SCHOLAR Study Guide

SQA Advanced Higher Computing

Unit 1
Software Development

David Bethune

Heriot-Watt University

Andy Cochrane

Heriot-Watt University

Ian King

Heriot-Watt University

Heriot-Watt University

Edinburgh EH14 4AS, United Kingdom.

First published 2001 by Heriot-Watt University.

This edition published in 2009 by Heriot-Watt University SCHOLAR.

Distributed by Heriot-Watt University.

SCHOLAR Study Guide Unit 1: Advanced Higher Computing

1. Advanced Higher Computing

ISBN 978-1-906686-11-6

Printed and bound in Great Britain by Graphic and Printing Services, Heriot-Watt University, Edinburgh.

Acknowledgements

Thanks are due to the members of Heriot-Watt University's SCHOLAR team who planned and created these materials, and to the many colleagues who reviewed the content.

We would like to acknowledge the assistance of the education authorities, colleges, teachers and students who contributed to the SCHOLAR programme and who evaluated these materials.

Grateful acknowledgement is made for permission to use the following material in the SCHOLAR programme:

The Scottish Qualifications Authority for permission to use Past Papers assessments.

The Scottish Government for financial support.

Contents

Topic 1

Software Development Process

Contents

Learning Objectives

After studying this topic, you should be able to:

- *describe the progression through project proposal, feasibility study (economic, legal, technical, time), operational requirements document (ORD) and specification, design, implementation, testing, evaluation and maintenance.*

1.1 Introduction

This topic builds on the work done at Higher Grade Computing in the Software Development Process. We revisit the various phases of the software development in more detail and focus on the progression of a project from its initial proposal right through to its final implementation on the client site.

1.2 Review

You will recall from the Higher Computing course that the traditional software development process, referred to as the waterfall model, is an iterative process and contains a sequence of well-defined stages. A study of the literature on this subject might reveal various adaptations of these stages; for our purposes the following list is representative of the processes involved:

1. Analysis;
2. Design;
3. Implementation;
4. Testing;
5. Documentation;
6. Evaluation;
7. Maintenance.

The historical notion is that the development flows down through the stages like water down a fall, or at least a fish ladder. The method's sometimes known as the throw-it-over-the-wall method, on the grounds that the person responsible for one stage throws his completed work over the wall and thinks no more about it.

The problem is that things don't work out that way. Mistakes made at one stage of the process sometimes don't become apparent until later on. In the worst case, a flaw in the analysis of the operational requirements might not become obvious until the system's been installed.

If a mistake in the system later becomes apparent, the stage where it was made and all later stages of the process need to be revised. The waterfall becomes more like a system of locks on a canal, with water being pumped back up to the higher locks. Figure 1.1 represents this:

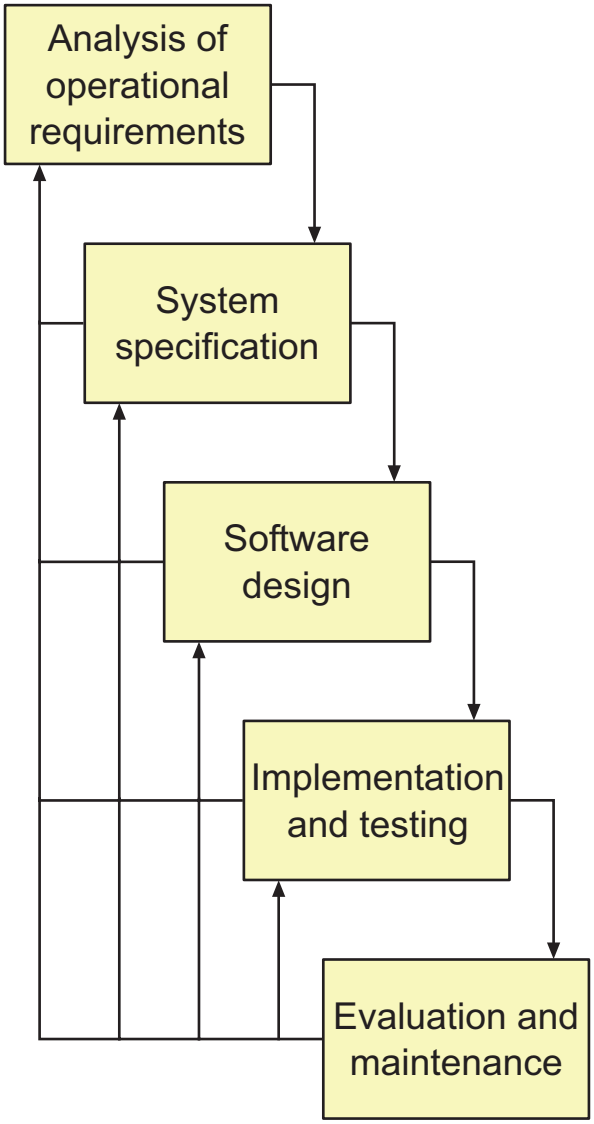


Figure 1.1: Iterative waterfall model

Although it has some shortcomings , the waterfall model is still widely used, especially when projects don't contain a great deal of innovation. It provides a structure that the development can follow. From the point of view of theory, it's the basic method of software development against which other methods can be compared.

1.3 Project progression and scope

For the Advanced Higher course we want to focus on some of the finer details of the software development process and take forward some of the earlier ideas while introducing others.

In particular we want to describe the progression of a project through the following stages as shown in Table 1.1:

Table 1.1: Progression through development

1	Project proposal	What is involved and how long should this take in the overall cycle of events? Who is responsible for this?
2	Feasibility study	Is there a valid case for designing a new or updated system? What are the objectives? Is the old system failing to meet company requirements?
3	Analysis of the problem	What are the new system requirements? What are the costs involved? How long will it take to implement? What is 'The operational requirements document' (ORD)?
4	System design	What type of hardware, networked or otherwise, software, inputs, outputs, HCI, training schedules. All input/output shown with screen designs.
5	Implementation and testing	Coding and testing of the new system. All program listings with pseudo-code and expected outcomes documented.
6	Evaluation and maintenance	Does the new system meet all the requirements outlined in the specifications? Maintenance schedules should be included to take into account unforeseen problems such as coding errors, missing files or lack of hardware compliance.

The scope of Projects can vary in size, complexity and cost from fairly small, local developments to large multi-national company contracts involving global interaction.

For example they may:

- Involve changes to existing systems.
- Entail organisational changes.
- Involve a single person or many people.
- Involve a single section of an organisation, or may cross organisational boundaries.
- Cost anywhere from £100 to many £millions
- Require less than a full day's work or take several years to complete

Some Examples are given in Table 1.2:

Table 1.2: Scale of projects

Small scale project	Medium scale project	Large scale project
Student computing project. Installation of home computing system. Computerising a small GP practice	Computerisation of a small library with web access. Updating of office filing system from manual to computerised College extending LAN to WAN	Multi-national company updating global network and hardware. Supermarket chain installing new OS and terminals in all stores nationwide Software house implementing a major upgrade, globally.
Computer systems: 1 - 10	Computer systems: 20 - 600	Computer systems: 10,000+

You will have probably been involved in some project or another at this stage, or are busy contemplating one. You might also be aware that hardly a week goes past without some software venture being reported in the media for good or ill. Computing disasters were covered in the Higher Computing course and yet, such events are still making news, as seen in the following government issues:

- June 2004 - thousands of people were stranded at airports across the UK after a software crash hit the flight data processing system at West Drayton;
- August 2004 - local education authorities had problems processing student loans because of ongoing computer problems;
- October 2004 - one million cases were stuck in the Child Support Agency computers due to long term problems in the £500 million system;
- November 2004 - department of Work and Pensions PC network crashed for several days denying payment to thousands of people. This occurred because a software update was wrongly sent to 80,000 PCs.

However, not all news is negative! Interesting computing developments also feature regularly and the following are representative as of 2004/2005:

- ASDA the retail store is following the lead of its American parent company, Wal-Mart in announcing plans to introduce RFID (Radio Frequency IDentity) tags in their supply chain. This would abolish the need for barcode labelling of goods. Tesco and Marks and Spencer are already well advanced in trials which will see a multi-million pound replacement of present hardware and software systems.
- One of the UK largest IT projects is the NHS National Programme. This will bring modern computer systems into the NHS and will connect over 30,000 GPs in England to almost 300 hospitals.
- The NHS has also joined forces with Microsoft who are investing £40 million to develop computing resources to improve clinical care. Also Microsoft will allow the

NHS to use up to 900,000 software licences on a perpetual basis, saving the NHS an estimated £330 million

There is quite a debate about the use of RFID. You may wish to consult the following link to find out more information:

http://www.vnunet.com/features/1160085

Exercise

Review press, computing journals or the internet and list around five recent computing projects that have been developed or are in the process of being developed. If using the internet some search engines offer a 'News' option as well as the 'web' option.

