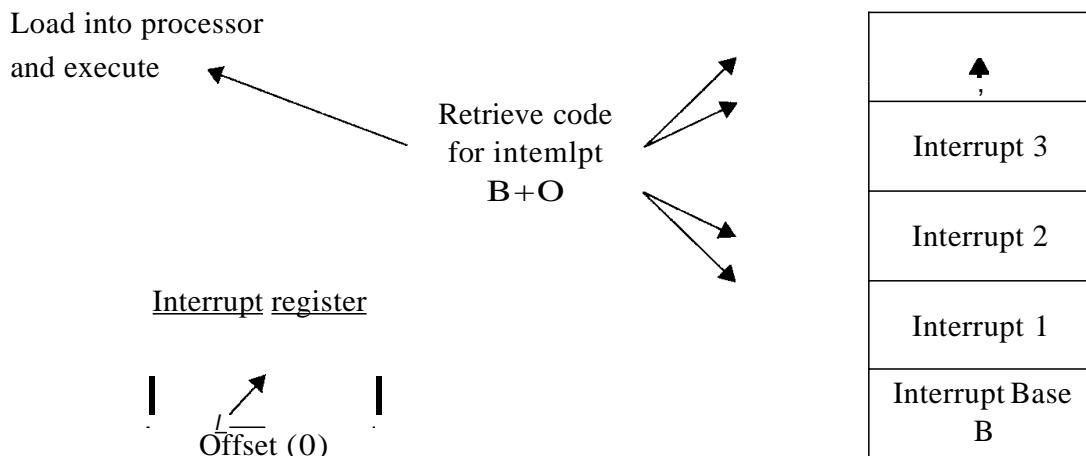
Software interrupts operate similarly to hardware interrupts except that the condition that needs responding to is generated by the processor itself. When an EVENT or EXCEPTION occurs that requires handling, a specific piece of software, designed for the purpose, is executed. Java, for instance, allows a range of exception errors to be handled. Examples include attempting to read past the end of a file or accessing an index beyond the dimension of an array.

The code to handle hardware interrupts is stored in memory in fixed positions starting from a fixed memory position or base address. To access the desired interrupt software, the memory offset position is required. When a device indicates an interrupt by sending a signal to the appropriate port, the interrupt register is updated with the offset position of the start of the handling code in memory. The address of the required interrupt code can then be calculated as:

$$\text{Interrupt Code Address} = \text{base address} + \text{interrupt register.}$$

When the interrupt is detected, the current states of the processor's registers are stored (PUSHED) onto a stack and the interrupt is executed. The processor returns to its prior state by POPPING the register values off the stack.

Figure: 6.2 - Interrupt Register Operation



©IB06.1.4 ROLE OF A BUS TO LINK THE PROCESSOR UNIT, 2004 RAM, ROM AND CACHE

What is a BUS?

A bus is a set of parallel wires that allow bits to be transmitted over one of the single wires. Buses can be internal or externally connected to the CPU and to the I/O ports of the main control board. For example, a disk drive controller is connected to the CPU via a bus. The CPU's control unit also controls the timing of operations via a control bus.

There are two main types of buses within the CPU.

Data Bus

Data is moved around within a processor via an internal bus called the 'data bus'. The registers of the CPU are linked to the contents parts of memory addresses so that the data held in the memory can be transferred to the appropriate register. For example, in the STO Y assembler command the contents of the accumulator are shipped to the contents part of the memory address Y via the data bus.

Memory Bus

When memory address (location) information is moved around inside the CPU it is moved via the memory bus. A control bus is also used to synchronise the activity of the CPU. Each wire of a bus matches to an individual bit in the address of a memory location or individual bit in the contents part. Thus a bus size determines the amount of memory that can be addressed. For example, in a 32 bit computer the memory bus needs to be 32 bits wide to enable all the 2^{32} different memory addresses to be accessed.

The role of CACHE

Memory cache is a 'speed up' mechanism. It is RAM memory made up of memory chips that are faster than the normal memory. This means that the data can be retrieved faster than from standard RAM. Cache is best thought of as sitting between the CPU and the main RAM as shown in the diagram below.

Figure: 6.3



The role of the cache is to store the most recently accessed memory addresses and their contents.

When an address is required, the first place the CPU looks is the cache. If it finds the address the operation continues using the data found. Otherwise the data is found by accessing the standard main RAM area. When data is found in the cache, it is referred to as a 'cache hit'. The higher this hit rate is, the faster the performance.

Most cache is internal and built into the architecture of the computer, however it is also common to have external cache available via expansion cards.

The way that data is updated in the cache also contributes to performance. The two main algorithms are referred to as 'write-back' or 'write-through'. The write back algorithm only updates changes to the data in the cache, whereas the write-through algorithm updates the data in cache as well as the main memory. Hence this is a bit slower but safer.

Further System Fundamentals

Data and memory addresses moved in and out of cache are effected using the appropriate bus.

Disk units can also operate a cache located on the disk controller. This speeds up access times.

Internet servers and PCs can also operate a cache by storing the most recently accessed web data. This speeds up operation if what is required is stored in the internet cache because it can be retrieved from the local storage area.

EXERCISE 6.1

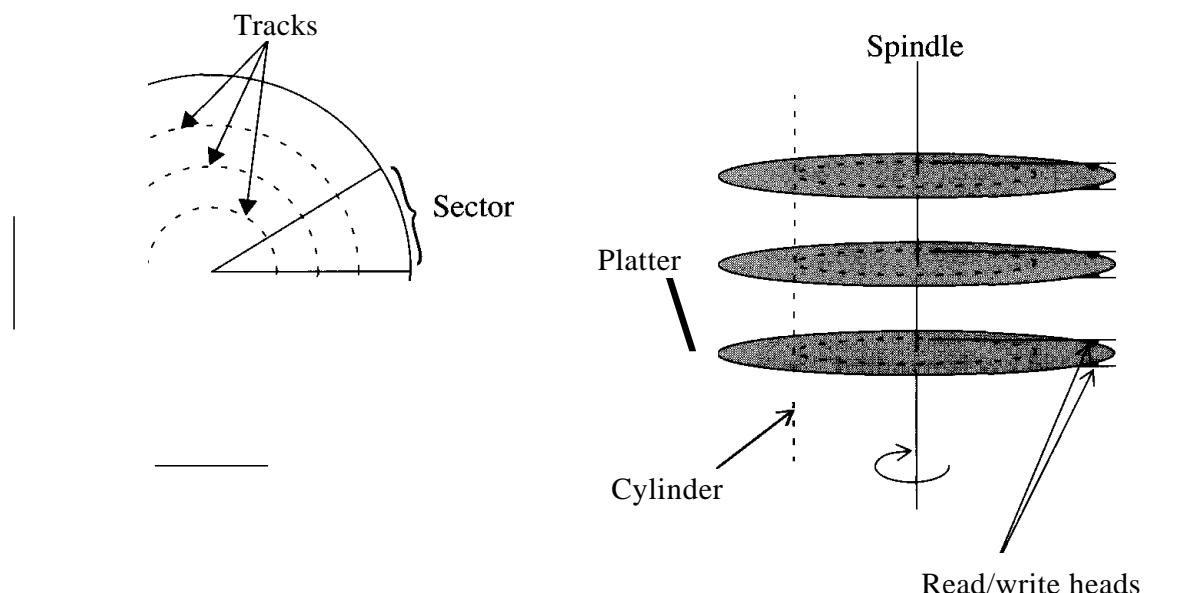
1. Describe the function of the accumulator.
2. Describe the function of the instruction register.
3. Describe the function of the program counter.
4. Describe the function of the interrupt register.
5. Describe how the buses link the processor to RAM, ROM and cache.
6. Draw and clearly label a diagram of the processor showing the major components and buses.



© IBO 2004 6.2 DISK STORAGE

Hard disks record data by magnetising the binary code on the surface of a disk. The data area is reusable, just like other magnetic media. Files can be deleted or re-written. Hard disks allow both sequential and direct access file organisation. Hard disks have many platters and both the top and the bottom surfaces of each platter, except for the top and bottom platters, are used.

Figure: 6.4



There are also fixed and removable disk drives.

The floppy disk, so called because the disk is flexible and housed in plastic, operates on the same principle as a hard disk. It has a single platter and is a convenient method of storage. It is cheap

and has a reasonable capacity for text based storage. However, access is slow, the storage is limited and the disks can easily be damaged. Access is also much slower than access to a hard disk.

There are also optical disk drives that allow data to be read from CD-ROMs. The data cannot be altered, only read by a laser beam shining on the surface. The binary bit pattern is encoded at the time of production (writing) by a stamping process. The plastic surface is stamped with pits and 'land areas'. The changes from pit to land or land to pit represent binary ones and zeros with the pits absorbing, and the land areas reflecting, more of the laser light.

There are also erasable optical disks referred to as magneto-optical disks that allow data to be read and written many times in a comparable fashion to a hard disk.

Hard disks can also be grouped together to form what is known as a 'Redundant Array of Independent Disks' or RAID for short. Such grouping allows very large amounts of storage that can be accessed quickly and which provides a range of data protection features.

When data is written to a RAID disk it can be spread across a set of disks. This improves performance by taking advantage of parallel access across the many disks. This feature can also be combined with disk mirroring which provides a fault tolerance feature. Disk mirroring means that the data is written to duplicate disks. This technique is vital if the system needs to be accessible at all times.

©IBO 2004 621 ROLE OF BLOCKING, SECTORS, CYLINDERS AND HEADS IN STORAGE

Blocking: the block size determines the number of bytes that are read and written in a single physical read or write operation on a hard disk. The blocking factor is usually more than one disk sector and is also known as 'cluster size'.

Sectors, Tracks and Cylinders

A disk surface is divided into a number of separate circular tracks and these are, in turn, divided into sectors. The collection of the same track on all the surfaces of the platters in a hard disk are referred to collectively as a cylinder of tracks i.e. the same track on each surface.

The capacity of a hard disk is determined by the number of tracks per surface, the number of sectors per track and the number of bits or bytes per sector.

A floppy disk has a single platter with a top and bottom surface. A high density floppy has 160 tracks on 2 sides with 9 sectors per track and 512 bytes per sector. The capacity is given by the equation below.

$$\text{capacity} = \text{number of surfaces} \times \text{number of tracks} \times \text{number sectors per track} \times \text{bytes per sector}$$

Data can be stored around the tracks in the sectors. Direct access is possible by specifying the required track and sector. In this way the data is retrieved without reference to the other related data. This is in contrast to sequential access.

Data stored downwards in cylinders can be retrieved using a parallel technique utilising multiple read and write heads.

Read/write Heads

The data is read from, and written to, a disk sector by use of the read and write heads that are located on an arm. The read and write arm can be fixed or moveable. Fixed read/write arms have

heads for each track and can read many sectors at the one time down the cylinder in a parallel operation. A moveable arm has one set of read/write heads and these move across the surface of the disk.

© IBO 2004 **6.2.2 DISK ACCESS TIME**

Disk access time is made up of two components. The first component is a 'seek time' which occurs whilst the read/write arm seeks the desired track. The second component is a wait or latency time which is incurred as the head write arm waits for the desired sector on the track to spin around.

Access time can be specified as a relationship as follows:

$$\text{Access time} = \text{track seek wait time} + \text{sector wait latency time}$$

Data on disks is accessed in times that are measured in terms of milliseconds. This is much slower than the processing speeds of CPUs. I/O is still slow and, whilst improvements have been made in this area, they do not match the speed improvements of processors.

EXERCISE 6.2

1. Outline how a disk drive operates.
2. Outline the role of the disk drive heads.
3. Define the term 'disk sector'.
4. Define the term 'cylinder'.
5. Define the term 'latency' or 'rotational delay'.
6. Define the term 'seek time'.
7. Define the term 'access time'.
8. How does seek time differ from access time?
9. Describe the access time to a hard disk with reference to the latency and seek times.



© IBO 2004

6.3 OPERATING SYSTEMS AND UTILITIES

© IBO 6.3.1

DEFINITION OF THE TERM OPERATING SYSTEM

An operating system (OS) is required for all general purpose computers to function. The OS is an example of system software. It controls the operation of the computer system by managing the execution of programs, allocating resources, scheduling, controlling input and output operations and management of data.

Operating systems can be classified as:

Single User: Allows the operation of the CPU by one user or task. The traditional DOS environment on a PC was a single user system in that only one user could use the system at any one time.

Multi-user: Allows the operation of the CPU by more than one user or task. The single CPU shares out its time between the demands of the users or tasks running at anyone time.

Multiprocessing: Allows the running of a program across a number of CPUs.

Multitasking: Allows a single user or task to run more than one program at the same time. A single user system can also have this capability.

Multithreading: allows the multiple parts of a single program to run at the same time.

© IBO 2004

6.3.2 FUNCTIONS OF AN OPERATING SYSTEM

An operating system must:

- handle the execution of programs.
- monitor input via input channels such as the keyboard and other input ports.
- control output over the various output channels and ports e.g. disk drives, tape units, printers, modems, display to screen etc.
- manage the file system: create, edit, delete, copy etc.
- manage security measures: password checking and storage, access rights, disk space protection.
- manage memory allocation and swapping to ensure that clashes do not occur and one user's area is protected from being overwritten by another task or user's program.
- manage the operation of virtual memory where a hard disk surface is used as a paging surface to expand available memory.

Utility software: provides a basic function or performs a particular task such as copying data from one location to another.

EXERCISE 6.3

1. Define the term 'operating system'.
2. Outline the key functions of an operating system.



© IBO 2004 **6.3.3 OUTLINE THE FUNCTIONS OF LINKER, LOADER AND LIBRARY MANAGER.**

Linkers: a linker combines compiled object code modules supplied by the programmer with programs from the runtime library of standard function modules e.g. input and output modules supplied by the programming environment to form the executable object code program.

In Java the process is initiated by lines of source code such as import java.io.*.

The linker also replaces symbolic addresses e.g. the variable int x with the real physical memory address.

The linker will also link any operating system modules required to handle such issues as direct calls to the I/O system.

Loaders: loaders copy the linked object code into the main memory so that the linked program can execute. Loaders are examples of an operating system utility program.

Library Managers: a library manager allows central library functions to manage the library functions that are used by the linker. It allows library functions to be added and compiled.

EXERCISE 6.4

1. Define the term 'operating system'.
2. Outline the key functions of an operating system.
3. Outline the functions of a linker.
4. Outline the functions of a loader.
5. Outline the functions of a library manager.



© IBO 2004 **6.4 FURTHER NETWORK FUNDAMENTALS**

© IBO 2004 **6.4.1 THE ROLE OF COMPUTERS IN DIFFERENT NETWORKS**

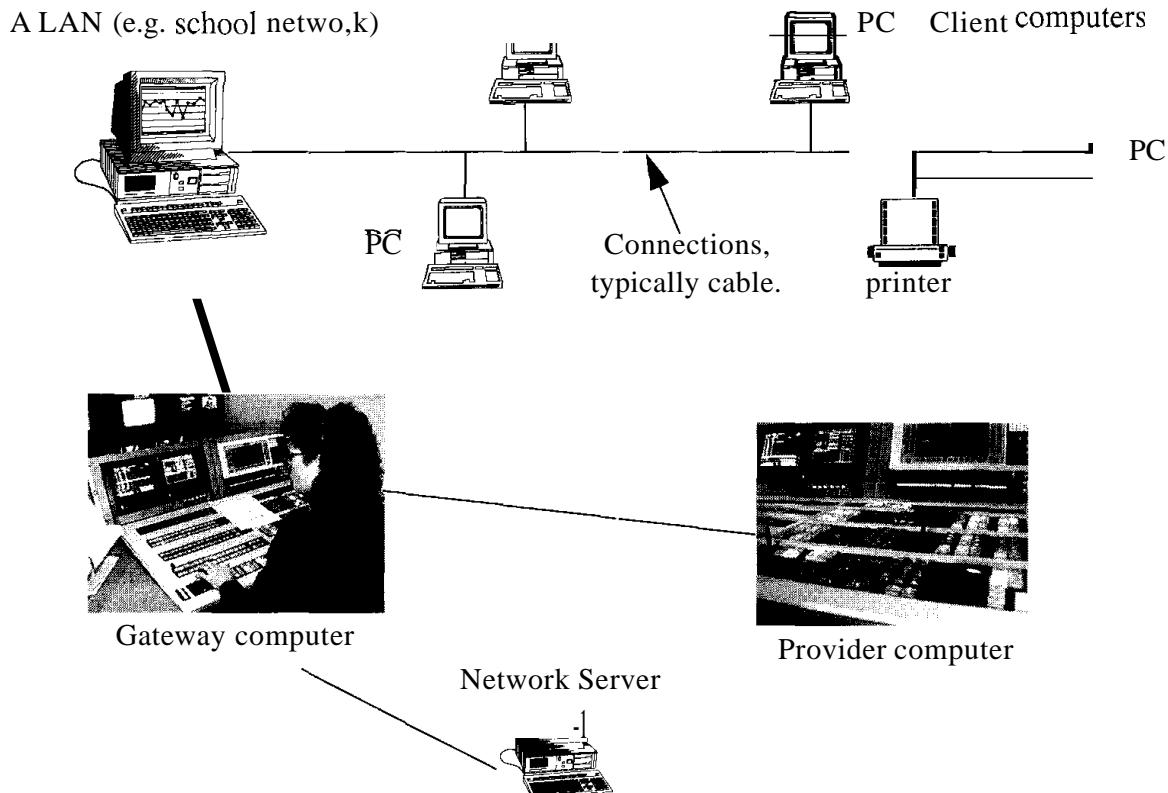
Network Server computers: computers that control the operation of the network. The network server has a network operating system and will enable a range of server based applications e.g. shared databases to be available to users of the network. The network server will control security via a login process, controlling file access levels of users and via interaction with the network firewall or proxy server.