1.4 Project proposal

The process starts with a perceived need that will originate from a **client**. The client may be an individual but in all probability it will be a business or company that requires changes to be made, either in their existing system or by installing a new one. In most cases, within the client organisation, **management** will be responsible for all decisions regarding the installation of a new system: this can be a major undertaking and be a costly and time-consuming process.

Important considerations to be addressed at this stage might be:

1. Current system may be too old and expensive to maintain. The cost benefits have now diminished to such an extent that the system is no longer viable.
2. Present system is so outdated that it can no longer respond to customer demands. It has fallen behind in terms of technological advances. Business change with the times and have to adapt to the market place.
3. Current premises inadequate to house and run present system effectively.
4. Dissatisfaction within the business of personnel using current system.

Problems with an existing system in an organisation can come to light from a variety of sources and for a variety of reasons. Table 1.3 gives some examples of the problems that might arise from within the organisation:

Table 1.3: Problems with existing systems

End users	• too much paperwork • errors in system • system too slow • bottlenecks
Middle management	• difficulty getting reports from system • problems interacting with other systems • not meeting targets • complaints from customers
Senior management	• system too labour intensive • system too costly • system causing poor service, compared with competitors

Once the problem has been recognised, a decision has to be made whether to do anything about it. If the problem seems serious enough, it is usual for somebody in the organisation, usually a manager, to draw up a report that will define the problem which is referred to as the problem definition or **project proposal**. This will outline the project in terms of the scope and objectives that need to be attained if the problem is to be solved.

At this stage the objectives of the project may be totally unrealistic.

The project proposal will convert an idea or policy into the details of a potential project, including the outcomes, outputs, major risks, costs, and an estimate of the resources and time required to complete. This information is contained in a formal proposal document.

For large projects a proposal template might be used for the company to complete and send to competing organisations for tender. This will be a pro forma report that will help the company to articulate their requirements in a clear and unambiguous manner. It will be completed by a member of the management team or other responsible person nominated by the organisation and he/she will be the key contact during this phase of the enquiry.

The proposal document should include a full company profile and list of project requirements with options for additional services if required. These could include, for example, migration routes from the old system to the new, conversion of company data files (legacy files) to run on the new syste, preferred dates for installation and whether training is to be onsite or remote.

An example cover page might look like that shown in Figure 1.2.

Figure 1.2: Proposal document cover page

Further information contained within the document will cover the major points:

- Project outline: a concise summary of the main objectives.
- Project type: small, medium or large.
- Project justification: how did the idea originate and what will it achieve?
- Budget constraints: is sufficient funding readily available (including contingencies).
- Risk factor analyses: what risks are involved, what are the risks of failure?

The company will then receive tenders for the proposed project where the main issues will be addressed and costed. Depending on the company, a time scale for each

phase of the project might also be included together with an indication of the number of deployed staff, essential resources and a summary of risk assessments for each phase of the project.

A typical page from the reply document is shown in Figure 1.3.

Project Proposal
by
O.K. Marketing Ltd

Summary Page

Stages	Start Date	End Date	Man Days
Analysis	20/02/05		10
Design	24/03/05		30
Implementation	05/06/05		12
Testing	22/07/05		45
Documentation	15/09/05		20
Training	20/10/05		4
Total			121

Costs

Time and resources	Daily Rate	Total
100 days	£2000	£200,000
Fixed Fee	£5000	£205,000

Page 18 of 20

Figure 1.3: Example reply page

Should the matter be taken further then management will call in **consultants** with a view to commissioning a **feasibility study** which represents the first formal stage of the software development cycle. Many large organisations, where the need for new systems almost never stops, have specialist departments for systems analysis and software development. In the majority of cases, consultants come from an independent company.

1.4.1 Review questions

Q1: Explain what is meant by the scope of a software project.

Q2: Management of a company are concerned about the conversion of their legacy files to the new system. Give two examples of what might be considered legacy files.

Q3: What is a project proposal document? Outline three items that company management might include in a project proposal document.

1.5 Feasibility study

You will now realise that the creation of a new system can take months or even years, and can cost thousands or even millions of pounds. The basic purpose of a feasibility study is to ascertain whether such expenditure of time and money is likely to prove worthwhile and whether the objectives of the problem definition can be realised: aspirations of some companies might be totally unrealistic

The results of this study will determine which, if any, of a number of possible solutions can be further developed at the design phase.

One result of undertaking a feasibility study may indicate that only a simple solution is required to solve the problem:

- a software fault may be identified that is easily fixed.
- more staff training might be required if particular software is to be used.

The feasibility study is complete in itself. It should be conducted relatively cheaply and within a fairly short time frame. There are no legal or contractual requirements at this stage and neither should there be any committal to further expenditure nor any long term development.

1.5.1 Who carries out the feasibility study?

Approaches vary from organisation to organisation. Some large organisations have a committee whose task it is to evaluate new systems. In others, it is customary to select a working party, made up of management personnel, system analysts and other employees to look at the problem in question. Other organisations might give the responsibility to a project director who would be a manager or analyst on the staff.

In most cases the person responsible for undertaking the feasibility study is the **project leader**, usually an experienced member of staff appointed by the team of consultants.

1.5.2 Feasibility criteria

An effective solution to a project may well be determined by operational constraints within an organisation. These constraints must be taken into account when commissioning a feasibility study. These can be classified as:

1. Technical feasibility
2. Economic feasibility
3. Legal feasibility
4. Schedule feasibility

1.5.3 Technical feasibility

The feasibility study must ascertain what technologies are necessary for the proposed system to work as it should. It may be the case that suitably advanced technology does not yet exist. Unless it is the object of the project to design a system to use such advanced technology, this would rule the project out as being a non-starter. It would be a foolish move for a feasibility study to evaluate technologies which are either under development or undergoing testing.

Given that suitable technology does exist, the study must establish if the organisation already has the necessary resources. If not, the study must make clear what new resources the organisation would have to acquire. This will also involve determining whether the hardware and software recommended will operate effectively under the proposed workload and in the proposed environmental conditions.

The development of a new system involves risks of one kind or another. Every understanding that might be reached could carry the risk of some misunderstanding:

- software companies and their clients often have different vocabularies and consequently they appear to be in perfect agreement until the finished product is supplied.
- management may have unrealistic visions of computer systems. The feasibility study is where idealism meets reality.

Further issues might include the training of personnel to use the new system, consideration of service contracts, warranty conditions and the establishing of help desk facilities for inexperienced users.

1.5.4 Economic feasibility

This deals with the cost implications involved. Management will want to know how much each option will cost, what is affordable within the company's budget and what they get for their money. A **cost-benefit-analysis** is part of the budgetary feasibility study. If the project is not cost-effective then there is no point proceeding.

Setting up a new computer system is an investment and involves capital outlay. The costs of a new system include the costs of acquiring it in the first place (consultancy fees, program development, etc.); the costs of installing it (disruption of current operations, cost of new equipment, alteration of workplace, etc.); and the costs of maintaining it which also includes training.

In the long term management will also want to know the '**break-even point**' when the new system stops costing money and starts to make money. This is extremely difficult to quantify. However an accurate estimate of a system's operational life span is a valid option and will rely solely on the knowledge and experience of the systems analyst involved.

Figure 1.4 depicts such a case where the break-even point is at the intersection of the graphs:

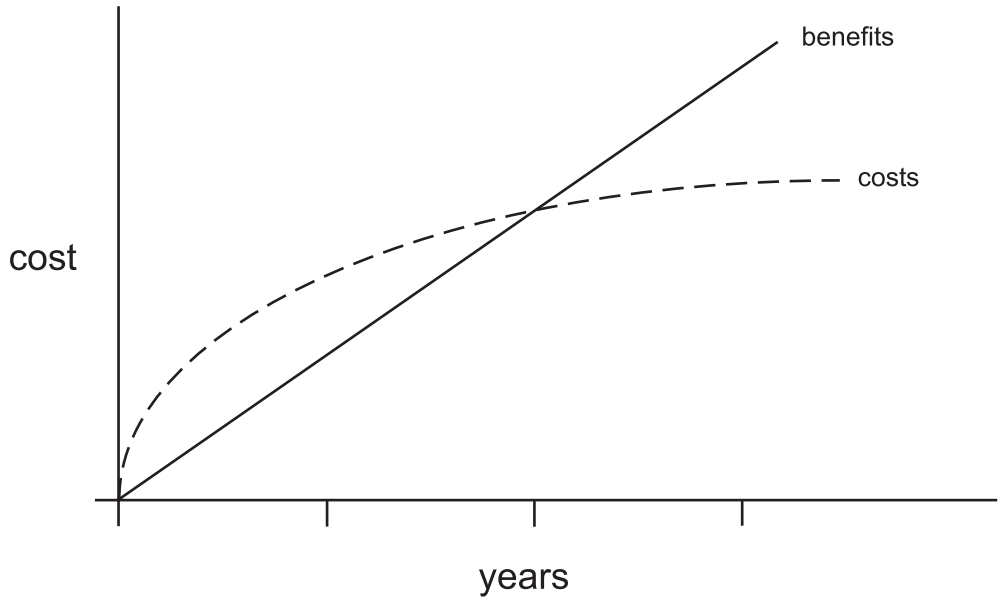


Figure 1.4: Break-even point

Tangible benefits that management would certainly be looking for in the new system would be:

- reduced running costs
- increased operational speed
- increased throughput of work
- better reporting facilities

Note that not all the costs and benefits lend themselves to direct measurement. For example, new systems generally affect the morale of the staff involved, for good or ill. This can only be resolved by competent personnel management practices.

1.5.5 Legal feasibility

This has to do with any conflicts that might arise between the proposed system and legal requirements: how would the new system affect contracts and liability, are health and safety issues in place and would the system be legal under such local laws as the UK Data Protection Act? What are the software licensing implications for the new system?

Software licensing can be quite a thorny problem. Licences can be purchased as: client licence (per seat), server licence, network licence or site licence and the period of operation may be annual or perpetual. Software vendors vary in their licensing regulations so this has to be fully investigated.

1.5.6 Schedule feasibility

Schedule feasibility may be assessed as part of technical feasibility. Most organisations have an annual schedule of events such as the AGM, end of financial year, main holiday period and so on. Obviously time is a main factor in the development of a new system.

Questions to be asked at this stage might include:

- how long will the proposed system take to develop?
- will it be ready within the specified time-frame?
- when is the best time to install?

For example, a project might have to start within six months; assuming it would take three months to purchase and install the required hardware and software and a further six months to train the end users. Such a project is not technically feasible because of shortage of time so it would not go ahead unless some of the time constraints were reviewed and changed.

In many cases of project management the scheduling component can be aided by means of a Gannt chart.

A Gantt chart is like a horizontal bar graph used to plan and schedule projects involving several concurrent tasks. The horizontal axis represents the time scale and start and finish times of component parts are graphically represented.

The advantage of a Gantt chart is that it shows, at a glance, the progress of a project as shown in Figure 1.5.

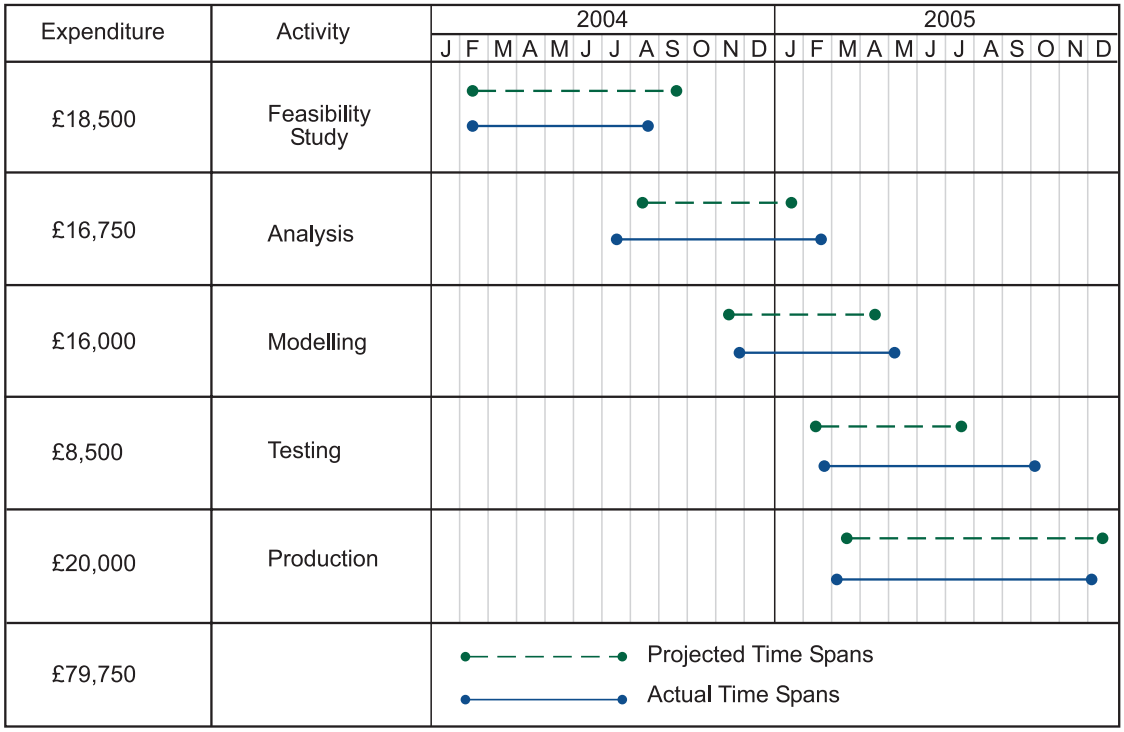


Figure 1.5: Example of a Gannt chart

1.5.7 The feasibility study report

The final step in the feasibility study is the production of a report for the client company. This will contain a summary of the findings by the consultants in terms of possible solutions to the problem with expert guidance and alternative options to consider. Also included will be proposed project schedules, target dates and cost options with recommendations to management based on the current system.

If the project could be implemented then management must decide if it should be implemented. All systems involve costs as well as benefits, tangible or otherwise. As a general rule management like to feel that benefits are likely to outweigh costs before proceeding with a project. Other issues that have to be addressed by management in relation to a new system might include social and political aspects:

- Will customer loyalty be retained?
- Will staff feel threatened?
- Will a move to larger premises be justified?

If management decides that the project should not go ahead then the matter rests: the feasibility study report is shelved pending future reconsideration, or a different firm of consultants are hired.

Time taken at this stage is well worth the effort. The decision by any company to abandon a project at the feasibility phase should not be seen as a failure but as a sensible business outcome!

If, however, the feasibility report is favourable and the project gets the go-ahead, the next phase of the software development process is entered, namely a full **system investigation**.

Identifying Advantages of a Feasibility Study

An online interactivity is available at this point.

1.5.8 Review questions

Q4: In the context of systems development, explain what is meant by a feasibility study and who carries it out?

Q5: Identify **four** kinds of feasibibilty and describe their differences.

Q6: Why is it important that a feasibility study should generate a report?

Q7: Explain what is meant by the term *cost-benefit analysis*.

Q8: What is the purpose of a Gannt chart?

1.6 The system investigation

The purpose of the system investigation is to determine, in as much detail as is possible in advance, what has to be done to solve the problem. Either an existing system will have to be analysed and a new one designed, or a new system will have to be created from scratch.

If a system (not necessarily computerised) is already in place, the work it carries out is investigated. This investigation forms the basis for the design of the new system. We could take as an example a small firm that has always done its accounts by hand. Its accounting system would be analysed. The new system would be based on this analysis, so that it performs at least as well as the old one. Of course, the new system should also offer benefits to the user: it should be easier to use than the old system, or offer functions that the old system could not carry out.

When an organisation is thinking of something new, there is often not a system in place. For example, a company might want to start making a new kind of product which will need to be made using new machinery. In such a case, there is no current system to investigate and the new system will have to be created on some other basis.

The investigation will be carried out by another representative of the consultants: a **systems analyst**. In the case of a large system, there might be a team of systems analysts. The systems analyst will report to the project leader.

The analyst's work is undertaken in two stages:

1. completion of a full system investigation.
2. production of an **operational requirements document**.

The analyst must carry out detailed observation of the operations of the company by interviewing relevant people and observing how the current system operates. If the system is large, it may not be possible to interview or observe everything fully within the time frame, and the investigators may have to use sampling techniques for certain parts of the system. For example, the boundaries of the system have to be established early on in the analysis and this will include; the system inputs, the processing it must do and the outputs that the program is expected to produce.

This task might, at first glance, seem to be fairly straightforward. However investigators have to wary:

- people fear change, on the grounds - not necessarily justifiable - that it's usually for the worse.
- it may be the case that employees feel threatened by the prospect of a new computer system being brought in. If they are currently doing things by hand, they may feel that the new computers will pose a threat to their jobs. Even if assured that no jobs will be lost, they may be anxious about learning new procedures and be afraid that they won't be able to cope with the technology.
- an analyst coming in to such an environment may find it difficult to gain the co-operation necessary to do the job properly.

Eventually, however the investigators must document all system operations. They must note what functionality is required and what the data processing requirements are.

When the investigation is completed, it will be summarised in an operational requirements document that describes what the system must do. This document is sometimes referred to as the **functional specification**.

1.6.1 The operational requirements document (ORD)

The operational requirements document contains a full description of the problem with all inputs, processes and outputs. The document is used throughout the remainder of the development process. For example, test data will be drawn up on the basis of the information contained in the document.

The management of the client organisation will review the operational requirements document. The process of reaching an agreement will be iterative: there often follows discussions with the consultants regarding amendments to the specification. When clients and consultants are agreed, a final copy, a formal operational requirements document is produced.