Client computers: clients are computers that are typically operated by users to access the services provided by the network server computers e.g. printing, accessing application servers or access to the ISP and the Internet. Client computers need to have client software installed so that communication can take place between the client computer's operating system and the network server's operating system.

Providers' computers: Providers deliver access to the Internet i.e. Internet Service Providers and to other dedicated databases. Providers' computers are basically network servers that are dedicated to controlling access to the Internet and communications between the user's client computer and the local network server.

Gateway computers: Enable connection from one network to another, e.g. from a LAN to a WAN. Whenever you login to a LAN i.e. connect to the LAN's network server and then access the Internet, you are utilizing a gateway. Gateways are implemented using a combination of software and hardware. Gateways are more than a simple connection. They pass data packets through complex security and filter operations when implemented as a proxy server or firewall.

Figure: 6.5



© IBO 2004 6.4.2 PRINCIPLES OF NETWORKING

The principles of networking cover the following: connection, mode of transmission, network architecture, role of communications software. These principles are now outlined in more detail in the following section.

Remote connection

To utilise a network, a device must be able to physically connect to it. Individual users who are located in their homes and wish to access the Internet or remotely access their work place computer system need to be able to physically connect their computer to 'something' to enable access to the required computer system.

This is typically done via use of a telephone system. Users need to connect a device called a 'modem' to their computers and then connect the modem to the telephone system.

Modems can be internal or external to the user's computer. A modem operates by converting a digital signal from the computer into an analog voice signal that is transmitted over the telephone line. At the receiving end another modem converts the analog signal back to a digital signal. The term modem stands for MODulator-DEModulator.

Further System Fundamentals

Modems can transfer at various rates: 9600, 14400, 28800, 33600, 56800 bits per sec.

Direct connection with a network

Users within an organisation who wish to connect their computers to the organisation's network require the use of a network card and network connection software. The network card enables the physical connection and allows data to be sent and received. The network communications software is called by the user's computer and facilitates the connection to the main network communication software located on the network server. To the user, the accessing of other networks such as the Internet is handled by the network server.

MODE OF TRANSMISSION

Data transmission

Data is transmitted in small packets comprised of individual bits (1 or 0). Hence the rate of transmission is measured in terms of bits per second. The standard telephone line can transmit data at up to 56,800 bits per second and includes the following rates 9600 or 9.6Kbps, 14400 or 14.4Kbps, 28800 or 28.8Kbps or 33300 or 33.3Kbps or 56800 or 56Kbps. Thus, a standard phone line can provide a range of speeds and this range is referred to as the BANDWIDTH of the transmission channel. These transfer rates can be described by the terms 'voice' or 'medium band' or 'bandwidth'.

Broad band is a term used to describe the large transmission rates offered by use of microwave, satellite, cable or fibre optic cables.

Serial and parallel transmission. Serial transmission is the process of transferring one bit at a time and parallel transmission is the process of transmitting more than one bit simultaneously. For example, an 8 bit byte could be transmitted using 8 separate connections. Serial transfer mode is what is used to transmit data using a modem and is used for long distances. The modem's connection cable is plugged into the serial communications port of the PC. Parallel transfer is faster but is used only over short distances e.g. with a CPU to move data from one location to another in memory.

Direction data transmission

There are three modes of data flow used: simplex, half-duplex, full-duplex.

Simplex communication allows transfer of data in only one way. Half-duplex communication allows data to be sent or received i.e. two way but not at the same time.

Full-duplex communications allows data to be sent and received at the same time.

Asynchronous and synchronous transmission

Asynchronous transmission means that data is sent in small amounts at any time. There is no attempt to accept all the transmission in one large amount. Synchronous transmission requires that the data be sent in larger blocks in a timed manner. This allows for faster volume of transfer.

Communications channels

Telephone lines: traditional phone cables use a twisted copper wire. They allow the connection of PCs via modems over the telephone system.

Coaxial cable: this is a thick cable that allows modest transfer rates. It is usually seen in

peer to peer or bus networks.

Twisted pair: very common in LANs; more expensive than coaxial cable but offers higher bandwidth.

Fiber-optic: uses single glass tubes to transmit pulsed light waves to code the digital sequence. Offers high speed and increased capacity. Can transfer at a rate of 26,000 times faster than a standard phone line. They are expensive but are small in physical size and light in weight. They do not generate any electronic signals that can be monitored and hence offer a high degree of security.

Microwave: use line of sight transmission of very high frequency radio waves (microwaves) to send and receive data streams. Offer a convenient way to transmit data without the need to have physical network infra-structure. Offer a variety of transfer rates of up to 2 Mbs per channel.

Satellite: orbit the earth in fixed position. Can be used to relay radio signals.

NETWORK ARCHITECTURE

The components of a network need to be placed into a network design that allows the activities and resources of the network to be shared and accessed by users of the network. Network architecture is the term used to refer to various alternative ways of designing a network.

There are several terms that need to be defined in terms of networks:

Node: a component such as a PC or printer connected to a network.

Client: a client is a user of services on a network. A client makes requests for services, such as a printing request to a printer server.

Server: a server responds to requests from clients for access to a resource. Thus a file server provides access to files, a printer server provides access to printers, a communications server provides access to the Internet and a database server would provide access to a shared database.

NOS: a network operating system (NOS) enables servers and clients to interact on a network and allows users to use the services of a network. Network operating systems are different from normal operating systems that control the operation of a single computer. Novell's netware is a commonly used network operating system used to operate Novell LANs.

FEATURES OF COMMUNICATIONS IN NETWORKS

Ethernet: an architecture used to control data communication between servers and clients within a LAN. Commonly operates at 10 Mbps, but can operate at speeds of 100 Mbps to facilitate high volume applications such as multi-media. The Ethernet standard is typically implemented using a Star network topology, but can be implemented using a Bus topology. Star implementations require that each physical device be connected to the network server via a single cable. Hubs are used to connect many devices to a single high speed connection which in turn can be connected to a further hub and so on until the final connection into the server.

Further System Fundamentals

Public and Private telephone lines: typically used by home users to connect to their Internet Service Provider. Public lines enable inter-user connection via a public telephone network, whereas private lines allow secure dedicated connection between users.

ISDN: Integrated System Digital Network is a standard that allows normal telephone lines to be used to transmit voice, data and video. The transmission speeds are typically 64Kbps. ISDN can also be used with fibre optics to gain significantly high throughput. Requires an ISDN compatible modem.

ADSL: Asymmetric Digital Subscriber Line technology allows very high transmission rates (up to 9 Mbps) to be achieved using the normal copper telephone lines. Requires an ADSL compatible modem.

Fibre optic: Uses very small glass tubes to transmit digital data using light waves. Very high through-put can be achieved and there is far less likelihood of interference, but they are expensive. LANs often use fibre optic cables to connect hubs to servers or to provide a high speed backbone to improve transmission speeds.

Wireless: radio waves to enable communication that does not require a cable. The client PC requires a wireless network card which communicates to the server via communication stations. Significant advances have occurred recently to enable good performance and to enable multi-media to be transmitted effectively.

The implementation of these features will vary from country to country. The reader is urged to relate the above basic theory to way it applies in their case.

Students are not required to know technical detail, but should be able to describe these basic features and be able to justify why one feature maybe more desirable than another.

Some possible points of comparison are listed below:

- fibre optics are normally not appropriate to connect a home computer to a ISP because of cost and it simply may not be available as an option.
ADSL is cost effective and gives very high transfer rates and uses the standard phone lines. Thus it might be a sensible option as compared to using a cable connection.
- ISDN allows the integrated use of the phone system and the transmission of digital data i.e. you can use the phone and connect to the Internet using one existing current phone connection.
- Ethernet is a standard that has stood the test time and therefore one would normally expect an Ethernet network to be the system of choice.
- Wireless offers the possibility of network connection without the need for a physical cable connection. This offers significant advantages in terms of ease of use especially in relation to the connection mobile computing devices such as laptop computers. However, the intended use in terms of high volume multi-media would need to be assessed.

EXERCISE 6.5

1. Outline the role of a network server computer in a network.
2. Outline the role of a gateway in a network.
3. Outline the role of an Internet Service Provider (ISP).

4. By doing some further research, what roles can specific gateways such as firewalls and proxy play in a network?
5. Describe the key features of the Ethernet standard.
6. Describe the key features of ISDN.
7. Describe the key features of ADSL.
8. Describe the key features of fibre optics.
9. Describe the key features of 'wireless'.
10. In what circumstances would ISDN be preferred to ADSL?
11. In what circumstances would ADSL be considered as appropriate for a home users?
12. In what circumstances is fibre optics to be considered as a communications medium?
13. Compare the advantages and disadvantages of wireless as a way to connect mobile computing devices as compared to using physical cables.



© IBO 2004 **6.4.3 PACKET SWITCHING**

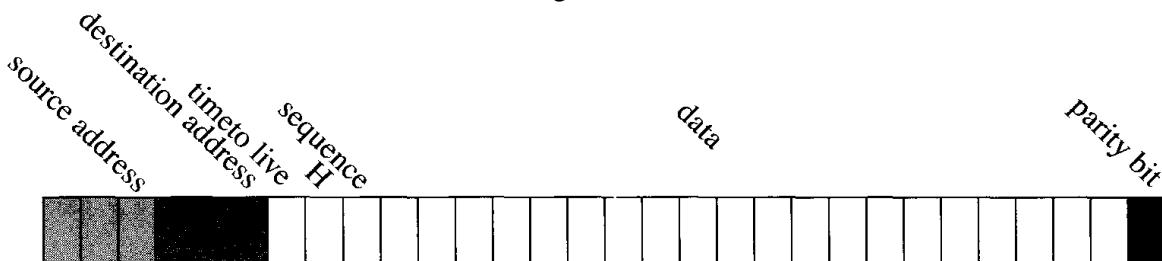
Definition of data packet

Data is transmitted in groups of bits referred to as data packets. A data packet is typically fixed in length and its structure is determined by the protocol being used.

The contents of packet typically holds the source address, destination address, the data and a parity bit or check digit. In a packet switched network, the packet also contains a packet number.

A diagram of a data packet is shown below.

Figure: 6.6



Definition of Packet Switching

Packet switching sends data in individual groups called data packets. A single data transmission is broken up into individual packets and then each packet is transmitted. The key feature is that the individual packets need not be transmitted over the same links, but can be sent over the best available link. The original data transmission that was broken up into packets is then reassembled at the destination.

The re-ordering is possible because each packet has a separate packet number. Thus the packets can be re-ordered according to this packet number.

Further System Fundamentals

Each packet also has a counter which decrements as it passes through a node. If it reaches zero, the packet is discarded.

The main advantage is that if a link is broken or unavailable during transmission another link can be used and the transmission can continue. Packet switching is suitable when small delays can be tolerated e.g. email transmission as compared to real-time video where it is important that there is no delay in either the sound or pictures. Packet switching is also suitable where data transmission is not a constant stream or rather where there is a burst of data transmission and then some idle time before another burst. During the idle time the network link can be used to handle another transfer.

This method is in contrast to circuit switching which uses a fixed link to send the data transmission. The advantage is that it can be fast, but subject to failure if a link is broken during transmission.

ROLE OF NODES IN TRAFFIC MANAGEMENT

As a packet traverses the links, each link is connected via a node and the data traffic is managed via a router. The router inspects each packet and routes it on to the appropriate link. Network traffic management involves ensuring that the packets are passed on to an appropriate link and that congestion is managed so that network performance is maintained.

There are two factors to be considered by the routing algorithms: determining the shortest path and determining free nodes. Both involve complex mathematical algorithms.

ROUTING OF PACKETS OVER DIFFERENT PATHS

As mentioned above the routing of packets is a fundamental feature of a packet switching network. A router will inspect the destination address and then inspect the available links. A traffic management algorithm will be employed to decide which link is the most appropriate to use. Remember, in a high speed network, a large amount of traffic is entering a single node at any one time. The router needs to be able to handle each individual packet and ensure that it is sent to the correct destination, but the route that packets take can vary.

PACKET ERRORS

Packet errors can occur as a result of 'collisions' or by 'interference'. In collisions, the same link is attempted to be used by two packets at the same time. In this case, the router will communicate with the sender and the packet will be resent.

On arrival, a data packet is checked to see if there have been any transmission errors e.g. parity is checked. If an error is found, the destination router will request a resend.

Obviously the higher the error rate the slower the performance of the transmission.

© IBO 2004 6.4.4 PACKET SWITCHING PROTOCOL

Protocols

Packet switching protocols establish the rules for node to node communication and application to application communications. The Internet has established *TCP/IP* as the default standard protocol. This protocol is a combination of two separate protocols.

The transaction control protocol (TCP) controls the direct links between computer applications and ensures that the data arrives and is assembled into the correct order. It is a transport level protocol of the OSI model.

The Internet protocol (IP) enables the identification of packets so that the destination or source address can be determined. It is a packet switching protocol and handles the dividing of the data into small packets. It is a network-layer OSI protocol.

World Wide Web addresses e.g. www.myWeb.com represents an **IP** addresses of the location of a computer or hypertext page or resource on the Internet. Addresses of computers on the web are known as DOMAIN names.

The **IP** protocol uses four sets of 8 bit bytes to denote the address. These bytes are used to address the network and computer on the network.

ROLE OF PROTOCOLS IN PACKET SWITCHED NETWORKS

Protocols play an important role in packet switching networks. In very simple terms, the data must be split up into packets and then the packets reassembled into the correct order at the receiving end. This is a very complex process. Rules or protocols are required to govern each stage of this process to ensure that the data is transmitted and is error free.

© IBO 2004 6.4.5 NETWORK SECURITY AND HOW IT IS ACHIEVED

Network security is concerned with preventing unauthorised or illegal access to data that is stored on the network.

There are essentially four main ways that security can be implemented.

NETWORK LOGIN ACCESS CONTROL

Networked computer system applications that go 'live to the world' via the Internet, for instance, can be accessed remotely via a WAN connection. A user is prompted for a user login and then a password. Local area network applications are also accessed via the same kind of process.

The operating system authenticates the user by checking a valid user list and the user is then free to execute commands.

The user list will also allow the system administrator to assign access rights. The top (most powerful) access level is that of supervisor or administrator. With these rights, a user can run any command and access any area of the network. Normal user rights can be set to various levels of access depending on the need for the user to be able to access, view or edit and delete data.

Most networks provide secure private home drive areas where the user has full rights, but users' rights will be restricted to other areas. For example, a user maybe able to view a file of customers but not be able to edit or delete records from the file.

Sensitive data can be placed in folders/directories that only certain users have the rights to access.

The essential point is that the computer cannot distinguish between valid and invalid users if the password is either guessed, left blank or discovered by other means.

In recent times, a range of biometric devices have become available to replace or augment the traditional login. Some of these devices are listed below. Each works by taking a representation of some biological feature, e.g. finger print, and storing this away as a 'biometric measurement'. When the user presents again the biometric characteristic is re-measured and compared to the stored version to achieve authentication.

Some biometric devices include: finger print recognition, eye scanning, face measurement etc.

CONTROL OF ACCESS VIA PERMISSIONS (LAYERED ACCESS)

Layered access via permissions is a more formal application of the access control discussed above. The idea is that you need to gain increasing levels of permission to gain access to high levels of sensitive data or commands.

ROLE OF A FIREWALL

A firewall is a separate component of a network, both hardware and software, that sits between a private network and the outside world. The firewall inspects data packets to determine destination and/or the source of the packet. Thus known user access to destinations that users are not allowed to access can be enforced and known malicious external sources can be denied access.

<http://www.interhack.net/pubs/fwfaq/#SECTION00030000000000000000>

<http://computer.howstuffworks.com/firewall.btm>

DATA ENCRYPTION

Data encryption is the process of encoding data so that it can't be sensibly read without the use of a key that will enable decoding. Data that is stored can be encrypted so that if it is stolen it can't be read. Data that is transmitted can also be encrypted to prevent effective interception.

© IBO 2004 6.5 COMPUTER PERIPHERAL COMMUNICATION

© IBO 2004 6.5.1 DEFINE PORT AND HANDSHAKING

Port

A port enables the interfacing of a device with a computer. It enables data to enter and exit between connected units. Within a personal computer there are ports that allow the connecting of disk drives, display units and the keyboard. You can also connect peripheral devices to a range of external ports such as a modem to the 9 pin serial port or a printer to the parallel port.

Serial ports allow data to be transmitted in the bit stream. The connection can be over a reasonable distance, but is slower than a parallel connection. Data transmitted over the Internet from a home PC is sent and received using a serial port connected via a cable to a modem. The modem typically operates at 56 kbs. More expensive cable modems are available that allow 1 Mbs transfer.

Parallel ports allow streams of bytes of data to be transmitted. Thus the 8 bits of the byte are transmitted at the same time. This improves the rate of transfer but there are restrictions on the distance over which parallel transfer can take place.

The term 'port' is also used to refer to the way that connections are logically made over the Internet between a user's computer and the destination computer they are connecting to, using the TCP/IP protocol. For example port 80 is used for http traffic.

Handshaking

The IE glossary definition 'The exchange of predetermined signals when a connection is

established between two modems' (IE Subject Guide, 2004, Glossary: © The British Computer Society, 2002).

'Handshaking' is a process by which two devices initiate communication over a communications channel. A channel refers to the medium that is used for the communication. For example, an I/O port connected to an I/O port on another computer via a cable or infra-red beam forms a communication channel.

The devices then send messages back and forth to establish that the communications protocol to be used is understood by both devices.

©IBO 6 5.2 DEFINE DIRECT MEMORY ACCESS (DMA) AND BUFFER

Buffers

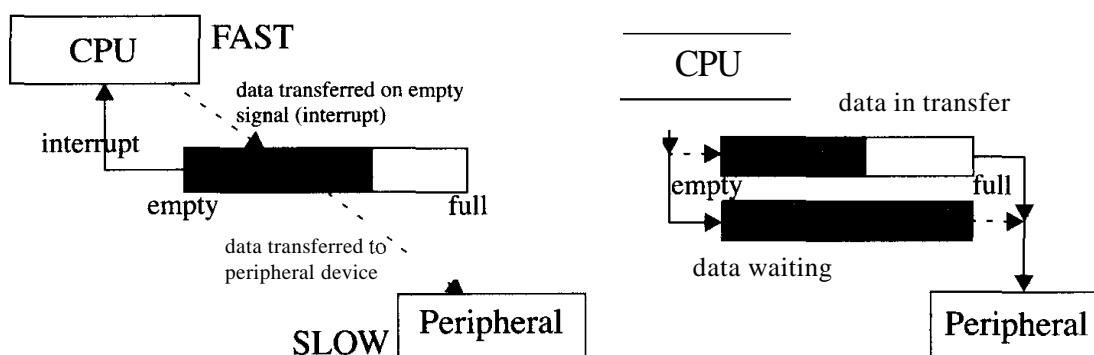
Buffers allow the temporary storage of data. The CPU uses buffers to store and manipulate data before transmitting it to a peripheral device. Buffers are located in RAM and thus occupy free memory space. They are used to coordinate the faster pace of the CPU with respect to the slower operation of peripheral devices such as printers and hard disks.

The print operation uses a buffer approach by firstly copying the file to be printed into RAM and then sending the data to the printer as required, thus freeing the CPU to get on with other things.

Read and write operations to a hard disk often use buffers. The data is written to the buffer area and this is cleared every so often. This design feature means that the writing is done in a time efficient manner. Writing does not happen each time a disk write is indicated but at set points when the buffer is 'flushed' (its contents are written to the hard disk).

Keyboard strokes entered at the command line are stored in a buffer prior to the ENTER key being pressed. The buffer is used to display the keystrokes on the screen but no action is taken on the contents of the buffer until ENTER triggers an interrupt that then runs the command interpreter to see if the contents of the keyboard buffer form a command known to the operating system. With a single buffer an interrupt is sent to the processor when it is empty. The CPU then re-fills the buffer. However sometimes, if the processor is busy with more important tasks, the buffer might not be filled straight away. Therefore two buffers can be used. When one buffer is empty the interrupt is sent but the data is then read from the second buffer and there is no delay.

Figure: 6.7



DMA

Direct memory access is a design feature of the computer architecture that allows peripheral devices such as disk drives to access the computer main memory without the CPU being involved. The feature allows for faster access and can be used to perform backups as a background process.

If the CPU was directly involved in the process, the data would have to be read into the CPU and then transferred out again to the destination peripheral. Using this design, large amounts of data can be read and written to a hard disk without the CPU needing to waste time reading the data in the normal read and write cycle.

© IBO 2004 **6.5.3 DEFINE INTERRUPT AND POLLING**

Interrupt

Recall that a standard CPU processes a single instruction at a time. But we can have systems with many different users (multi-user) as well as *systems* performing many different tasks apparently at the same time (multi-tasking) - some of these tasks are carried out without the user necessarily being aware of them.

It would be very inefficient if all requests to the computer for actions were simply placed in a queue and worked through in the time order in which they occurred. The computer works at a very high speed compared to a human. A second may seem a very short time to a human but, to a computer that is able to perform millions of instructions per second, one second is a very long time.

This means that the computer can share out its time across a range of conflicting demands. It does this by allocating small slices of its available time to the things demanding its attention. Some things require a longer time than others and some are more important than others.

Next time you sit in class and the teacher is issuing some instructions, watch what happens if someone asks a question. The teacher may say "please don't interrupt me" or s/he may stop and handle the question i.e. allow the interruption. When *s/he* has answered the question s/he will continue on where *s/he* has left off.

The computer system operates the same way by the operation of an INTERRUPT mechanism.

What things (EVENTS) can cause the computer to suspend what it is doing and HANDLE the interruption?

A printer malfunction may trigger a hardware interrupt that says to the computer the printer is not available and this fact needs to be communicated to the user who is currently trying to print.

An ENTER (carriage return) keypress means that the computer has to suspend what it is doing and handle the command that has been entered.

A program may fail and trigger an exception error that needs to be handled.

The process of the computer detecting an event it needs to respond to is called 'handling an interrupt'. Interrupts can come from signals sent by hardware and are called 'hardware interrupts'. They can also come from software and these are called 'software interrupts'.

How does the interrupt mechanism work?

The basic mechanism is that a signal is received by the computer indicating that an event has

occurred to which the computer needs to respond. The interrupt event handler is then activated. The computer must suspend what it is doing and then service the nature of the interrupt. When the interrupt has been handled, the computer resumes operation.

Polling

'Polling' is the process of requesting some service from another device. Polling is used by the CPU to periodically check certain registers or sensors to see if some request has been made. The computer then needs to service the request. For example a set of data loggers could be connected to a central computer to monitor air pollution levels in a city. To read the data stored in the data logger, the data logger needs to transmit the data to the central computer. This could be done by getting the computer to periodically poll the data logger to indicate that it is ready to receive the data. Such a process would be very efficient because each device could be read in turn in an orderly manner.

The act of polling is also used to determine the readiness of a device to receive data. For example, a modem can be used to poll the other destination line to see if it is available or a printer can be asked if it is ready to receive.

Summary of interrupt operation:

- Interrupt is sent by a hardware or software process.
- The processor determines what to do.
- The processor responds to current tasks and places current task on stack.
- The processor attends to processing associated with the interrupt.
- The processor pops off stack and resumes.

©IB06 5.4 EXPLAIN HOW PERIPHERAL DEVICES ARE CONTROLLED WITH REFERENCE TO PRINTER, MODEM AND DISK DRIVE

Printers

Printers are controlled by the use of polling and interrupts. When a printer is required, the computer polls the printer to determine its readiness. If it is ready, a hand shaking operation takes place to ensure that the computer has the necessary printer driver and then the data is copied to a buffer area and sent to the printer. When the printer has finished, it indicates this to the computer and the next printer job is sent. A printer may also use a buffer to store the entire document before it is printed.

Modem

To initiate a connection between two modem devices, a hand shaking routine is undertaken that establishes that the communication channel is open and that the data transfer protocols are compatible. Normally the sending modem tries to send at the fastest rate and works backwards until a match is found. If no connection is possible, the modem signals back to the computer via an interrupt that the channel is not available. If the data transfer is successful, the modem indicates that it is free to be used.

Disk drive

Disk drives are controlled via the disk drive controller. The computer indicates that it wishes to write data to, or read data from, the disk. The controller is detected and the data transferred from

Further System Fundamentals

the internal buffer to the controller that coordinates the write operation or the transfer to the internal buffer area of the computer.

©IBO 6.5.5 COMPARE THE FEATURES OF DMA, INTERRUPT SYSTEMS AND POLLING SYSTEMS

DMA does not require the intervention of the CPU and thus large amounts of data can be moved in and out of the computer without the need to use the CPU. This speeds up the operation of data transfer. For example, backups can be performed as a background process.

Interrupt systems allow the CPU to work without needing to actively monitor devices. As a device needs the CPU, or wishes to communicate, it activates an interrupt. The CPU is then made to respond, but it has not wasted time checking if it needs to respond.

Polling systems require the CPU to actively monitor if it needs to do something in relation to a particular device. The advantage of polling is that it can be done to suit the processing profile of the CPU i.e. poll during low usage times.

Polling is to be preferred when it is better for the CPU to determine when to deal with a device. Usually this means that the device can wait and it is not critical that it be serviced immediately. Interrupts are necessary where the event must be serviced when it occurs. For example, a modem not responding needs to be detected as it happens, not when the CPU gets around to polling the line to see if transfer was occurring.

DMA is to be preferred when large amounts of data need to be moved that otherwise would take a lot of CPU time which could better be used in serving online requests.

©IBO 6.5.6 COMPARE THE FEATURES OF SERIAL AND PARALLEL INTERFACES

Serial interface features

A serial interface allows data to be transferred one bit at a time. A serial port is a general purpose interface and is used by mice, keyboards and modems. The common standards controlling the way serial interfaces operate are the RS-232C and RS-422 standards. The normal PC has both a 9 pin communication port and a 25 pin D-type port, but not all these pins are used. Serial communication is possible with only 3 wires: one as the ground signal, one to send and one to receive.

Data can be sent in either a synchronous or an asynchronous mode and can be sent in both directions at the one time.

Error detection is possible using an odd or even parity bit.

The length of cable varies depending on the quality of cable used. For example, using shielded cable and transmitting at 9600 bps, cables can reach 250 feet. If using unshielded cable, the length is restricted to about 100 feet.

Parallel interface features

Parallel interfaces allow much faster rates of transfer because 8 bits are transferred at one time. They are commonly used to interface a printer to a PC by direct connection using a 25 pin centronix standard interface. Parallel ports are also used to connect other peripheral devices that require faster data transfer than that offered by serial ports.

Traditionally the standard parallel port on a PC did not allow bi-directional operation and the length of cable was limited to very short distances. In 1994 a new standard was developed called the IEEE 1284 standard that allowed for bi-directional communication over the parallel port, but not full-duplex. The interface used 17 signal and 8 ground lines. The signal lines are divided into three groups: 4 control lines, 5 status lines and 8 data lines. The control lines are used to initiate handshaking with peripheral devices and the status lines are used to communicate the state of peripherals e.g. indicating errors, busy status or paper status reports. These new standards increased data transfer rates from around 150 kilobites per second to over 1 mega bytes per second.

In summary, the serial interface is slower but is simpler to implement and the cable can reach a reasonable distance. The serial interface allows full-duplex operation for use with, for example, a modem. The parallel interface allows for much greater throughput, but the connection is more complex and the cable run is shorter. Bi-directional communication is possible, but not full-duplex communication.

EXERCISE 6.6

1. Define the term 'port'.
2. Define the term 'handshaking'.
3. Define the term 'buffer'.
4. Define the term 'interrupt'.
5. Explain why buffers are normally used to aid in the transfer of data from a peripheral device to a computer.
6. Define the term 'controller'.
7. Explain, with reference to appropriate terms from the above questions, the general operation of a printer.
8. Explain, with reference to appropriate terms from the above questions, the general operation of a modem.
9. Explain, with reference to appropriate terms from the above questions, the general operation of a disk drive.
10. In general terms, compare the features of DMA, interrupts and polling when connecting an external device to a computer.
11. An external device is being designed to be connected to a computer. The device is a warning mechanism for an industrial process. When the device signals that the process has become unstable, the computer is to initiate the shutting down procedures to halt the process. For the above device, compare the pros and cons of using an interrupt or polling system.
12. A high speed input device is to connect with a computer. It will transmit data at the rate of 100,000 bits per second and will do this twice per day for a period of about 30 minutes. You are asked to compare the pros and cons of using either a serial or parallel interface. The device will be situated some 20 metres from the computer.

Further System Fundamentals

- 13.** A high speed disk drive storage device is required to connect with a computer. A temperature gauge is to be connected to the computer and the data stored on the high speed drive. The temperature gauge will make readings every 100th of a second and transfer the data to the computer every 5 minutes. Suggest a method of transferring the data from the computer to the high speed disk drive so as to minimise the impact on the normal functioning of the computer.

QUESTION PAPER

,I.

ANSWER

7

Chapter contents

Introduction

7.1 File Organisaton

INTRODUCTION

Recall that files store data permanently on a secondary storage medium or backing store medium such as disk or tape. The contents of a file can be organised sequentially or randomly i.e. in any order. The term 'serial file' is used to denote an unordered sequential file. A hard disk provides the ability to access data in files either directly or sequentially, whereas magnetic tape only allows sequential access.

A file stores data related to a particular entity such as a STUDENT. The data for a particular student is stored as a record of the file, hence a file is a collection of records. Each record is a collection of the attributes about the particular occurrence of the entity about which data is being stored. A record has a defined structure and allows data of different data types to be stored about the single occurrence of the entity. The different data in a record are referred to by their field names.

A record is thus a collection of data fields related to attributes of the occurrence of the entity being stored.

For example, a school may wish to collect and store data about the students enrolled at the school.

The file is thus a collection of the entity STUDENT.

The record of the file could collect the following attributes of a student:

ID as an integer.

First Name as a String limited to 20 characters.

Second Name as a String limited to 20 characters.

Year Level as an integer.

Each attribute would be described by a filed name, which in tum has a data type.

In this case the field names could be: **ID**, firstname, surname, year. The data types for each field name are shown above: integer, string, string and integer.

An organisation could also collect together files to form a database. Thus we have a simple hierarchy:

Database, is a collection of related

File(s), is a collection of related

Record(s), is a collection of related fields

Fields have a name and data type and are a collection of bytes.

A record structure is determined by the nature of the data required to be stored.

A record length is the total of the bytes required to store the data for each field.

The size of the file is given by the number of records multiplied by the length of a record.

The record structure of the file defines the characteristics of the occurrence of the entity stored.

The term 'entity' refers to the noun used to describe the collective term for the file e.g. file of STUDENTS, file of CARS, file of TREES. The idea of an entity is related to the concept of the term 'object'. When we add a record to the file we are adding a new occurrence of the entity i.e. a new student is added to the STUDENT file. The new student is a new occurrence of the entity student and is represented by filling out the fields of the record with the data that relates to the new student.

Data in records needs to be written to files for permanent storage and read from files into RAM to allow the data to be used in processing.

The organisation of the data in the records in the file is important in terms of what data is stored for each record and the associated data type. It is also important in terms of how the data is stored and how the data is accessed.