This document will then stand as a **legally binding contract** between the clients and the consultants. It serves to protect both parties. The consultants must produce a system to meet the specification, so it serves to protect the clients. On the other hand, any additions to the specification that the clients might think up will be regarded as alterations to the original contract that will incur additional costs, so it serves to protect the consultants.

The operational requirements document can also be used by the clients to validate any proposed design offered in the software development stage. Discrepancies and differences of interpretation can then be sorted out before more detailed development has been carried out (and that development time wasted).

The operational requirements document will be used by the consultants as a working document that will inform everyone involved in the development process.

Having gathered as much information as possible, the analyst then has to make sense of it. The process will be iterative: analysis of the answers to questions usually leads to more questions and answers.

Even at this stage there are many problems for the analyst to overcome:

- People (especially management) don't always know what they want of a new system. When they do know, they can't always express it clearly.
- Even when expressed clearly, people's requirements can involve specialised knowledge that the analyst does not yet possess.
- Different people want different features in a system. Interests can conflict.

The analyst then goes on to write the specification of the operational requirements. This will be the basis of the contract for the system. It should be written as clearly as possible and contain no ambiguities. This is by no means easy.

Many attempts have been made to create formal, unambiguous languages for writing operational requirements. Specifications written in these have not always been

welcomed by clients, who find them hard to understand and are reluctant to accept them as contracts.

However misunderstandings between customer and client arising from uncertainty in the operational requirements document can lead to any amount of ill feeling and possible litigation, resulting in costly outcomes.

When completed, the operational requirements document will contain a functional specification, a physical specification, a note on data requirements, and a system prospectus.

- The functional specification will detail what the system will actually do.
- The physical specification will list the hardware needed.
- The data requirements will declare what storage capacity is required.
- The system prospectus will contain a project schedule, and details of the user documentation and training needed.

The operational requirements document marks the end of what might be called the investigative phase of the project. When it has been agreed, work passes to the development phase.

1.6.2 System design

The project now enters the development phase, the first part of which is system design. The components described in the operational requirements document are brought together into a coherent whole. Flows of data from one component to another are clarified. Each component is further refined, usually from the top down, level by level using such methodologies as

- structure charts
- data flow diagrams
- Jackson structured programming
- pseudocode

until the design is at a low enough level for putting into code.

Design is a creative process. Although methods and guidelines are helpful, judgement and flair on the part of the software developers are still required to design a software system.

The final design produced usually results from the iteration of a number of preliminary designs with increasing formality, as illustrated in Figure 1.6.

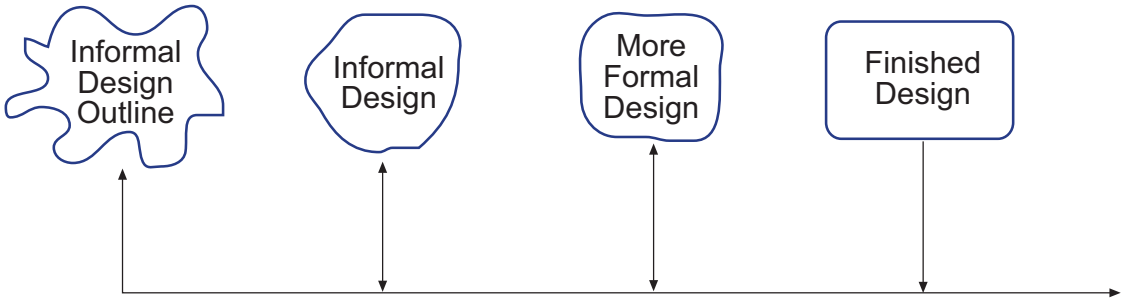


Figure 1.6: The Process of Design

The design process is usually hierarchical (modular) and results in a chart where the nodes represent the components of the design, as shown in Figure 1.7. The aim of this process is to create a system with no inconsistencies and no illegal relationships existing between the components.

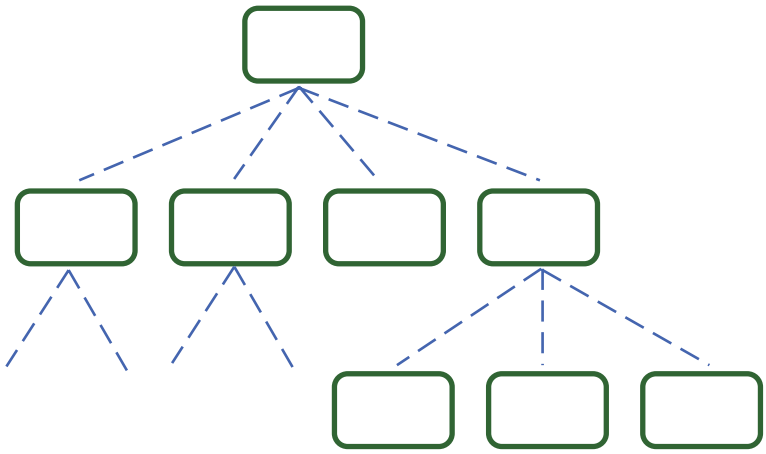


Figure 1.7: Modular Top Down Approach to Design

The design process iterates between design and specification and any errors or omissions found from earlier stages of the process are fed back and corrected. This is important to make sure that the system being built is the one that was asked for. Some analysts call this process of checking that the product matches the specification, **verification**, as distinct from **validation**, whose purpose is to check that the product is what the customer wants. Other analysts use validation to describe both checking the product against its specification and confirming that it meets the customer's requirements.

Specification is essentially describing *what* is to be implemented, whereas the design describes *how* it is to be implemented.

Once a successful design has been formulated the next stage of the development process, namely **implementation**, begins.

1.6.3 Review questions

Q9: What is the purpose of the system investigation and who carries this out?

Q10: State **two** disadvantages of interviewing personnel as a fact finding method.

Q11: What is an operational requirements document (ORD) and what is its significance?

Q12: What is the purpose of the functional specification?

Q13: What is meant by the term structured design in the software development process?

1.6.4 Implementation and testing

This phase of the system development lifecycle can be the crucial part of the project. This is where the theory of analysis and design is put into practice and often problems not previously encountered are discovered. However, by giving enough consideration to the implementation phase, problems may be anticipated and unavoidable problems managed.

Tasks to be carried out during implementation will include:

- Coding and testing of the system;
- Human Computer Iinterfaces fully designed and tested;
- Setting up hardware on site;
- Legacy files converted to correct formats and media;
- Manuals and documentation completed and available;
- Staff trained.

Here, the actual coding takes place in a language suited to solving the problem and success here will be attributed to the expertise of the programming team.

It's important that the design is tested and that the testing, to be of any value, has to be systematic.

Dry runs and structured walkthroughs are used to test individual modules. Test data is designed using the following parameters: the input, which is often called the test case; the reason for choosing that test case; and the expected result. A set of such cases is designed for each module. The designer then runs the test cases through the pseudocode to check, as far as possible, that the logic is correct. (The same test data will be used later, to test the actual code.)

Top-down testing goes with top-down development. In top-down testing, a module is tested as soon as it is coded. Coding and testing can be regarded as a single activity, the purpose of which is to ensure that the detailed logic of the module is correct, but it can also reveal design errors in the overall structure.

With large applications, things become a little more complicated. The approved approach is still top-down. However, the system has to be divided into sub-systems that are developed separately. What happens, then, is that modules are created and tested within their sub-systems. When all modules have been tested, the sub-system as a whole is tested. Then the sub-systems are brought together, or integrated, and the complete system is tested.

Test Strategies

Two test strategies that are usually implemented at different stages of the testing process are **black box testing** and **white box testing**.

In black box testing (Figure 1.8), sometimes called functional testing, the program specification is used as the basis for the testing procedures independent of any knowledge of the code, hence the term black box. Test data will be used to cover all inputs and outputs within the scope of the program.

Figure 1.8: Black box

White box testing (Figure 1.9), sometimes called structural testing, tests the actual code structure rather than its function. In this case a knowledge of how the code works is essential and accurate test results will rely on the testers' understanding of what the program is supposed to do. White box testing precedes black box testing since the source code is produced late in the development process.

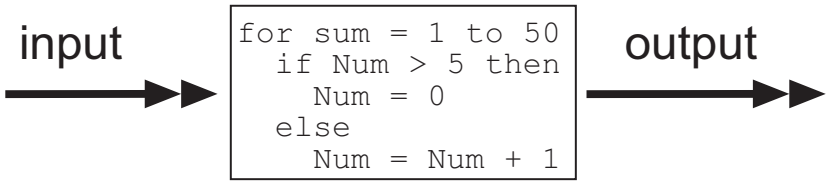


Figure 1.9: White box

In order to fully test a software project both black and white box testing are required.

Finally, the system is installed on the client's site, and is subjected to what is called acceptance testing: if it passes, it's accepted.

The implementation phase will also see the acquisition of the necessary hardware required to support the new system. This could range from a simple conversion of an already existing system to the commissioning of a completely new system installed in a purpose-built site. If networked then this would involve cabling strategies (copper, fibre) or may include wireless connectivity.