The term 'data storage' here means 'how the data is organised on the storage medium used'. Data in files can be organised in a variety of ways on storage media. The characteristics of the storage medium used also impacts on the choice of organisation.

The term 'data access' means the way in which data is accessed on the secondary storage medium that results in the data being transferred into the random access memory. Data can be accessed in two main ways. Either you access the data with reference to the position of other data in the file and must work sequentially through the data to find what you require. Or, because of the characteristics of hard disks, you can access data directly without reference to the other data in the file. You will recall that the read and write arm of a hard disk can be directed to a specific track and sector, thus data can be accessed without the need to sequentially work through other data in the file. For this reason we can scatter data in a disk file all over the place in a random way and still be able to access each record by knowing its track and sector number.

In all of the file organisational techniques we will look at, the idea of fixed length fields and records is implied. Whichever file organisation is used, the bytes of the file are stored in a long sequence. These sequences can be stored over a number tracks and sectors or clusters on a disk or as a long byte sequence on tape. The tracks need not be contiguous i.e. next to one another. Also, a logical record as described by the programmer will usually fit into the cluster size of the disk as used by the operating system i.e. 512 bytes or some multiple of 512. This is known as the 'physical record' and it is the amount of data that is accessed in one physical read/write operation. A logical record may take 50 bytes and a cluster may store 10 of these logical records in one physical record.

Records in a physical record can only be sequentially accessed once the cluster has been accessed. The physical record is read into memory and then the record located by a sequential search.

If the fields are fixed in length and the number of fields per record is fixed, it is easy to count the bytes that belong to each record. This can be done sequentially by, for example, starting at the beginning and counting off the number of bytes for each record.

EXERCISE 7.1

1. Define the term 'file'.
2. Define the term 'record'.
3. Define the term 'field within a record'.
4. Why do fields have a datatype?

System Life Cycle

5. Construct a record structure for the following file and state the data type for each field. A teacher wishes to record details about the address, phone number, contacts and email address of each student in one of their classes.
6. Assume an integer takes 4 bytes, a real 8 bytes and a char 1 byte, size the following record structure and if there are 1,000,000 records, size the file.

Fields are A(integer) + B(char) + C(real) + D(string of 10 char)

Record Length = length of all fields in record

File Size = Record Length × Number of Records

7. Define the term 'database'.



© IBO 2004 **7.1 FILE ORGANISATION**

© IBO 2004 **7.1.1 DEFINITION OF THE TERM KEY FIELD**

A 'key field' is used to identify a record.

A primary key is used to uniquely identify a record e.g. a student ID number is a primary key. A unique key can also be created by combining two or more fields to uniquely identify the record. For example, employee surname + date of birth + phone number fields can be combined to form a unique key.

Secondary keys can be used to classify records, e.g. sex type could be a secondary key.

© IBO 2004 **7.1.2-3 SEQUENTIAL FILE ORGANISATION**

The organisation of a sequential file is the same as that of a serial file except that the records are stored sequentially in a specific order, for example, in the numeric order of ID or alphabetically according a surname field.

A standard sequential file algorithm has the following format:

```
Open file for reading
Found = false
While (not end of file and not found) Do
    Read record
    Set fields to data variables: data1=field1, data2=field2 etc
    If record matches required record then found = true
    If (found) then perform desired processing on data variables
Close file
```

Sequential and serial files were originally designed to operate on magnetic tapes. For example, if you have read to the 10th record and wish to access a previous record you must start reading from the beginning of the file.

Sequential files cannot be updated directly. They must be read into the primary memory (RAM) and sorted by using arrays and then written back out to the disk or tape.

EXERCISE 7.2

1. What is the difference between serial and sequentially organised file structures?
2. What is the difference between the organisation of a file and the method used to access records in the file?
3. Write an algorithm to append a new record to a sequential file. Assume that a command Append(filename) record exists that writes to the end of the file.
4. Assume a file record structure stores a person's ID, a their surname and their age. Write an algorithm to read and display each record where the age is over a fixed value.
5. Using the file structure from above, write an algorithm to edit a specific record in the file. Assume that the person's **ID** is used to access the record.
6. In order to convert a serial file into a sequential file, what would need to be done?
7. Write an algorithm to read a serial file of 1000 records and then write a new sequential file. Assume that the records in the file are single and unique integer values i.e. the record contains one field that is of integer type.



©IBO 7.1.4 PARTIALLY-INDEXED SEQUENTIAL FILE ORGANISATION

An 'index' to a file operates the same way that an index works in the front or back of a book.

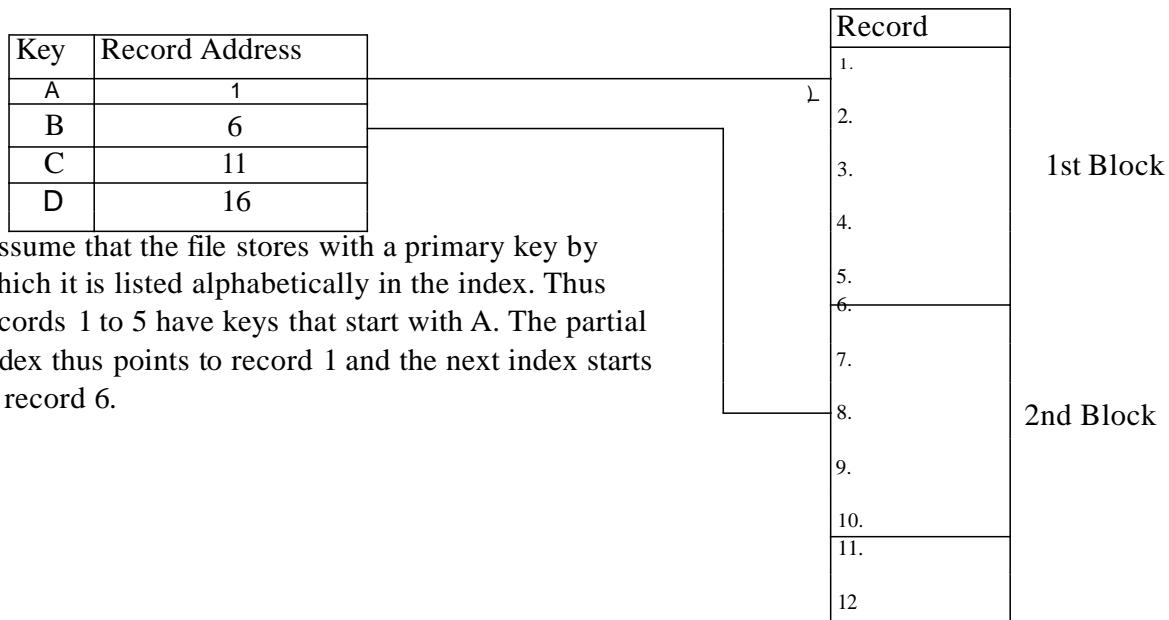
In a book chapters begin on designated page numbers. You could find them by looking through the book page by page in a sequential way. Thus operating the book as if it were a sequential file. Alternatively you could use the index that provides the chapter headings and the corresponding page numbers. Unfortunately you would still need to resort to some sequential searching. Firstly, to look up the index page and then to actually find the page, but at least you would have some idea where to begin looking.

Many web sites use an index approach. Consider a set of web pages for a glossary. The first page could be an index, with a link for each of letters A, B, C etc., that would point to the top of the corresponding page. On the page pointed to by A would be an alphabetic list of all the words starting with the letter A. Such a setup is operating a 'partial index'.

The initial look up of the main index is via sequential look up. The access to the desired page is via a direct link and then a sequential search is made of the linked page to find the desired term (record) in the page.

The diagram below shows how such a system could operate in a computer system. An important point to note is that the index itself is a file.

The basic idea is to have an index that points to the first record in a group of records. This index is sequentially searched and then the group that contains the desired record is accessed directly. The group is then sequentially searched to locate the required record. Thus the file is split or organised into separate groups. The groupings are usually based on some logic such as alphabetic order.

Figure: 7.1

The basic algorithm for a partially indexed file is shown below.

```

Open index
Open file
Get searchKey
Found = false
Locate access address in index using searchkey
Direct access record group
While not at end of group and not found Do
    Get record
    if match then
        found = true
        return record
    if found then process record as required
close file and index

```

In summary:

- The index and records in each group are ordered.
- The index is searched sequentially.
- The first record of each group is accessed directly.
- The other records in each group are searched sequentially.

© IBO 2004 **7.1.5 FULLY INDEXED FILE**

A fully indexed file has an address for each record in the file associated with each separate index. The index is firstly searched sequentially to find the address. This address is then used to access the desired record in the file using direct access. Importantly, the records on the disk do not need to be in a specified order as the index provides the appropriate access.

The diagram below shows how this operates.

Figure: 7.2

| Key | Record Number |
|-----|---------------|
| a | 4 |
| b | 7 |
| c | 5 |
| d | 3 |
| e | 12 |
| m | 9 |
| n | 10 |
| p | 2 |
| s | 11 |
| t | 6 |
| z | 1 |

| Record Number | Key | Data fields |
|---------------|-----|-------------|
| 1 | 4 | |
| 2 | 7 | |
| 3 | 5 | |
| 4 | 3 | |
| 5 | 12 | |
| 6 | 9 | |
| 7 | 10 | |
| 8 | 2 | |
| 9 | 11 | |
| 10 | 6 | |
| 11 | 1 | |

Assume that the key is a single letter so that there are 26 possible records. The index allows access in sorted order, one entry per record. Records on file are in serial order.

Files can be indexed in a variety of ways and thus multiple indexes are common.

The advantage is that the file can be accessed directly but can also be accessed in sequential order by reading down the index file and using the address of each index to access the file in the desired order.

In all cases there is always an overhead associated with indexed files. The index file itself takes up disk space, and thus memory, when it is used. Also, to maintain the index in sorted order requires constant re-ordering and hence takes processing time.

EXERCISE 7.3

1. Define what is meant by the term 'partially indexed file' and explain how records are accessed using this method.
2. Define what is meant by the term 'fully indexed file' and explain how records are accessed using this method.
3. What are the advantages and disadvantages of using either a partially or fully indexed file structure?
4. In a customer management system, assume that a sequential file was ordered on the customers' surnames. Use diagrams to explain how the records could be accessed firstly by a fully indexed method and then by a partially indexed method.



©IB07.1.6 DIRECT FILE ORGANISATION

2004

As mentioned above, all files are simply a sequence of bytes stored one after the other using fixed length fields and records. To implement sequential access is very simple. You open the file and then read the data back. The important point to note is that different data types are stored differently. A string is a sequence of 8 bit characters, an integer is usually 4 bytes or 32 bits and a real can be up to 64 bits or 8 bytes. Thus to read data back it must be read back in the same order in which it was written and the fields assigned to appropriately typed data variables.

If you were able to keep track of the number of records written, you would be able to point to the start of the n^{th} record in the byte sequence by using the offset formula:

$$\text{Start of record offset byte} = \text{number of records} \times \text{length of record}$$

As usual, the first record is the 0th. Thus, if we had a record size of 50 bytes, the 10th record would start at an offset position calculated as $50 \times 10 = 500^{\text{th}}$ byte position and go for 50 bytes. Thus to get access to record in the 50th position we need to provide the computer this offset position. The implementation of this is different from one programming language to another and between operating systems.

The organisation of the records can be in any order. To access the desired record, we need to know its position in the byte sequence that makes up the file. To facilitate this type of access we need the operating system to cooperate. Fortunately the designers of computer systems know this. The operating system keeps track of the files in its system by use of a file management system. The file management system keeps track of the start of files on the disk. If the program can provide the name of the file and the byte offset position, the file management system can use this information to locate the required start point on the disk by positioning the read head over the required track and sector.

The reading and data transfer then takes place and this is the same for either form of access. In fact the physical record is read. The data is then assigned to the various variables specified in the reading part of the program. At this point it is important that the data types match.

The key question is: how does a user remember the position of the data in the file? One way is to use some form of 'key hashing'. The user simply enters the desired key sequence, e.g. surname or product code, and the computer program converts this into a record number that is then used to provide direct access.

Identification numbers are commonly used because they provide a unique number. An ID is usually allocated when a new record is added to a file. Thus, if the program keeps track of the number of records already used, the next ID generated can be linked in some way to the ID. For example an ID could be generated that used the year and record number and check digit. An example is shown below:

```
NextAvailable record location    100
Year = 2000
Check digit = 2000 mod 100 = 0
```

Thus the new ID is 20001000. This ID would be recorded for future use.

When this ID is entered, the check digit is calculated and then string functions are used to extract the desired record off-set position.

Standard File Direct Access Algorithm

The following algorithm demonstrates the basic logic of accessing a direct access file. It assumes that record I is specified by recordNumber = 0.

```

Start
  set lengthOfRecord    length of record in bytes
  open file
  get recordNumber
  moveto byte position recordNumber*lengthOfRecord
  read record
  edit record
  write record
  close file
End

```

EXERCISE 7.4

1. Explain what is meant by the term 'direct access file organisation'.
2. Why can this method of access not be efficiently done using magnetic tape as the secondary storage medium?
3. Why can direct access be achieved using magnetic or optical disk as the secondary storage medium?
4. Explain the operation of the Java seek method in randomAccessFile class.
5. Using your chosen programming language, look up the equivalent command to 'moveto'.
6. When using direct access, explain why it is not necessary to re-write the entire file when a single record is updated?
7. When accessing a direct access file why do we use the PHYSICAL RECORD position of the record in the file?
8. Implement in your chosen language a simple direct access system for the following problem. Assume a record contains a person's surname and age. Your program needs to be able to access a specified record position, read the record, edit the age field and then write the new record.
9. Records in a direct access file are normally not ordered as in a sequential file. If a direct access file contains 100 records, the records can be accessed sequentially by, for example, using the 'moveto' command to move incrementally through the file. Suggest two ways to overcome the problem of how to get access to the data records in some form of order.
10. A bank wishes to allow customers access to their bank accounts via an automatic teller machine. Justify your choice of file organisation and access method.
11. A satellite beams back temperature readings at the rate of 2 readings per hour. Justify your choice of file organisation and access method.



©IB07.1.7 FIXED AND VARIABLE LENGTH RECORDS

2004

The length of a record is determined by calculating the TOTAL number of bytes in the record.

A record is made up of fields and each field has a data type, which in turn is fixed in length for primitive types such as int or double. In Java an int field takes up 32 bits or 4 bytes.

However, data types such String are reference types or Objects. Unless you ensure that they are fixed in length the length can vary. Example: a String surname field can obviously have varying lengths as the length of surname varies.

Consider a record defined by the following class:

```
class Record

    int age;
    String name;
    double weight;
```

The length of a general record = 32 bits + variable number of Bits + 64 bits.

The specific record 23, Smith, 78.9 has length $32 + 40 + 64 = 136$ bits.

(note Smith = lbyte + lbyte + lbyte + lbyte + lbyte i.e. 1 byte (8 bits) per character, which gives 5 bytes = 5 bytes * 8 bits per byte = 40 bits)

The specific record 4, Ng, 10.3 has length $32 + 16 + 64 = 112$ bits.

Whilst the general record structure is the same the length of the record is different depending on the number bits in the name field.

A sequential file can easily have variable length records. Each field is written and read separately or the entire record concatenated and read and written in one operation.

The sample Java program below would happily read a test.dat file of the following structure:

```
23, Smith, 78.9
4, Ng, 10.3
```

```
import java.io.*;
public class seqDemol
{
    public static void main(String args[] )
    {
        new public seqDemol();
    }
    public seqDemol()
    {
        try
        {
            BufferedReader inFile=new BufferedReader(new FileReader("test.dat"));
            if (inFile.ready()) output("file ready for reading");
            PrintWriter outFile=new PrintWriter(new FileWriter("copyTest.data"));
        }
    }
}
```

```

if (!outFile.checkError())output("file ready for writing");

String inRecord = null;

while (inFile.ready())
{
    inRecord = inFile.readLine();
    output(inRecord);

    input.close();
    output.close();
}

catch(Exception e)
{
    output ("Something wrong");
}

```

Each record is variable in length but the record is read and output to the screen.

If we were to implement a direct access system, we need to use fixed length records. The reason is that the start byte position of the desired record is calculated by multiplying the record length by the relative record position.

Hence, to access record 4 we say seek(fixedRecordLength*record-1). We subtract 1 from the record number because we start counting at 0!

If the records could be variable in length the seek operation would fail.

The general way of ensuring records are of fixed length is to force each String field to take up a fixed number of bytes. Extra spaces are added to the front of the field to pad out the length if needed.

Another way is to use a fixed byte length into which the fields are written and, provided the record does not overflow this fixed length, the seek operation will work. The problem with this method is that it potentially wastes space on the disk.

©IB07 1.8 THE USE OF HASHING TO FACILITATE FILE ACCESS

The notion of 'hashing' has already been covered. One of its uses is to create record position numbers that can be used to gain direct access to data. In the product code example in 5.2.3 the data can be stored as records in a direct access file. The example shows a hash table that is an array. Arrays are volatile structures and thus, for the data to be available the next time the program runs, it will need to be retrieved from a disk file and loaded into the array. Alternatively the records could be accessed from the file using direct access as required. If the data was to be constantly accessed, the array option is likely to give the best performance as the overhead of loading and using the array would be offset by faster memory access as opposed to slower disk access. If the data was used infrequently, then the disk file would be quite suitable.

Records in direct access files do not need to be in any order. To use a hashing approach, the data records are spread out randomly over the remainder space using the remainder as the record position. In the products example the file would have 51 available record spaces of which only a

System Life Cycle

few would actually be used.

To access the records using a hashing algorithm the following steps are undertaken.

Open file

Get key

Create hash key

Use the key as the direct access address

Read the record in the position specified by the hash key

Close file

To write records to the file we would follow these steps:

Open file

Calculate the hash key address

Write record to the position designated by the hashed key

Close file

EXERCISE 7.5

1. Create a hash file structure for the students in your class. Use the surname and the person's initials to calculate the hash total by summing the ASCII values of the letters and keeping the remainder from the result of dividing the letter total by the number of students in the class.

Draw a diagram and record any clashes.

Explain in broad algorithm terms how you might overcome the clash problems.

Experiment with different divisors to see if this reduces the clashes e.g. double the size of the file, then triple the size of the file etc.

2. When using hashing it is likely that a number of record positions are left unused. Why might this cause a problem when using very large files?
3. Does hashing allow records to be stored in order? Explain your answer.
4. Why might hashing be used in preference to using an index?
5. What advantage does indexing have over hashing?
6. Convert the example in the previous section to determine the record number by using a hashing algorithm on the surname.



©IB0 7.19 COMPARISON OF THE SPEED OF ACCESS AND STORAGE REQUIREMENTS FOR THE VARIOUS FILE TYPES

Storage requirements

Sequential and direct access files require the same physical number of bytes to store the file.

Indexed files require extra storage space because of the need to store the index.

In a large system this could be quite an overhead and typical indexes need to be stored in memory while in use. For example, if a fully indexed file had 1,000,000 records and a key of 10 bytes, the index file would be 10,000,000 bytes. Thus the storage space required for the index is a function of the number of index entries and the length of each entry.

Speed of access depends on:

- access to a hard disk.
- access to an index in memory.
- access to a tape and other media.

Speed comparison

Sequential access is $O(n)$, with a worst case of n file reads. All the access involves disk reads and thus is slow.

Index access is also $O(n)$ but the access is usually done on an index stored in an appropriate data structure such as an array in memory. When the file access is complete, only one disk read is necessary, unless a partial index approach is used. In this case the sequential search of the data records is a function of the number of records in each group.

Record access is also affected by the number of logical records stored in a physical record. The number of logical records stored in a single cluster is sometimes referred to as the record blocking factor (not to be confused with the number of sectors blocked into a single cluster). The more records that are blocked into a disk sector, the faster access will be because only one physical disk read is needed to retrieve a record.

© IBO 7.1.10 LOGICAL AND PHYSICAL ORGANIZATION OF DATA 2004

The physical structure of data stored in RAM is that of a sequence of data items stored in linear order. The data items are stored as set of binary bits.

The physical structure of data stored on an secondary storage device such as a disk is simply a sequence of binary bits encoded in some way.

The logical structure is the abstract way that the computer program is able to access the data. The logical structure is under the control of the program designer.

In RAM, logical structures can be represented using a range of data structures:

Array data structures have the logical structure of a list. Logically the list can be accessed using a linear search where individual elements are accessed in the physical order they are stored in RAM.

Array structures can also be accessed using a binary search, which is logically very different to the physical structure.

Linked lists allow a logical structure that is similar to the physical list structure of the data.

Binary tree structures are logically very different structures to the physical structure of the data as a list. Arrays can also be used to create a logical binary tree.

System Life Cycle

On disk, the physical file structure resembles a list or sequence of bits. The data can be accessed using an index to access the data in logical alphabetic order.

In summary, the physical structure is simply that of a sequence or collection of bits. The logical structure is imposed on this by the data structures and index methods used by the programmer.

© IBO 2004 **7.1.11 EXTERNAL SORTS**

Data stored in sequential files needs to be in a particular order. As records are added, they need to be placed in the correct order. This requires the file to be sorted.

To sort a file we normally read the data into an appropriate set of arrays i.e. one for each field in the record and the arrays are then sorted using the appropriate key. When the sort is complete, the corresponding array's elements, which comprise the record of the file, are written out to a new sequential file.

Today's computers have very large amounts of RAM or primary memory. It is not uncommon for PCs to have in excess of 64mb. However, disks are also very large and files can easily exceed the available primary memory.

To sort sequential files of this nature, a 'merge sort' is required. An example is covered below.

© IBO 2004 **7.1.12 EXAMPLES OF DIRECT ACCESS FILE HANDLING**

An example of a simple sequential file system is shown above.

Example 1: General three field record file.

This worked example uses a simple record structure of three fields as shown below:

pKey acts as the primary key and is equal to the relative record position of the record on disk.

name holds a String data value e.g. name of a person.

field1 is an int field.

The record is defined in the Record class and is simplified to not use accessor and edit methods.

In this example the recordLength variable is set to 40 bytes. The record is made up an int+String+int and, provided it does not exceed 40bytes, the direct access reading and writing operate. In a following exercise the reader is asked to improve on this fixed record length design.

Comments have been placed in the code to guide the reader. It is hoped that the code is understandable without a long additional explanation.

```
import java.io.*;
public class directDemol
{
    RandomAccessFile rf; //direct access object
    int recordLength=40; //fixed length of record
    int recordPosition=0; //relative record address
    final int MAX=200; //maximum number of records in file
    Record current = new Record(); //current record

    public static void main(String args [])
    {
        new directDemol();
```

```

class Record

    int pKey;
    String name;
    int field1;

public directDemol()
{
    //attempt to operate on file using try/catch/exception handler
    try
    {
        //create the direct access file object for reading and writing
        rf = new RandomAccessFile("randomFile1", "rw");
        //check that the file maybe empty
        if (rf.length() == 0)
        {
            System.out.println("File empty");
            initFile(); //initialize file if empty
        }
        //make pKey = recordPosition just for the example
        writeRecord(10, "Smith", 23, 10); //write record to file
        current = readRecord(10); //read record to current from file
        displayRecord(current); //show current record to screen
        rf.close(); //close file
    }
    catch (Exception e)
    {
        System.out.println("File error");
    }
}

void initFile()
// set all records to have the record structure -999+empty+-999
{
    try
    {
        for (int i=0; i<MAX; i++)

            rf.seek(i*recordLength);
            rf.writeInt(-999);
            rf.writeUTF("empty");
            rf.writeInt(-999);

    }
    catch (Exception e)
    { }
}

```

```
void writeRecord(int pk, String n, int f, int r)

    try
    {
        rf.seek(r*recordLength); //seek the position
        rf.writeInt(pk); //write the primary key
        rf.writeUTF(n); //write the String
        rf.writeInt(f); //write the int
    }
    catch (Exception e)
    { }

Record readRecord(int r)

    Record rec = new Record(); //create new record
    try
    {
        rf.seek(r*recordLength); //seek record
        rec.pKey = rf.readInt(); //read int as first field assign to pKey
        rec.name = rf.readUTF(); //read String
        rec.field1 = rf.readInt(); //read int
    }
    catch (Exception e)
    { }

    return rec;//return the record

void displayRecord(Record c)

    String rec = c.pKey+c.name+c.field1; Ilmake up output String
    System.out.println(rec);
```

EXERCISE 7.6

1. Compile and run the program to check that it works. The output should look like this:
IOSmith23
2. The record is set to 40 bytes, but name can vary in length and no check is done to see that it is not more than 40 bytes etc.
How many bytes is an int in Java?
How many bytes does a String take in Java in a direct access file.
Assuming a fixed length record of 40 bytes, how long can name be?
3. Modify the writing of the String name to be always a fixed length as per your calculation above, i.e. you will need to add extra spaces.

4. Modify the Record class to include suitable accessor and edit methods and a `toString` method to return the contents of the record.
5. Add some suitable data validation. For example, add a range check on the `pKey` value or check that the name is always less than a suitable length.



Example 2: Simple application of a hashing function

This example uses the same record structure as above and includes a more complex Record class, which also shows how to address question 4 above.

The major difference is the way that the relative record position is calculated. This is done using a hash function.

The `writeRecord` method accepts the current record and the relative record position. The relative record position is calculated by a call to the hash function/method `hashCode`. The method `hashCode` accepts the concatenated String formed from the `pKey` and the name and the maximum number of records allowed i.e. 200 in this example. The function returns the relative record position and this is assigned to the variable `hashRecID` as shown in the line of code below.

```
int hashRecID = hashCode(current.pKey+current.name, MAX);
```

The value of `hashRecID` is then used to write and read the record.

The hash function operates by simply converting the input String `pKey+name` into a ASCII `hashTotal` and then using this value to determine the hash value by evaluating `hashTotal%hashSpace`, where `hashSpace` is 200. This value is returned by the hash function and is used as the relative record number.

The only aspects that are commented relate to the operation of the hashing operation.

```
import java.io.*;

class Record

    int pKey;
    String name;
    int field1;

    public Record(int pIn, String nameln, int field1ln)
    {
        pKey = pIn;
        name = nameln;
        field1 = field1ln;

        String toStringRecord()
        {
            return pKey+name+field1;

        int getpKey ()
```

System Life Cycle

```
    return pKey;

    String getName ()
    {
        return name;

    int getField1 ()
        return field1;

    void editpKey(int newPkey)
        pKey = newPkey;

    void editName(String newName)
        name = newName;

    void editFeild1(int newField1)
        field1 = newField1;

    }

public class public directDemo3
{

    RandomAccessFile rf;
    Record current;
    int recordLength=200; //arbitrary length
    int recordPosition=0;
    final int MAX=200;

    public static void main (String args [] )
    {
        new public directDemo3();
    }
    public directDemo3()
    {
        try
        {
            rf = new RandomAccessFile("randomFile1", "rw");
            if (rLength()==0) System.out.println("File empty");
            initFile();

            //make pKey    recodPosition just for the example
        }
    }
}
```

```

current = new Record(lO,"Smith",23);
//determine the record position-relative record number
int hashRecID=hashCode(current.pKey+current.name, MAX);
writeRecord(current,hashRecID);
    //write record using record position
current = readRecord(hashRecID);
    //read record using record position
displayRecord(current);
rf.close();
}
catch (Exception e)
{
    System.out.println("File error");
}

```

```

void initFile()

try
{
    for (int i=0; i<MAX; i++)

        rf.seek(i*recordLength);
        rf.writeInt(-999);
        rf.writeUTF("empty");
        rf.writeInt(-999);

}
catch (Exception e)
{ }

```

```

void writeRecord(Record c, int rec)

try
{
    rf.seek(rec*recordLength);
    rf.writeInt(c.pKey) ;
    rf.writeUTF(c.name);
    rf.writeInt(c.field1);
}
catch (Exception e)
{ }

```

```

Record readRecord(int r)

Record c = null;
try
{
    rf.seek(r*recordLength);

```

System Life Cycle

```
c = new
Record(rf.readInt(),rf.readUTF(),rf.readInt());

catch (Exception e)
{ }

return c;

void displayRecord(Record c)

System.out.println(c.toStringRecord());

int hashCode(String code, int hashSpace)

int hashTotal=0; Initialize hash total to zero
char ch; is single character holder variable
for (int i=0; i<code.length(); i++)
{
    ch = code.charAt(i); //get the single char at the position
    hashTotal = hashTotal + ch; add to accumulating total

    return hashTotal%hashSpace; //determine hash value and return
```

EXERCISE 7.7

1. Compile and test the program.
2. How might you cater for clashes? Add a way to check for collisions and then resolve the collision by searching from the start of the file until a blank record is found and finally write the desired record in that position.
3. Add a suitable interface to enable the following functions to be performed:
 - a. Add new record.
 - b. Access existing record.
 - c. Edit an existing record.
 - d. Delete an existing record.
4. Using your knowledge of linked lists or binary trees, implement a fully indexed direct access system using the name. Consider allocating the pKeys and relative record positions on an independent random basis.

8

Chapter contents

The Case Study



8 THE CASE STUDY

To allow teachers to examine real world applications in depth, the IE introduced the Case Study which is a description of an existing computer system. This case study is typically distributed to schools about 6-9 months before the final examination but is also available earlier via the IE online curriculum website at <http://online.ibo.org> - follow the 'curriculum resources' link).

From 2004, first examinations in 2006, the IE plan to use each case study over a two-year period (four examination sessions) instead of the present 1 year cycle.

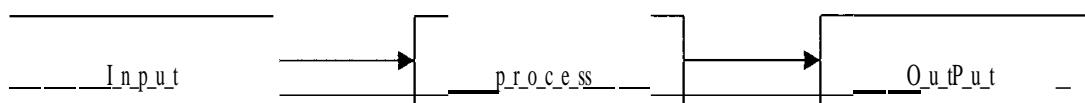
This section draws on material presented in chapters I and 3 and, in many cases, further develops the themes presented. Our approach here will be to use the previously presented case study material to illustrate the types of issues that can be presented to students. The case studies released to schools at the time of writing are:

- Sample: Medical Applications of Computers
- 2000: The Use of Computers at a Large Bank
- 2001 The Use of Computers in Weather Forecasting
- 2002 Computer Technology and Human Evolution Research
- 2003 Convenience and Protection versus the Fear of Big Brother
- 2004 Computer Aided Engineering

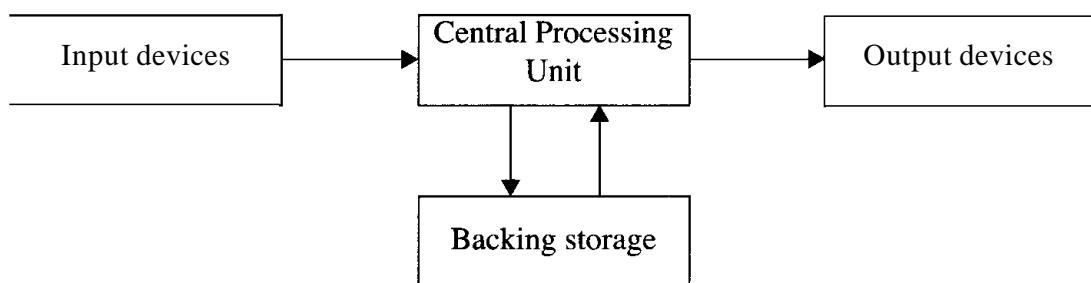
PARTS OF A SYSTEM

The main parts of any computer system follow the input-process-output model of data flow:

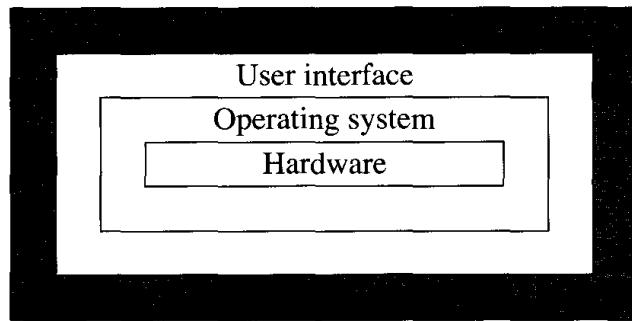
Figure: 8.1



In hardware terms, it is only necessary to add some backing storage since the CPU has only RAM (temporary storage) and ROM (read only storage) in its primary memory:



The user would find it difficult to deal directly with the hardware since all operations at this level are carried out in binary machine code. Therefore, successive layers of software have developed - operating systems (including the user interface) and applications software:

Figure: 8.2

DATA IN A COMPUTER SYSTEM

One of the important processes identified in software development (see chapter 1) was analysis and fact-finding. This involves carefully identifying the data which needs to be held and processed by a system.