The next aspect to be considered is the changeover strategy from the old system, should one exist, to the new. There are several approaches that can be adopted from the conversion options: *Parallel*, *Phased*, *Pilot*, *Direct* and *Combination*.

Parallel conversion involves running the new system in parallel with the old one for a period of time ranging from a few weeks to many months, depending on the complexity of the system. An advantage here is that the old system can serve as backup should anything go wrong and time is given to the users to become familiar with the new system. An obvious disadvantage is the duplication of effort in processing duplicate data on the two systems.

Phased conversion is similar to parallel implementation, but is carried out in smaller, more manageable sections. Here the new system gradually takes over thereby keeping

risks to a minimum and focusing on particular stages at any one time. However a phased implementation can take longer to achieve and problems may arise when controlling old and new phases as separate entities.

Pilot conversion is a fairly safe option since it would either be a retrospective trial on data previously processed by the old system or be restricted to a particular section of the old system and tested. If successful then a changeover can be implemented in either approach. Pilot implementation will also allow a number of trials to be conducted until an acceptable version are ready. The advantages of pilot implementation are that the disruption to the system is minimal and there is less pressure for success.

Direct conversion is the 'big bang' approach with immediate conversion from old to new. It is usually the most practical solution and attracts the minimum costs and disruption. However the system and the users must be fully prepared and the changeover done at a time when the workloads are at a minimum. The greatest risk is that in the event of failure, the old system is not available for backup.

Combined conversion involves a mixture of approaches that are often appropriate. For example, some parts of a system may be changed over directly without too many problems, whereas others may have to be tested by piloting. This approach depends entirely on the specific application and the existing system.

1.6.5 Evaluation and maintenance

When the new software system is up and running it is important to evaluate it in terms of running to the original specification.

Evaluations of various kinds are an important aspect of the software development industry.

Evaluations are used to determine if systems are usable, cost effective, conforming to performance criteria, etc. The basis of evaluation is in social science methods using techniques such as observation, interviews, and questionnaires. Additionally techniques such as automatic data logging are used. Many organisations bring in consultants who design and carry out evaluations as the skills required to carry out effective evaluations are highly specialised.

There is no limit to the number or type of criteria that are used in an evaluation. A very important aspect of planning evaluations is defining the criteria. Listed below are a number of evaluation criteria used in industry:

- the time it takes to install a piece of equipment;
- the number of errors an operator makes while doing a specific task;
- the time it takes an operator to complete a task;
- the number of times a computer crashes or hangs;
- the number of phone calls made to a help-line;
- the number of times a user consults a manual.

The key criterion in evaluating a software product has to be whether it is *fit for purpose* i.e. does it meet the original specification and allow the client to carry out their tasks?

Main questions that may be asked are:

- how closely does the solution match the specification?
- is the solution what the clients were looking for?

Other matters for review will include costs and schedules.

When the system has been installed and has passed its acceptance testing, maintenance comes into effect. Corrective maintenance may have to be carried out, to deal with errors that did not come to light during testing.

This, as a rule, is the most time consuming stage. Software does not wear out but it usually needs subsequent modification. Some bugs or design shortcomings only become apparent over time. In addition changes might have to be made to adapt the system to new demands.

Any enhancement to the system is also called maintenance. This may take the form of:

- installation of faster printers/fax machines requiring updated drivers
- use of more network servers
- extra hardware - interactive whiteboards, graphics tablets, wireless routers
- new network cabling topology

In terms of time, maintenance is often the longest part of any project, and the part that is least written about.

It can be tempting to add patches as they are required. However, changes should be made in an organised manner, having regard to the system as a whole and following good practice in software development.

Proper maintenance depends on accurate error reporting from users and a maintenance log should exist for recording such events.

There are different categories of software maintenance: *corrective*, *adaptive*, *perfective* and *preventive*.

Corrective maintenance is concerned with the repair of defects such as logic or coding errors found in the code after the beta testing stage.

Adaptive maintenance serves to modify the software should it be used in a new environment such as a new operating system for example.

Perfective maintenance deals with updating the software in response to user requests. User requirements may change because the voice network is to be integrated with the data network, for instance.

Preventive maintenance deals with updating the system documentation and changing the structure of the software to ease future maintainability.

Having progressed through the various phases of software development it is interesting to revisit the 'waterfall' diagram, only this time the percentage costs in relation to the overall development have been included. This puts the development phases into perspective as shown in Figure 1.10:

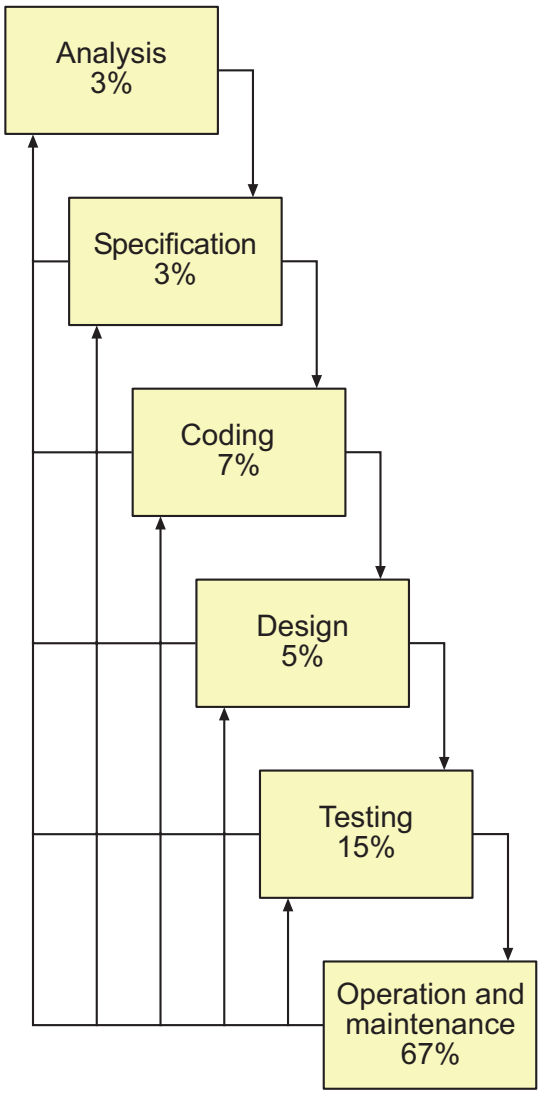


Figure 1.10: Relative costs of development phases as percentage of total

1.6.6 Review Questions

Q14: Outline **four** tasks that have to done during the implementation phase of software deveopment.

Q15: What is the difference between black box and white box testing startegies?

Q16: In the testing phase, test cases are often laid out in tables with columns. What are the four most common labelled columns used?

Q17: In the changeover from the old system to the new the following implementations exist:

1. Parallel
2. Pilot
3. Phased
4. Direct

Outline each method.

Q18: List two main questions that might be asked in an evaluation of a program as being *fit for purpose*.

1.7 Summary

This topic has covered details of the software development process, considering the progression of a project through the various phases.

You should now be aware of the following outcomes:

- the progression of a project from initial proposal to implementation and the methodologies used during the various stages;
- the significance of the feasibility study (economic, legal, technical and schedule);
- the significance of the operational requirements document (ORD).

End of topic test

Q19: The list below outlines the stages involved in the software development process.

A) Design of the system
B) Analysis of the requirements of the system
C) Definition of the problem
D) Maintenance of the system
E) Feasibility study
F) Collecting the information requirements of the system
G) Implementing and evaluating the system

Place them in the correct order.

Q20: Which one of the stages would involve testing the system?

Q21: Which stage would determine whether or not the system would be technically or economically worthwhile?

Q22: During which stage would the system specification be documented?

Q23: During which stage would user documentation be produced?

Q24: Which one of the following is a graphical notation used to describe the structure of software?

a) Outline diagram
b) Structure chart
c) Gannt chart
d) None of these

Q25: Pseudocode is a notation used to describe programs.

a) True
b) False

Q26: What does the term **test data** mean?

Q27: What type of testing splits a system into modules where outputs are checked against inputs for each?

a) Trace routine
b) White box
c) Maintenance
d) Black box

Q28: The type of maintenance that deals with updating a system in line with user requests is called _____ maintenance.

Topic 2

Interface Design

Contents

Learning Objectives

After studying this topic, you should be able to:

- *compare different user-interface design styles (menu driven, textual, graphical).*

This topic looks at interfaces and makes a comparison of the different user-interface designs that have developed throughout the computing age. The various types are discussed with emphasis on the relative advantages and disadvantages of each.

The "user interface" of a computer system is the component of the system which facilitates interaction between user and the system. Thus, the user interface must enable two-way communication by providing feedback to the user, as well as functions for entering data required by the system.

2.1 The need for a user interface

In the early days of computing, computers were large, expensive machines. Programs and data were fed into the computer on punched cards or paper tape, and programs ran in batch mode, with only an operator present, not the user. A program would either run to completion and produce the expected output, or it would fail, usually terminating prematurely, perhaps producing only incomplete results, or none at all with often obscure error message on the operator's console.

The only people who interacted with computers on a regular basis at this time were highly-trained engineers and scientists in research facilities. The cost and size of these computers made their wide-spread use impractical. At this time, communicating with the computer was a very complex task which required a detailed knowledge of the computer's hardware.

Advances made in technology allowed computers to be made smaller and affordable. As a result of this, and the increase in productivity afforded by computers, their use became more widespread.

With various people from diverse backgrounds now using computers in everyday life, came the need for a user-friendly interface through which the average person could interact productively with a computer system.

This led to the development of various types of user interfaces which catered for different types of users.

Nowadays, the vast majority of programs are interactive - the processor spends most of its time waiting for the user to input some data (usually from the keyboard) or to click a button or select from a menu, to indicate what the program should do next. Also the user interface is very often graphical. It includes, as well as representations of the application data or objects, graphically-represented controls such as buttons, menus, scroll-bars, forms, and dialogue boxes, collectively known as widgets.

2.2 Type of user interfaces

There a many different types of user interface currently in use. Each different type of interface can be characterised by the style of interaction it supports. The most common styles include:

- Command driven;
- Menu driven;
- Graphical user interfaces (GUI)
- Special purpose

Styles are chosen by interface designers to support particular tasks. Many systems have a mixture of different styles of interaction to support different tasks and subtasks.

2.3 Command line interfaces (CLI)

The command line interface was the first interactive user interface. It is derived from the teletypewriters (TTYs) that were used to communicate with mainframes. TTYs were notoriously prone to bottlenecks since commands were sent to the computer over relatively slow serial communication links and once there they had to be decoded. Thus to minimise the bottleneck, commands had to be short which led to very cryptic commands since every keystroke counted. Output was also limited since it was generated typewriter style, one character at a time.

This changed when the TTY gave way to the video display terminals (VDTs). These allowed a cursor to be located anywhere on the screen. This in turn allowed information to be printed anywhere on screen. With this capability of the VDT, a user with a few special keystrokes could enter information anywhere on the screen (within limits), and could go back and update the information or correct mistakes. The electronic VDT gave tremendous advantages over the paper based TTY.

Command-driven interfaces do not require much effort in the area of screen design and screen management since the user types everything into the system. A basic black or white background screen with white or black text is normally used. An example of a DOS screen is shown in Figure 2.1.

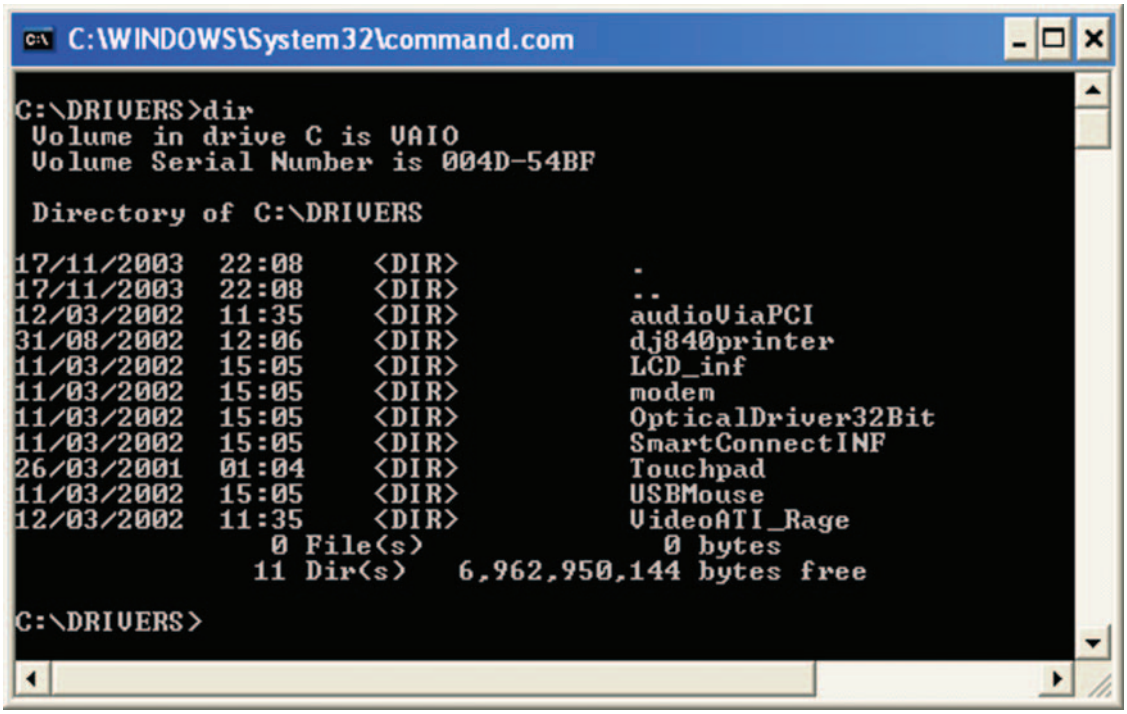


Figure 2.1: An example DOS screen

Early operating systems (and some still in use, such as DOS and UNIX) provided a command line user interface. Commands are entered at a command prompt and validated by the command language interpreter before they are carried out. These commands are usually words or meaningful mnemonics chosen to represent the nature of the activity. For example, in MS-DOS the command to perform a copy command is started with the word COPY. In UNIX type systems, the copy command is usually started with the characters cp. As these commands are usually entered from a keyboard, the designers of the system try to limit the amount of typing that the user has to perform.

Further examples of UNIX and DOS commands are shown below:

MS DOS

dir - display the contents of the current directory

cd - change directory

ren - rename a file

UNIX

ls - display the contents of the current directory

cd - change directory

mv - rename a file

Typical DOS commands would be:

```
C:\> Rename Myfile.bat Names.bat      {rename Myfile.bat to Names.bat}
C:\> Copy *.txt a:                    {copy all text files to the a:drive}
```

Command-driven interfaces must be equipped with error handling and message generation routines since users inevitably make mistakes in typing. Message generation

routines are also needed to alert the user to a change in the system's status. A help facility must also be provided to allow new users to the system to learn the commands of the system.

In DOS, for example, by typing 'Help' after the prompt, will display all the commands available to the user. Typing in 'Help' followed by a command will produce more information on that command.

Inexperienced users, however tend to be uncomfortable when using CLI systems, as the commands are not always easy to remember. Also, the error reporting from CLI systems tends to assume that the user has a certain amount of expertise, and can understand the somewhat cryptic messages that some systems produce when presented with unexpected or erroneous input.

For more experienced users, CLI based systems can allow complicated and sophisticated operations to be performed, and when combined with scripts (small programs written using the command language of the Command Line Interpreter) provide a convenient way of automating some of the more time-consuming tasks of administering a computer system.

A very popular operating system at the moment is Linux (derived from UNIX) which, at its most basic can provide a command line interface. It offers a selection of what are called shells; which surrounds the kernel of the operating system. Each shell has its set of commands and each command have a set of options or switches that can be used to shape its behaviour. Commands can be typed in directly and the shell acts as a command interpreter, executing the command if possible and sending an error message if not.

A series of commands can be brought together to make a shell program or script. A script can be a straightforward sequence of commands (to carry out, say, a sequence of actions that you often want the computer to perform) but it can also be much more. The scripting language is, in effect a programming language (much like a present-day scripting language used in Web page design). It provides high-level programming structures such as selection, repetition and modularity.

The shells and the scripting language offer the user precise control of the operating system. However, not everyone particularly wants to learn commands and switches and all the rest of it. So Linux also offers a selection of GUIs, rather than just one.

Advantages of command line interfaces:

- They may be implemented using cheap, alphanumeric displays.
- More experienced users prefer to have direct control over the operations of the computer.
- Commands of almost arbitrary complexity can be created by combining individual commands. This allows command macros and command language programs to be written.
- It is faster to issue a command sequence directly rather than searching through a menu.

Disadvantages of command interfaces:

- Users have to learn a command language which is sometimes complex. In some cases few users ever learn the complete language. The learning time for this type of interface is also greater than that for menu systems and GUIs. New users spend more time learning the interface than getting work done on the computer.
- Users always make mistakes in typing. This requires error handling and message generation facilities to be included in the command language processor. Systems interaction is through a keyboard. The interface cannot make use of pointing devices.
- For a novice it can be quite daunting and difficult to get even the simplest of operations performed.
- Commands across different operating systems are seldom the same. Users must become familiar with different command languages.

2.4 Review questions

Q1: Identify **two** situations where a command driven interface would be more appropriate than menu driven.

Q2: Outline **two** advantages and **two** disadvantages of a command driven interface.

Q3: Name **three** operating systems that you have come across that use command driven interfaces.