In a bicycle rental system, data can be collected via a manual system, using record cards, for example:

| Bicycle number: | IP201 | | Purchase date: | 18/07/00 | |
|-----------------|------------------|---------------|----------------|---------------|-------------|
| Make: | Elektra | | Value: | \$185.00 | |
| Model: | 18-speed de-luxe | | Hourly charge: | \$4.25 | |
| Date and time | Renter's ID | Date and time | Renter's ID | Date and time | Renter's ID |
| | | | | | |
| | | | | | |
| | | | | | |
| - | | | | | |
| - | | | | | |
| - | | | | | |

In order to fully describe the system it is necessary to consider what happens under many different circumstances, what happens when a bicycle is returned among others. In order for the total charge to be computed, the hours of renting need to be calculated and multiplied by the hourly charge.

Often, the data that needs to be held and processed in a system is identified using data flow diagrams. A data flow diagram typically uses the following symbols (although they are not complete consistency in practice):

The Case Study

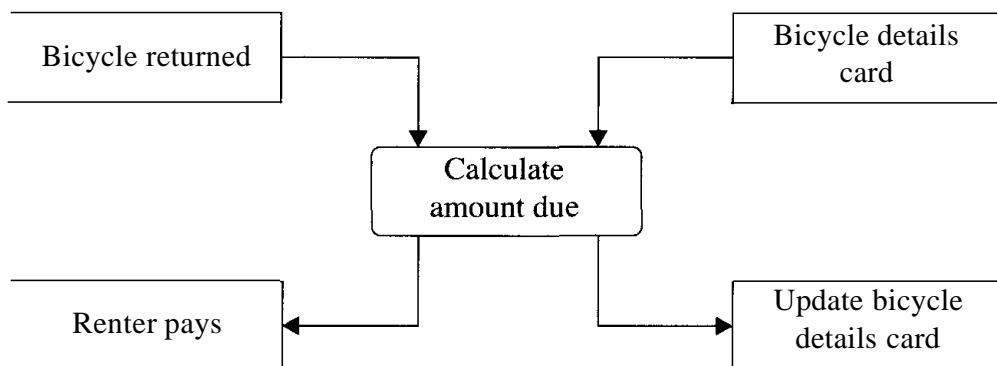
Figure: 8.3

- _____
- _____
- _____
- _____
- _____
- _____
- _____
- The box with rounded corners (maybe square or rectangular) which represents a process. An example would be the calculation required above.
- The box with an open right-hand side representing a data store. This would be the index card for the bicycle.
- The closed rectangle is a source or sink (destination) of data. It shows the limits of our diagram. How the data gets into or out of these boxes is not a concern of this diagram.

These boxes are accompanied by arrows to show the direction of data flow. Often the arrows have other associated information such as the data that is on the move or, in other systems, the person responsible for the process.

For a complete picture consider returning the rented bicycle using the documents and data in the manual system:

Figure: 8.4



No reference is made to hardware used. For example the calculation may be done with a calculator (or abacus or on the back of an envelope etc.), this does not concern the data flow. However, when a new system is to be created, it is important to know that this process needs to be carried out.

EXERCISE 8.1

1. Construct a data flow diagram showing what happens when a bicycle is rented out.
2. List all the other situations where updating of data may take place in this system.
Even a small system will require several diagrams to describe it completely.
3. Study your existing student registration system. Draw a series of data flow diagrams that illustrate where data is stored, how it is used and when it is updated.
4. Examine the Weather Case Study. Construct a dataflow diagram showing how a person at a TV station would prepare a weather forecast for presentation.



DATA CAPTURE AND PRESENTATION

The data flow diagram shows only data flow without reference to mechanisms of capture and display. There are a great many ways to capture data for use in a computer system. The main devices are described in chapter 2 and their methods can be classified in the following way:

| Input method | Example devices | Example of use |
|-----------------------|---|---|
| Manual data entry. | Keyboard, mouse, joystick, touch screen, touch pad. | Adding client or book records in a library. |
| Direct data entry. | OCR/OMR scanners, MICR reader, barcode scanner. | Lending a book, locating borrower details. |
| Automatic data entry. | Sensors - temperature, sound, pressure, light etc. | Controlling the temperature in the library. |

Similarly one can classify output devices in common use:

| Output method | Example devices | Example of use |
|--------------------------------|---|--|
| Temporary display. | VDU, LCD display, lights. | Showing the price of an item at a POS terminal. |
| Permanent display. | printers, plotters. | Printing a receipt at a POS terminal. |
| Electrical! mechanical output. | Actuators - relays, switches, converters etc. | Sending credit card details to a bank from a POS terminal. |

There are so many input or output devices that not all of them will fall into a particular classification.

EXERCISE 8.2

Study the Human Evolution Research Case Study (IEO 2002).

1. Identify and describe all of the input and output devices in the case study.
2. Explain why each of these devices is appropriate to a given task.



DATA PROTECTION, ERROR DETECTION AND RECOVERY METHODS

Data protection

Data that is in systems needs to be protected against accidental or deliberate loss or damage. Some examples of threats to data are:

- Unauthorised users (hackers) may gain access and alter or remove data.
 - Physical media (discs, tapes) may be stolen.
 - The hardware may be stolen (along with fixed backing store).
- There may be fire or flood damage.

A particular danger of data on networks is that data may be copied leaving no evidence that a copy was made as data may be accessed over networks remotely.

An important method of protecting data from unauthorised access is the use of passwords and privileges, particularly on networked systems. Only people who have been given a logon name and allowed to select their password can access a system or parts of a system. Privileges can be assigned to users; low-level users can access data but not change it or make a copy to a local floppy disc, for example. In a supermarket you may see this happen if a price has been incorrectly recorded in the computer database. The check-out person usually does not have the right to change prices and will therefore call a supervisor who will tap in an access code and correct the situation.

To be effective, passwords must be of a reasonable length (usually 6 characters or more) and hard to guess (not your partner's child's/dog's/parakeet's name) and containing special symbols besides alphabetic characters.

If, while you're at the supermarket, you look around carefully you may see examples of physical security -locking computer systems in special rooms accessible only by personnel equipped with key cards or special codes.

When data is transmitted over networks it may be encrypted if especially sensitive such as large financial transfers, police data on suspects, government military secrets so that it is very difficult (not impossible) to decode. Encrypting data helps to ensure that, even if data is accessed, it is not readable. A PIN number on the magnetic strip of a bank card is encrypted for this reason.

Error detection

Methods used to detect and correct errors are discussed in section 3.6.

Data recovery

To protect data against irreversible damage, backup copies are kept in a safe place; in most cases they will be kept in a different building. These can be used to restore a system to the state existing at the last backup. When did you last backup your data? Most businesses cannot function without the data from their computer systems; therefore they must have a thorough and thoroughly tested backup strategy.

Important real-time systems such as air-traffic control systems may use mirrored systems where two computers run the processing simultaneously - if one breaks down the other can continue operation.

EXERCISE 8.3

In the Human Evolution Research Case Study:

- Identify methods which have been used to ensure security of data.

In the Weather Case Study:

- Explain why the data from the data collection sites is coded before being forwarded to the National Weather Service.

For each of the above case studies:

- Explain the possible consequences of data corruption.

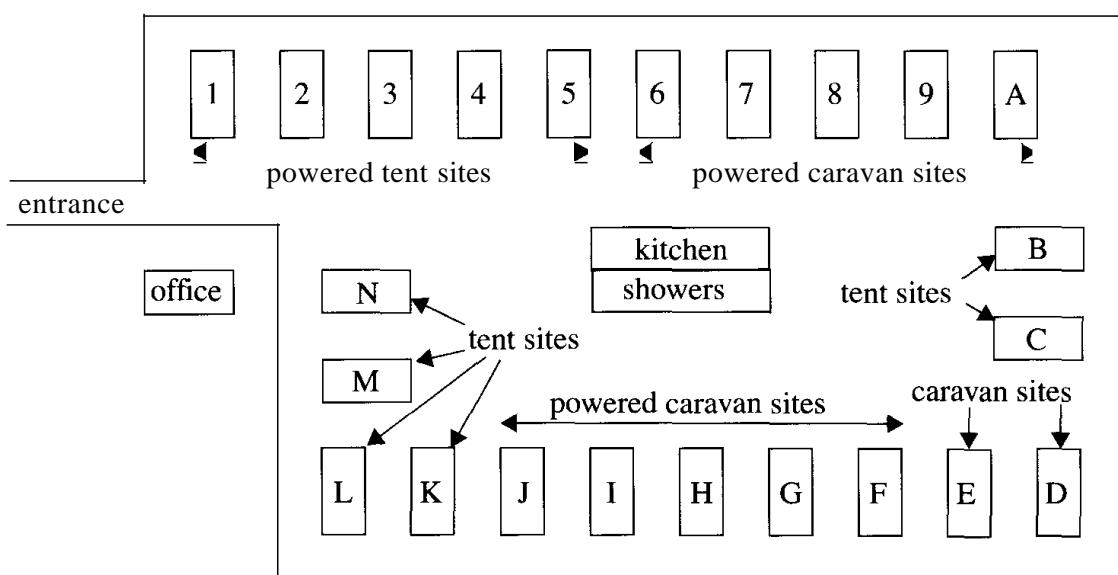


DESIGN OF APPROPRIATE DATA STRUCTURES

This topic has been covered in section 1.5.3 where the relevant data structures were described together with examples of when it would be appropriate to use them.

EXERCISE 8.4 (REVISION)

- A holiday park offers sites for tents and caravans some of which are supplied with power. To book a space the following system is used:



When a customer arrives the following details are collected and entered onto a form:

- Name.
- Address.
- Vehicle Registration Number.
- Site Allocated.

Information is held about each site in a card index file:

- Site number.

The Case Study

- Powered/not powered.
- Type (tent or caravan).
- Daily charge.

Carefully consider the data flow in the following scenarios:

- A new customer arrives.
 - A customer leaves.
- A site has its status changed (e.g. from powered to non-powered).

Draw data flow diagrams to illustrate the above scenarios.

Discuss, including diagrams, the data structures that could be used to hold the data for the system. Remember that the 'discussion' keyword requires you to consider a range of possible data structures and give reasons for selecting the ones you did.

HARDWARE COMPONENTS

You have already studied a range of input, output and backing store devices in Chapter 3. For each one given there, copy and complete the following table related to the holiday park scenario of the last section. An example is given for you:

| Device | Could be used? | Example of use | Advantages | Disadvantages |
|-----------|----------------|--------------------------------|---|---|
| Keyboard. | Yes. | To enter details of customers. | Easy to enter alphanumeric data such as an address. | Slower than direct entry methods such as a barcode. |

USER INTERFACES

Early operating systems operated with typed in commands (requiring command-line interpreters or CLI's) while later ones have developed graphical user interfaces (GUI's). The main features of these interfaces are:

Command Line Interfaces

Easier to implement for a programmer.
Require less memory to run. Can be run on systems without graphical monitors.

Users need to remember specific commands so new users can find them harder to use.

Long term users may find it quicker to type in a command at the keyboard than to use a mouse or other pointing device.

Graphical User Interfaces

More complex to implement. Require more memory, a pointing device and a graphical monitor.

Icons (small images) help users to remember commands, file types etc.
Commands are grouped in menus.

New users will find it easier to use because they do not have to remember specific commands.

Graphical User Interfaces are sometimes described using the term WIMP, variously interpreted as;

- Windows
- Icons
- Menus
- Pointers

- Windows
- Icons
- Mice
- Pull-down Menus

EXERCISE 8.5

Using the Human Evolution Research Case Study:

1. A researcher is preparing images to be posted on a web page while a computer technician is setting up a filtering task. Explain the advantages and limitations of both GUI's and CLIs for each of these tasks.

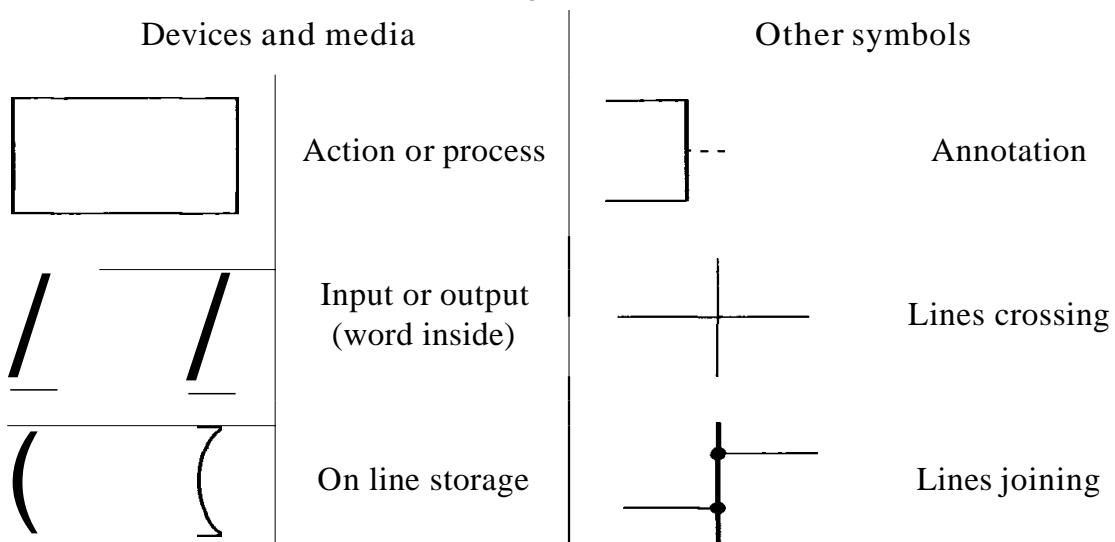


SYSTEMS FLOWCHARTS

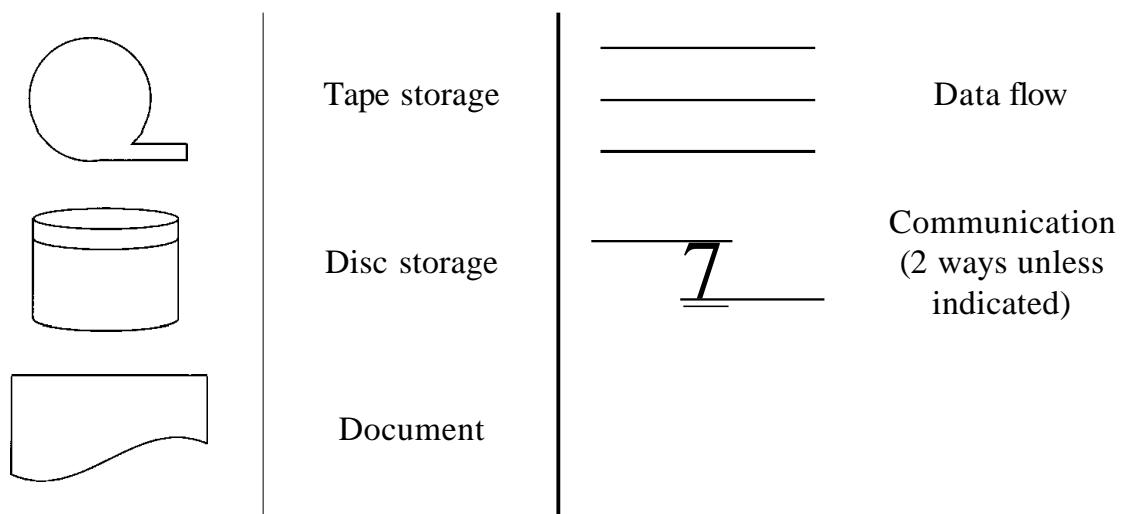
Systems flowcharts are designed to link data flow and processing operations to specific pieces of hardware. They are sometime known as input-output (systems) flowcharts. They should not be confused with flowcharts used to show the structure of algorithms.