2.5 Menu driven interfaces

Menu driven interfaces present the user with a list of options from which to select. The user may make this selection via a keyboard or a pointing device such as a mouse. Selecting an option may initiate a command (such as 'save' or 'print') or may present the user with a sub-menu which has another list of options. These lower-level menus are said to be nested inside the menu that activates them. An example is shown in Figure 2.2.

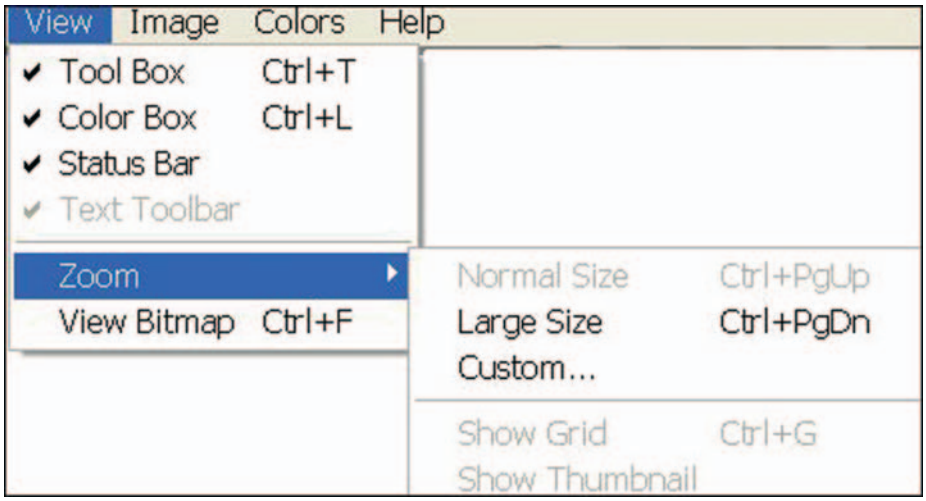


Figure 2.2: Nested menu system

There are three major categories into which menus can be divided:

1. Full screen menus
2. Bar and Pull-down Menus
3. Pop-up Menus

Full screen menus. These menus usually present the options to the user as a sequential list which occupies the entire screen. This is usually followed by a message prompting the user to select one of the options. In order to facilitate quick selection, the options may be numbered or a letter may be used to uniquely identify each option. In either case, the user selects an option by simply entering the corresponding number or letter.

The screen shot in Figure 2.3 shows a typical full screen menu example where an option is chosen using the mouse or cursor keys to highlight the choice :

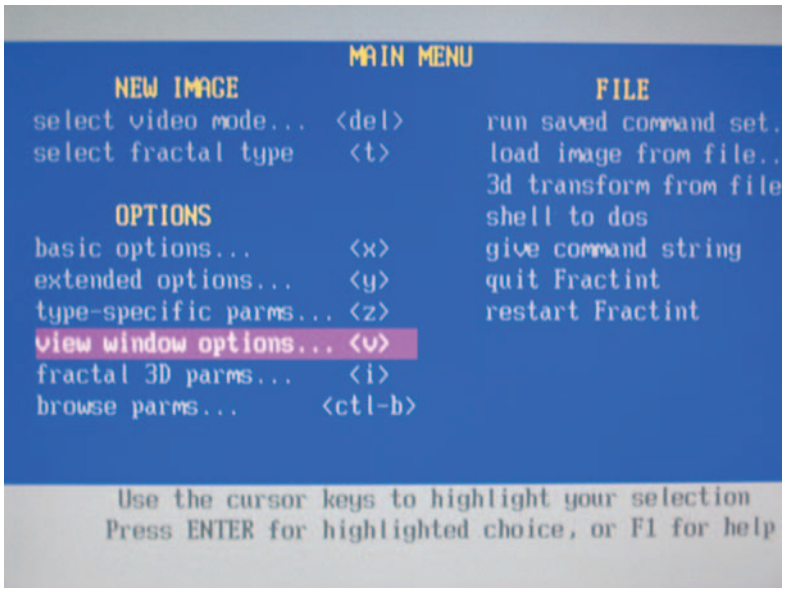


Figure 2.3: Full screen menu

Bar and pull-down menus. The main options available to the user are presented as pads on a horizontal bar across the screen. When the user selects one of the pads on this menu, the second-level options are displayed in a pull-down menu. This type of menu system is primarily used in conjunction with a pointing device. However, options may also be selected using 'short-cut' key combinations and arrow keys.

Consider the following screen shot as shown in Figure 2.4:

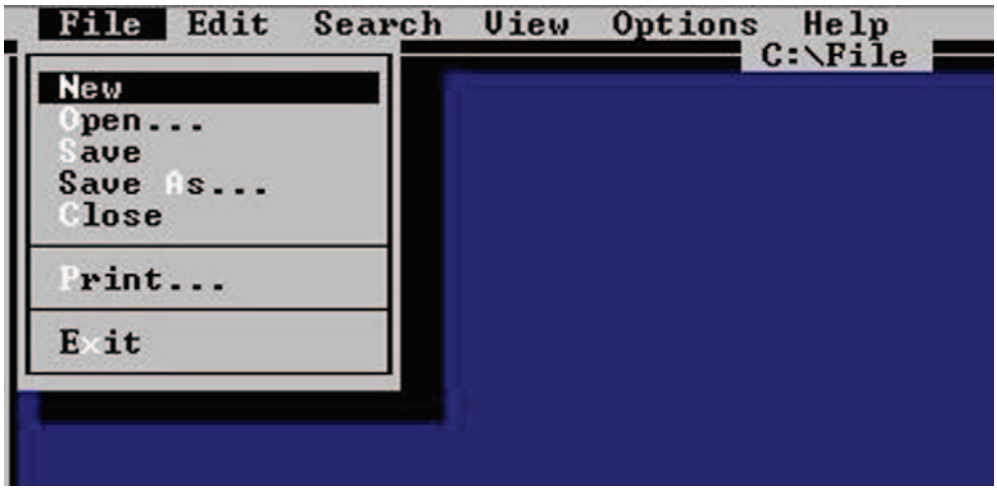


Figure 2.4: Drop-down menu

Initiation of any of the *File* commands may be achieved by placing the mouse pointer over the desired option and clicking once. Alternatively by using a 'hot key' combination (alt key + highlighted letter) the same effect can be produced (same semantics, different syntax) - only more quickly.

The following Table (Table 2.1) summarises some of these common 'hot key' commands.

Table 2.1: 'Hot key' commands

Alt + O	Open a saved file
Alt + S	Save file
Alt + P	Print file
Alt + X	Exit menu

Pop-up menus. These menus usually appear as a box with one of the options already selected. When the user points to the box with a mouse and presses the mouse button, the other options are displayed in a list. The user can then select the required option with the mouse. The menu options remain visible only while the mouse button is depressed. These menus are usually used in the task area of the application. If all the options cannot be displayed on the menu, the menu may scroll automatically or on command from the user.

The following example in Figure 2.5 illustrates a pop-up menu system:

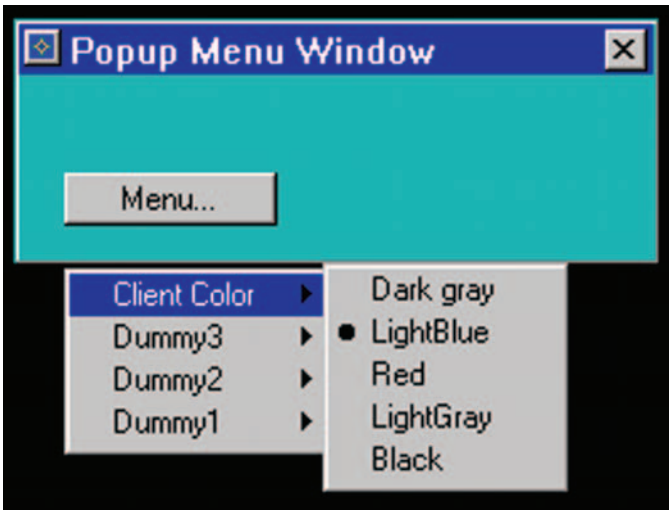


Figure 2.5: Pop-up menu

Menu driven interfaces tend to be favoured by inexperienced computer users who feel safer and more confident with the system taking more control of the interaction. Skilled computer users would prefer a command line system, as they feel constrained by having to navigate a hierarchical menu system to achieve an objective which could be achieved with a single command.

The advantages of menu systems are:

- The user is presented with a choice and so does not need to remember any commands. This interface is suitable for beginners and knowledgeable intermittent users.
- The opportunities for making serious errors are reduced since the user does not have to type in any commands.
- Menus can utilise special pointing devices such as mice, trackballs and light pens. Thus the typing effort required can be kept to a minimum.
- Context-dependent help can be provided since it is possible to keep track of the user's position in the menu system and link this to a help system.

Their disadvantages are:

- They are not as concise as command-driven interfaces. Accomplishing a task in a menu driven interface generally involves more steps than is required for doing the same task in a command-driven interface.
- If there are a large number of options available, it may difficult to structure the menu system so that the user is not presented with an unacceptably large menu.
- Tasks which involve logical operators (such as 'and', 'or') may be awkward to express in a menu system.