As with data flow diagrams there is a wide variation in symbols used to implement systems flowcharts; below are the ones specified by the IEO in the Computer Science Subject Guide:

Figure: 8.5



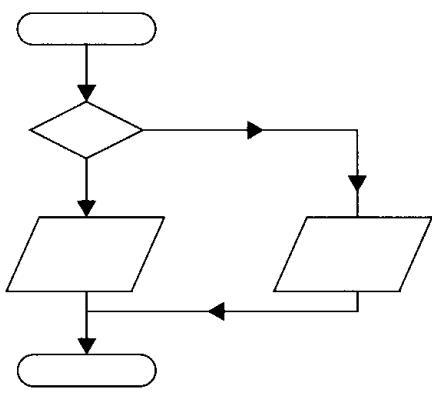
The Case Study



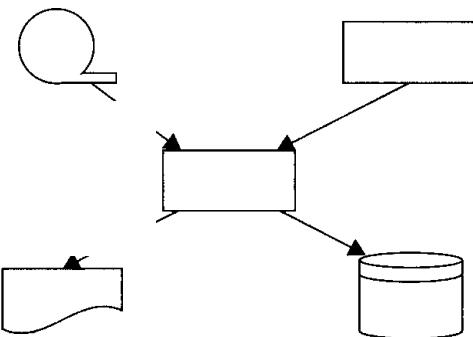
Students generally seem to have great difficulty with systems flow charts; the main problem seems to be that they think in terms of linear algorithm flow charts (I believe there is a conspiracy of subversive maths teachers at work here). The following diagram shows the outline shape of the four different types of chart used in this book:

Figure: 8.6

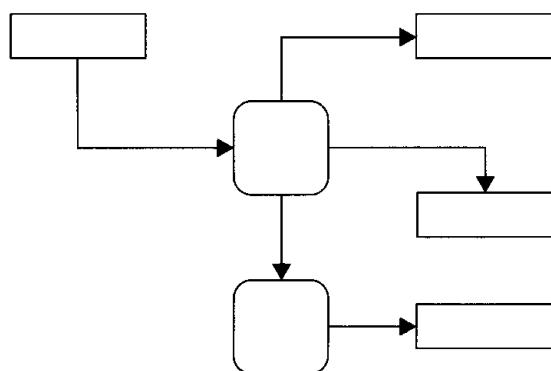
Flowchart



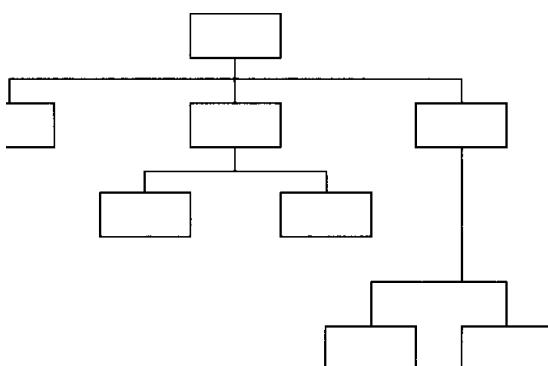
Systems flowchart



Data flow diagram



Module diagram (Structure chart)



Points to note:

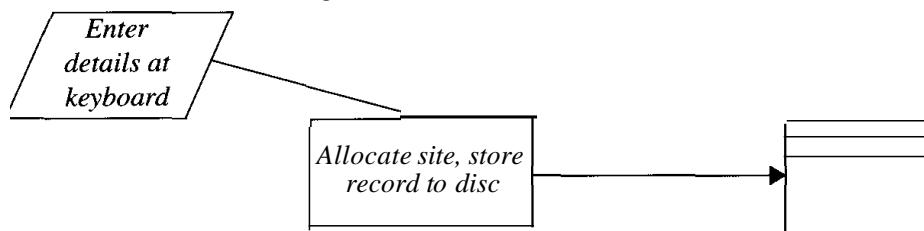
- Flowcharts are used to describe algorithms (although pseudocode is often preferred these days).
- Systems flowcharts are used to describe input-process-output in computer systems, they are the only charts to refer to hardware devices.
- Data flow diagrams refer to data objects and processes (people, paper files, computer files, etc.).
- Module diagrams are used to split a large problem up into several smaller ones (stepwise refinement). This makes the problem easier to solve and divide up among a programming team.

Simple processes

Consider the case of the holiday park described earlier. If this system were to be transferred to the computer, the following tasks (among others) would have to be carried out:

Registering a new customer could involve entering the new customer details at the keyboard, entering the site location and storing these to a database:

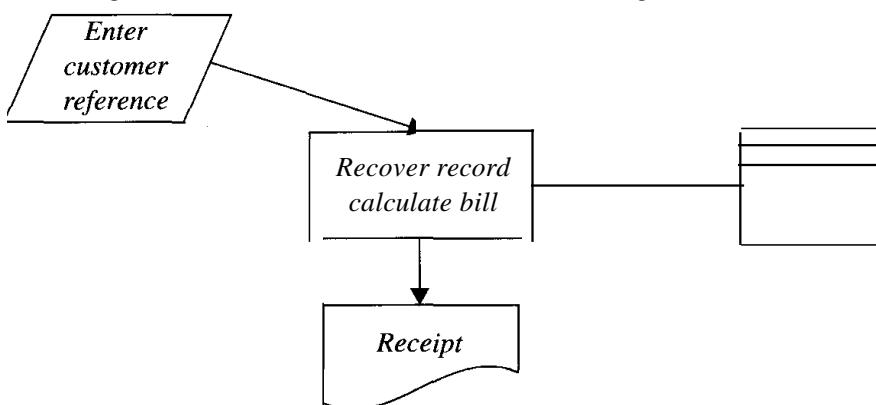
Figure: 8.7 A, B & C



More detail can be added, if required, as an annotation:



The process of checking out a customer would involve calculating the bill and issuing a receipt:



In chapter 3 you will have studied batch, online and real-time systems; here we examine how these types of process are represented in systems flowcharts.

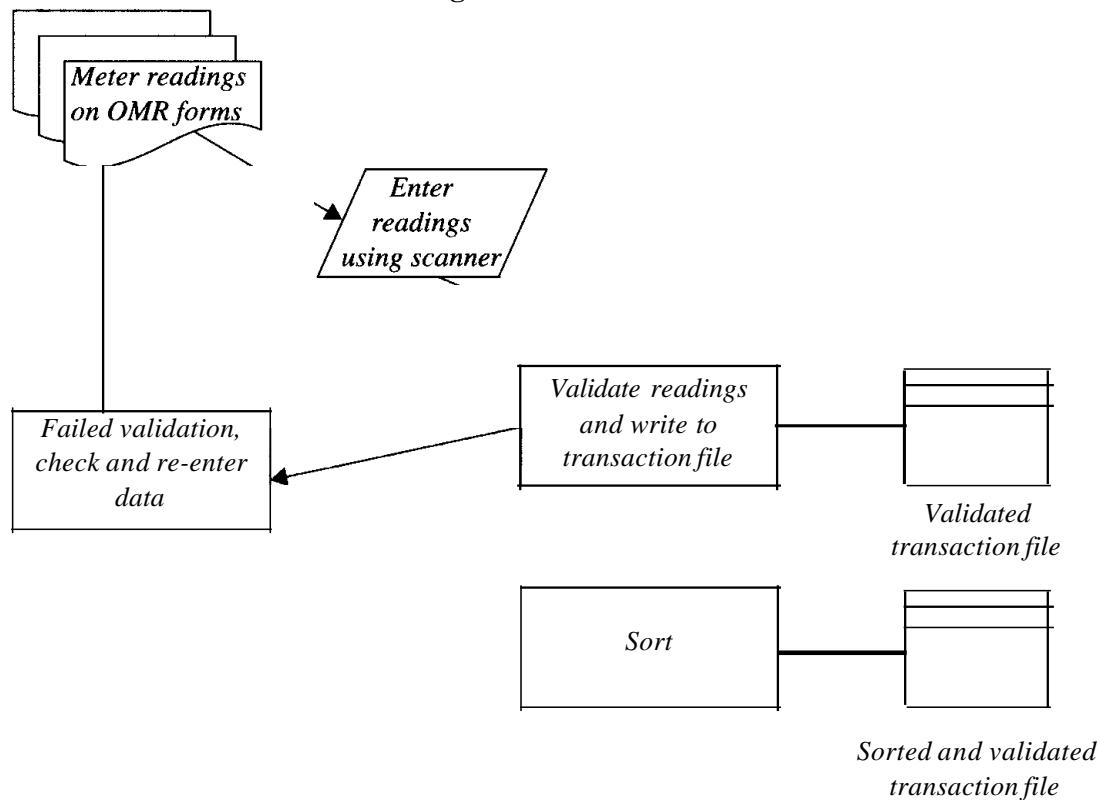
The Case Study

Common batch processing tasks

In batch processing, data is gathered first and then processed in one go. Typical operations update a master file using a sorted transaction file. Therefore, in many batch processes (cheque clearing, electricity billing, payroll processing, batch update of a stock file, etc.), paper documents will be collected, validated (section 3.6) and sorted. Items rejected by validation may be corrected and re-entered:

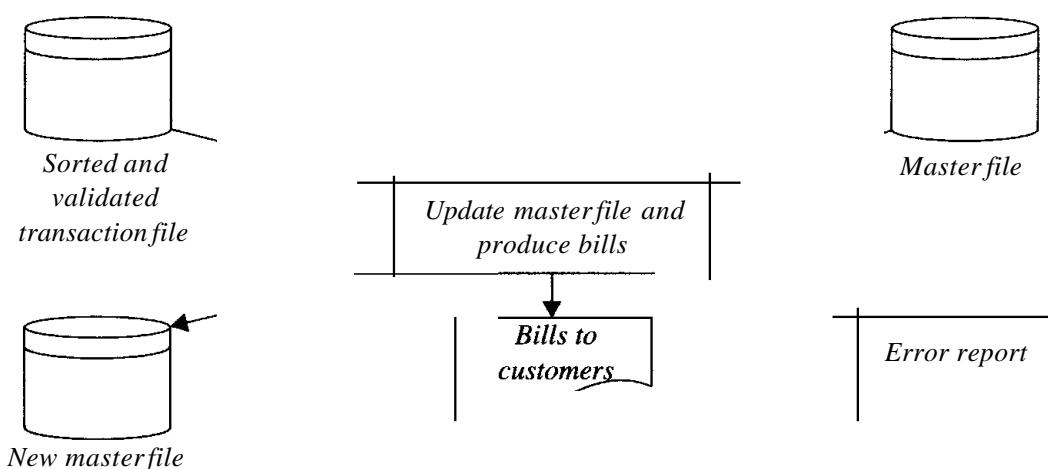
Example for electricity billing:

Figure: 8.8



The sorted transaction file is then used calculate the amount of electricity used (by subtracting the reading in the master file from the reading on the transaction file) then to produce a bill and to update the master file with the new reading. Of course some errors may still occur. If the meter reading is incorrect, bills can be very large or perhaps negative. These errors are typically recorded on a hard copy print out.

Figure: 8.9



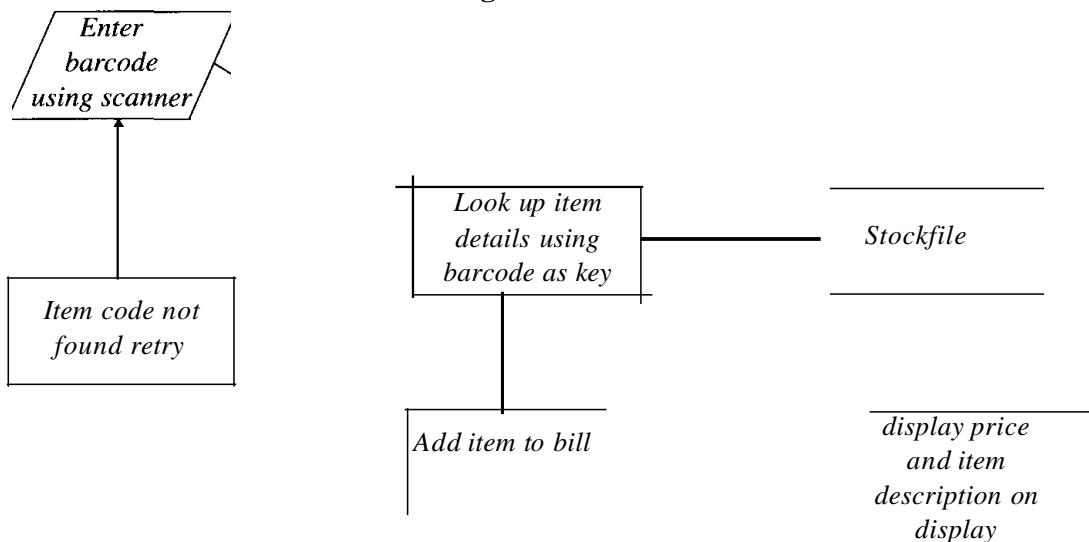
Activity

You will have studied cheque processing and payroll processing systems in Chapter 3. Construct a systems flowchart for each of these systems.

Common online processing tasks

Recall that, in online processing, any transactions are used to update a database immediately. A typical example is supermarket stock control where barcode scanners at a **PoS** terminal read the barcode, look up the item details in a stock database, return the details to the **PoS** terminal where they are printed on a receipt and shown on a display.

Figure: 8.10



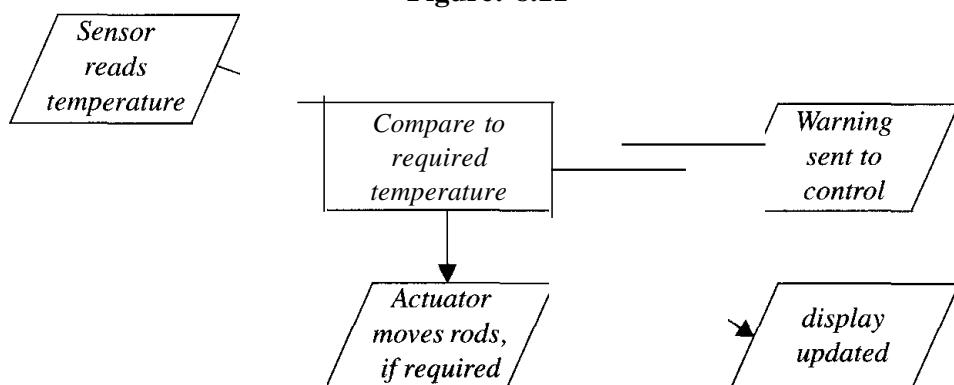
Activity

You will have studied airline reservation and library cataloguing systems in chapter 2. Construct systems flowcharts for these processes.

Common real time processing tasks

Real time systems are a type of online processing system in which the processing is fast - the input data is processed quickly enough to affect the next output of the system. Usually such systems collect their data through sensors (automatic data entry). A typical example is a system which monitors a nuclear plant's reactor core. In some reactors a set of rods is inserted into the core to damp the nuclear reaction. When this is done, a warning is sent via a communications system to the control room. This situation might be represented as follows:

Figure: 8.11



The Case Study

Activity

You will have studied air traffic control and patient monitoring systems in chapter 3. Construct systems flowcharts for these processes.

Further exercises on systems flow charts are found below.

INTEGRITY AND SECURITY OF DATA

In chapter 3, the distinction was made between the security and integrity of data.

The security of data refers to keeping it safe from accidental or deliberate loss by unauthorised persons. This includes both physical security and software security as previously described.

The distinction may be made between security and safety of data in some texts – safety would refer to the need for backup strategies to recover data after loss, however caused. In the IE guide it appears that integrity is used to mean maintaining the safety of data. This would cover strategies for backing up and recovering data.

These measures of integrity and security are vital to companies holding data and in many countries of the world are a legal requirement.

There are many disturbing (and well-documented) stories of the ease with which students and the like (malcontents, rebels, hackers, freaks etc.) can break into systems which are supposedly secure (including banking and military computer systems). The implications for safe-keeping of personal data are extensive (e.g. it would be possible for a malicious hacker to alter medical and financial data on people thus preventing them from getting work or opening a bank account etc).

EXERCISE 8.6

1. The Medical Case study describes the process of stock control in a hospital. Identify one (different) field in each record which could be changed by a user of the system attempting to steal drugs.
2. Explain two precautions the hospital could take to help ensure the security of drug data.
3. When drugs are taken out of store, the transaction is recorded on a disc file which is then used to update the master stock file. Explain why this transaction file is kept.



RELIABILITY OF SYSTEMS

The reliability of any system is only as good as the data that is entered. The correctness of data in a system may be described as the 'integrity' of the data. As systems become more complex and autonomous (making key decisions on software-based rules) then mistakes can happen, with serious consequences.

An X-ray machine that is designed to emit X-rays at a given power for use in a cancer treatment process can be computer controlled. If, due to a data entry error, the dose of radiation is entered wrongly, a patient could be killed or, at least, badly burnt.

Closer to home, students would probably like to be sure that their grades are correctly entered into whatever computer system the school is using. These grades may be used to write reports, which, in turn, may be used to write university, college and employment references. Thus an incorrect grade could have a direct impact on the future of a student.

When the systems are actually in place and running, the consequences of hardware or other system failures has also to be taken into account. It is not too serious if a bank mainframe computer goes offline for a short time, the transactions can be recorded on disc or even on paper and entered into master files later. However, as mentioned previously, systems like air traffic control and the monitoring of a nuclear power plant cannot fail and therefore two complete systems operate in parallel in case one fails. In extreme cases, a triple system may even be used.

EXERCISE 8.7

1. The Weather Case study describes the process of data collection for input into weather models. Discuss the implications of mistakes being made in collection or transmission of the raw data.
2. A hospital has medical records on paper and is transferring them into a new computer system. Discuss the importance of transferring these records correctly.
3. Two systems described in the hospital case study are patient monitoring and stock control of drugs. Compare the importance of the reliability of each of these systems.
4. In the 'Big Brother' case study, many attempts to obtain private data about people were described. Explain three implications for surfers of a company obtaining inaccurate data.
5. In the CAD/CAM case study, computer programs were written to control the operation of industrial machines. Compare two situations in which a programming error might have serious consequences.



THE CASE STUDIES

An effective way to use the case studies is given in the following exercise:

EXERCISE 8.8

For this exercise you will need

- One of the case studies.
- The Objectives and Action Verbs from the Subject Guide page.
- The Common Core Assessment Statements pages.

Different action verbs are used for different objectives. Each assessment statement has an associated objective.

Using action verbs appropriate to each assessment statement, construct questions and mark schemes on each listed topic that are appropriate to each case study. This work is best carried out in pairs or small groups.

This exercise is intended to familiarise students with:

- The meaning and significance of the action verbs.
- The breadth of the syllabus common core.

The Case Study

- The nature of the case study.
- Examination question structure.

The questions and mark schemes which are generated can be distributed to all students as a revision aid.

This exercise need not be carried out after the complete common core has been covered but could be done after each section of the programme has been covered. This would ensure that the appropriate Case Study has been thoroughly read by the end of the course.

A slightly different approach could be to have groups of students investigate different aspects of each case study (e.g. hardware aspects, network implications, data structures, security and integrity of data, etc.) and then present their findings to the rest of the class.



FURTHER EXERCISES 8.9

1. In the Banking case study, there is an article describing a retinal scan system which could potentially be used to ensure security at an ATM.
 - a) State three input devices required for this system.
 - b) Outline the type of interface that would be used in this system.
 - c) Using the 4-point description given in the article, construct a systems flowchart to describe the system.
 - d) Compare the security of the proposed system with that of using a PIN number encrypted on the card.
2. One of the hospitals (hospital A) in the medical case study needs a delivery of drugs. An email request is sent to another hospital (hospital B) in the system. The email request is entered into a transaction file which is then used to update the stock master file. A printed record of the transaction is made and the drugs are removed from store at hospital B and sent by van to hospital A.
 - a) Construct a data flow diagram of this process.
 - b) Construct a systems flow chart of this process.
3. In the weather case study, weather data is used from the National Weather Service; this data is in METAR format.
 - a) Explain why this format is unsuitable for presentation in newspapers, on radio and on television.
 - b) Explain why this format is suitable for processing by a computer system.
 - c) Discuss the suitability of different types of output from a computer system that could be used for presenting weather data using newspapers, radio and television.



[Chapter I discusses the possible social, economic, political and cultural aspects of technology as well as current trends in technology. The following exercises address these themes in the context of different case studies.

EXERCISE 8.10

1. The banking case study identifies many social effects of the introduction of modern computer systems (article 3, p 6).

Identify two benefits and two disadvantages to:

- a) the bank employees.
- b) the bank customers.
- c) the bank owners or shareholders.

2. The Big Brother case study details attempts by companies to obtain data on individuals. This is also an activity conducted by most governments (whether in the 'free' world or otherwise).

Outline two benefits and two disadvantages to citizens of this type of activity.

3. The CAD/CAM case study details some of the effects of globalisation with regard to mass-production of goods.

Explain one benefit and one difficulty created by the advent of CAD/CAM systems for:

- a) developed countries.
- b) less-developed countries.
- c) consumers.

The Case Study

9

Chapter contents

9.1 The Dossier

9.1 THE DOSSIER

INTRODUCTION

The IE Computer Science programme allocates 35% of the final grade to the dossier project. It is a significant piece of work and requires a suitable amount of time to be devoted to it. At Standard Level, 25 hours teaching time is expected to be devoted to the dossier and 35 hours at Higher Level. This is in addition to any time spent on teaching the programming concepts outlined in Topic 2 and further work outside the classroom would normally be expected of students.

The dossier is probably the largest single piece of design and programming you will have tackled in Computer Science and, perhaps, in any subject. It is therefore very important to be organized and to stick to the deadlines set by your teacher.

Many students make the mistake of thinking that the dossier is only about programming and spend all their time developing a (quite possibly brilliant) program. Unfortunately, many of the marks available are for analysis, design, testing and evaluation so even a brilliant program will only score approximately 25% of the available marks without proper attention to these criteria. Usually the other criteria are sadly neglected by students in a rush to start programming.

It is very important to remember, right from the start, that the examiner only sees what you have written down. The fact that you have discussed data structures at length with your teacher will not score any marks until you have the discussion written down as part of your dossier.

DOSSIER STRUCTURE

This is a summary of what should be contained within the dossier for each assessment criterion. The actual structure is given in the subject guide. The IE publishes Teacher Support Material that exemplifies the contents of each section together with examples from student dossiers. This should be available in early 2005.

The 'new look' dossier now has 4 major sections:

| Stage | Criteria | Maximum award | Corresponds in old program |
|-------------------|--------------------------|---------------|----------------------------------|
| A Analysis | A1 Analysing the Problem | 4 | A Analysing the Problem |
| | A2 Criteria for Success | 4 | none |
| | A3 Prototype Solution | 4 | none |
| | | 12 | |
| B Detailed Design | B1 Data Structures | 4 | C Data Structures |
| | B2 Algorithms | 4 | B Documenting the Design Process |
| | B3 Modular Organisation | 4 | |
| | | 12 | |

| Stage | Criteria | Maximum award | Corresponds in old program |
|-----------------|---------------------------------------|----------------------|---|
| C The Program | C1 Using Good Programming Style | 3 | J Using Good Programming Style |
| | C2 Usability | 3 | G Incorporating User Friendly Features |
| | C3 Handling errors | 3 | H Handling Errors |
| | C4 Success of the Program | 3 | I Implementing the Program |
| | | 12 | |
| D Documentation | D1 Annotated Hard Copy of Test Output | 4 | F Including an Annotated Hard Copy of Test Output |
| | D2 Evaluating Solutions | 4 | K Evaluating Solutions |
| | D3 Including User Documentation | 3 | L Including User Documentation |
| | | 11 | |

This makes a total of 47 and there are a further 3 marks to be awarded for a student's 'Holistic approach to the dossier' which are at the teacher's discretion.

The new total of 50 marks will give more 'granularity' in the application of descriptors (i.e. it should make it easier for teachers to award appropriate marks) and will be scaled back to 35.

The most significant changes here are the move to a requirement for a real user and the development of a prototype solution to show to that user. The intention is to provide more flexibility and realism to the design process. Candidates are not then stuck with any particular design methodology. Flexibility has also been introduced by allowing candidates to make their own criteria for success rather than a strictly data-based test plan. Such a plan may well still feature in many types of dossier.

More marks and more criteria should not necessarily mean more work for candidates. The organisation of the dossier will be different. As a rough and ready guide it should still be possible to complete an SL dossier in 40 pages or so of 'normal' quality writing, and 60 - 80 pages for HL. A lot depends on the type of problem tackled, of course.

STACiE A: ANALYSIS

This section should concentrate on the problem and not the program to be written. Dossiers which begin "I was asked by my teacher to write a program to solve a maze" are almost always

The Dossier

unsuitable because the problem (solving a maze) has not been considered as a problem, only viewed as a computer program to be written.

The use of some analysis tools is recommended, not only will this improve the dossier, it will also help the students understand this topic for the theory examination.

In addition, there is now a requirement for a real user to be involved in the project. While this could pose difficulties for some candidates, particularly in isolated areas, the user could be a person close to the candidate (a parent or colleague for example). At the analysis stage, after a prototype of the solution has been produced, the candidate is expected to obtain feedback from this user.

The prototype itself can take many forms and need not be functional. For example, the end user can be shown a mock up of a data entry screen and be asked to comment on it. The candidate would then refine the solution based on such a discussion. This approach ought to engage the candidate in more practical, less abstract, issues related to the design of computer systems.

Students could tackle this objective in several sections. For example:

Problem definition and analysis could include:

- Sample data.
- Client user requests.
- Other previous solutions.

Specific objectives of the solution, by which we mean:

- The goals being set.
- Definition of input and output data.
- Processing required.

This section will help the student to think about the data capture screen(s) required, the data structures which will be needed and the main modules of the solution.

Analytical tools. For example:

- Data flow diagrams.
- Use case scenarios.
- Screen designs.

There are many different types of tool that can be used by students. None of them are required by the assessment criteria for the dossier. However, choosing one such method is often more effective than text descriptions alone.

STAGE B: DETAILED DESIGN

The first criterion is 'data structures', although, logically, this need not be the first design step. Candidates should not see design as a single once-through process. There should be some thought put into the process of selecting algorithms and data structures that work together to achieve the desired result.

This section is an opportunity for students to put theory into practice. They should have learnt about many different types of data structure before attempting the dossier and should be able to discuss which ones are most suitable for the solution they have chosen. A good discussion should include consideration of the data structures that were not used as well as those finally selected.

The IE subject guide makes it clear that diagrams must be included for this section. Also required are sample data from the problem and, ideally, a discussion of those data structures included to achieve mastery aspects (such as arrays or records at Standard Level and linked lists or binary trees at Higher Level).

The subject guide refers to the algorithms that the candidate uses in the solution and these can be presented in many ways. Java code would not be expected at this stage. Candidates may have chosen a procedural approach (more likely for SL than HL candidates) or an Object-Based or Object-Oriented solution, or even some mixture of these approaches. The criteria call for three items:

- The data structures.
- The algorithms.
- The modules.

The criteria are written in such a way that the modules could be classes or methods of a single large object.

Descriptions of modules and algorithms can be in diagram or text form or some other mixed form such as CRC cards or UML diagrams, Use Case Scenarios, Data flow diagrams, even (shudder) flow charts. Key features should be:

- clearly identified modules.
- connections/data flow between modules.
- interior structure (data structures/algorithms).

It is likely that candidates will start to program and then realize the limitations of parts of their design approach. This is expected, even anticipated, by the revised criteria as candidates may depart from their original design ideas at the programming stage 'without penalty'.

STAGEC: THE PROGRAM

Teachers will probably be pleased that more marks have been allocated to this section although the dossier is still not a coding exercise but a problem-solving activity.

Code will be assessed in terms of its overall readability or 'programming style' as for the old criterion J. The required elements of style are:

- Abundant comments (preferably colour coded).
- Meaningful variable names.
- Consistent indentation.
- Clear, simple syntax.
- Highlighting of code for emphasis of code ownership, mastery factors, library routines etc.

The Dossier

It is an excellent idea for candidates to improve readability by presenting code listings in landscape form.

This section also includes the candidate's approach to user-friendliness (Usability issues) and robustness (error handling capabilities). All of these features are important in program development and implementation.

Usability refers to helpful instructions for the user and to the way interfaces are designed (be they text menus or windows-style interfaces). Candidates should aim to document all such user-friendly features by:

- supplying the code that produced them.
and
- annotating the relevant hard copy print out.

Annotations are most easily done by hand.

Handling errors refers to code that is used to detect and, if applicable, correct any data input or runtime errors (e.g. avoiding division by zero, or checking that a file exists before opening it). This can be shown by annotating relevant code and reproducing such code together with the error messages produced as output.

Both C2 and C3 require separate sections in the dossier; it is not sufficient to include these sections within test runs or code listings.

Section C4 allows the candidate to demonstrate that the program has achieved the objectives that were set out at the start and to show that the program functions by referencing test runs. Again, this approach allows much more flexibility than using one particular method, such as a test plan, to measure success. For maximum marks, the solution will need to achieve all the objectives set out during the analysis stage and the candidate will have to produce evidence that it does.

STAGE D: DOCUMENTATION

Sample runs should be produced that demonstrate all the aspects of your program. However, not every data input test for every screen need be shown. The testing can be annotated by hand or word-processed, but the teacher will have to confirm that the program actually produces the results claimed by the candidate.

The testing should indicate that every branch of the program has been demonstrated to work and, if appropriate, should include testing with valid and invalid data. There will be few programs that do not require any data input at all.

At a minimum you need to describe the solution that you made. For the best marks you must:

- Discuss its effectiveness as a solution to the problem identified.
- Discuss the efficiency of the algorithms used.
- Suggest improvements which could be made.

A useful strategy is to compare, point by point, the aims you set out in section A with the way the solution finally turned out.

There must be some form of simplified user manual (hard copy) which contains:

- details of hardware and software needed to load/run the application.
- instructions on installation and loading.
- sample screens to assist the user.
- examples of correct data input.

The award for Holistic approach to the dossier would normally be made by the teacher on the basis of clear guidelines given to students. The issue here is how much commitment the candidate shows in completing the dossier, which could be very subjective. To assist candidates gain a good mark in this section teachers should clearly communicate their expectations as to intermediate deadlines, the need for candidates to assist their colleagues, not by collaborative work (which is forbidden) but by making useful suggestions as to how some aspects of other projects might be tackled.

Candidates should be aware that teachers are free to add their own expectations to the list provided by IB, for example, a teacher might require 'best use of lab time' as a criterion and penalize candidates who are habitually late or frequently engaged in non-productive activities.

CHOICE OF PROBLEM

Some hints and tips:

- Problems should be open-ended so that they can be started small but expanded to fill the available time. Games programs almost always fail this test.
- Problems should be 'real world' and have real users who have real problems where possible - this makes investigation and analysis much easier. Games programs almost always fail this test.
- Projects should clearly focus on the essential requirements - every proposed module and feature should be put to the "does it enhance my chances of getting a good score?" test. Programs with neat and nifty graphic interfaces are great fun to do but can really eat into project time - and the examiner never sees them. Games programs almost always fail this test.
- Try to avoid stating the problem as "you are to write a program which..." but rather allow exploration of various solutions to gain insight and improve your analysis of the problem.

It might seem that we are advising against games programs (that's because we are). We 'never' allow our own students to do games programs - at least we do all that is in our power to stop them. In our experience students will lose at least 25% of their potential marks simply because they become involved with game-play aspects that do not impact their final project. There are students who can complete successful games projects but they are 'one in a million', just like lottery winners (no, that doesn't give you an idea for a dossier project!).

The website <http://www.ib-computing.com> has a list of suggested topics to help choose a good problem.

The Dossier

Mastery of topics

This is not the same as simply using a technique in the source code. The justification for inclusion of each mastery topic will be more convincing to the moderator when its use is justified and appropriate. Such justification should appear in the data structures discussion as well (where appropriate).

It is a good idea to ask candidates to create an 'index' of mastery for their dossiers and for the code listing. This will help them focus on the need to cover the appropriate aspects.

The IEO have made it much, much easier for students to choose a topic and then work in appropriate masteries. Under the old program it might have been necessary to adapt a topic to make the mastery factors fit. Nevertheless, students should still be prepared to explain how they will gain the required masteries when giving an initial description of their project.

Finally

In the best of all possible worlds your project is a satisfying piece of work. Avoid stress by sticking to deadlines, leaving yourself a realistic amount of time to complete everything. Best of all, finish early and laugh at your poor friends. Have fun!

Computer Science

Java Enabled



The second edition of this popular book addresses the IB Computer Science Diploma Programme with reference to the assessment statements in the revised subject guide which began in 2004. The new edition is expanded and extensively revised to include the use of the Java language.

This is an ideal student text which includes classroom activities for groups and individuals as well as examination style questions. The book also has a section on completing the dossier. This will be of immense help to all students entered for the course. Diagrams and illustrations are provided to enable students to understand the key concepts.

This book is simply written to be accessible to all students whether studying in a school or on their own. Prepared by IB teachers who have had responsibility for assessment, preparation of examination papers, marking and adjudication of programme dossiers, this book draws on their experience. For several years, the authors have led workshops for computer science teachers in many parts of the world.