2.6 Graphical user interfaces (GUIs)

The first commercially successful GUI was the Apple Macintosh user interface, released in 1984, which also served as the Mac's operating system (OS). It evolved from an earlier GUI found on the LISA computer (also made by Apple), which did not survive. Some advanced users did not like this new interface because they preferred the CLI, however it did appeal to casual and inexperienced users because they could interact more easily with the computer.

The following screen in Figure 2.6 is from the early Lisa and MacIntosh GUI:

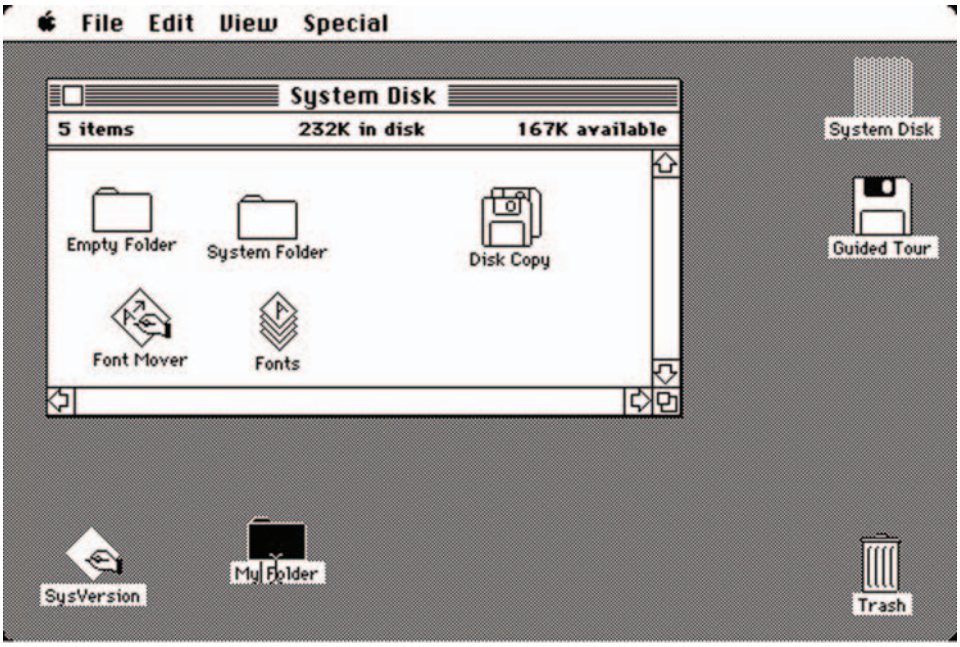


Figure 2.6: MacIntosh GUI

After the debut of the Mac Operating System (OS) a number of other GUIs appeared on the scene. The fundamental problem involved with implementing a GUI on IBM PCs was that the operating system, DOS, did not possess many of the necessary building blocks. As a result these resources had to be created and then piled on top of DOS. Because of this they tended to be slow and memory intensive. Despite these problems, there were a number of successful implementations. Chief of these were Quarterdeck's DESQView, Digital Research's GEM and Microsoft's Windows, which is still going through various transformations.

The main elements of a Graphical User Interface are as follows:

1. Windows
2. Icons
3. Menus
4. Pointers
5. Alerts and Warnings
6. Dialogue Boxes

The first four elements were responsible for the term 'WIMP' being applied to these types of interfaces. An example of a WIMP environment is shown in the following screen shot: (Figure 2.7):

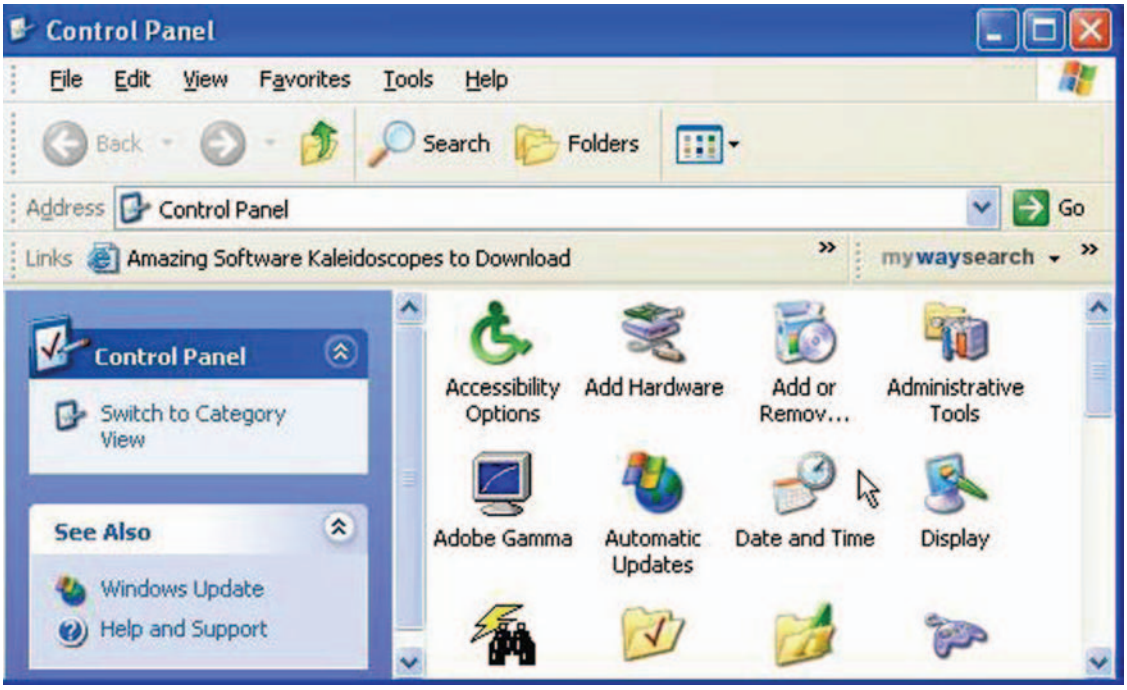


Figure 2.7: WIMP environment

Using a WIMP environment allowed the user to directly manipulate objects on the screen and perform operations such as saving files to disk, copying files or launching applications using drag, drop and click operations via the mouse.

2.7 Windows

A window is an interface component through which objects and actions are presented to the users. It is an area of the screen and is dedicated to a specific purpose. All messages, programs, icons and dialogs are contained in windows.

Windows may be tiled or overlapping. Tiled windows occupy a fixed area of the screen and no window can use the space occupied by another window. If one window is enlarged, all other windows are shrunk to maintain the tiled arrangement.

Overlapping windows do not occupy a fixed area of the screen. These can be moved around by the user at will. These windows can also be resized without affecting the surrounding windows. Overlapping windows can be obscured partially or wholly by other windows as shown in Figure 2.8.

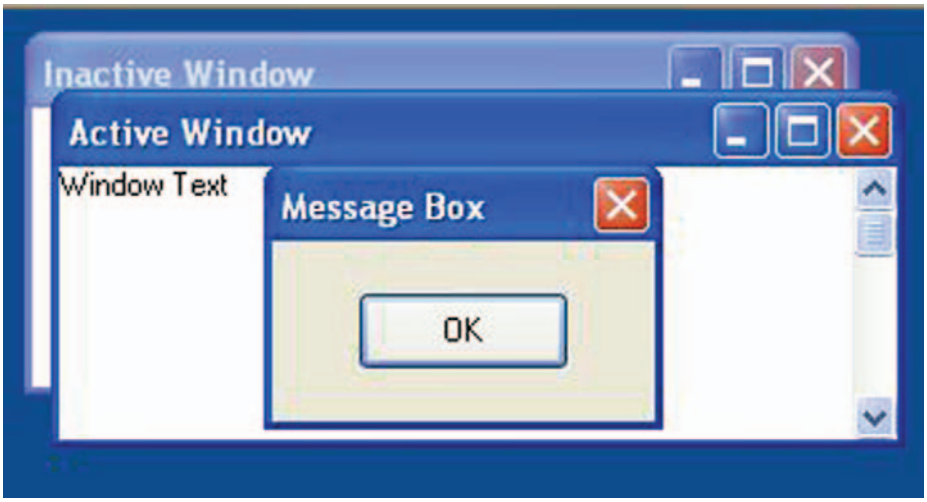


Figure 2.8: Example of overlapping windows

Overlapping windows are more flexible especially when large screen areas are unavailable.

Tiled windows, however, are more productive since the entire area of these can be viewed without obstruction.

Both tiled and overlapping windows may also be scrolling. When the physical size of a window does not allow all the elements within that window to be displayed, it becomes necessary to be able to move elements currently within the display area of the window out and move the undisplayed elements of the window into the display area. This process is known as scrolling and windows that have this capability are known as scrolling windows.

Figure 2.9 shows an example of tiled windows with three active applications; graphics, spreadsheet, Notepad:

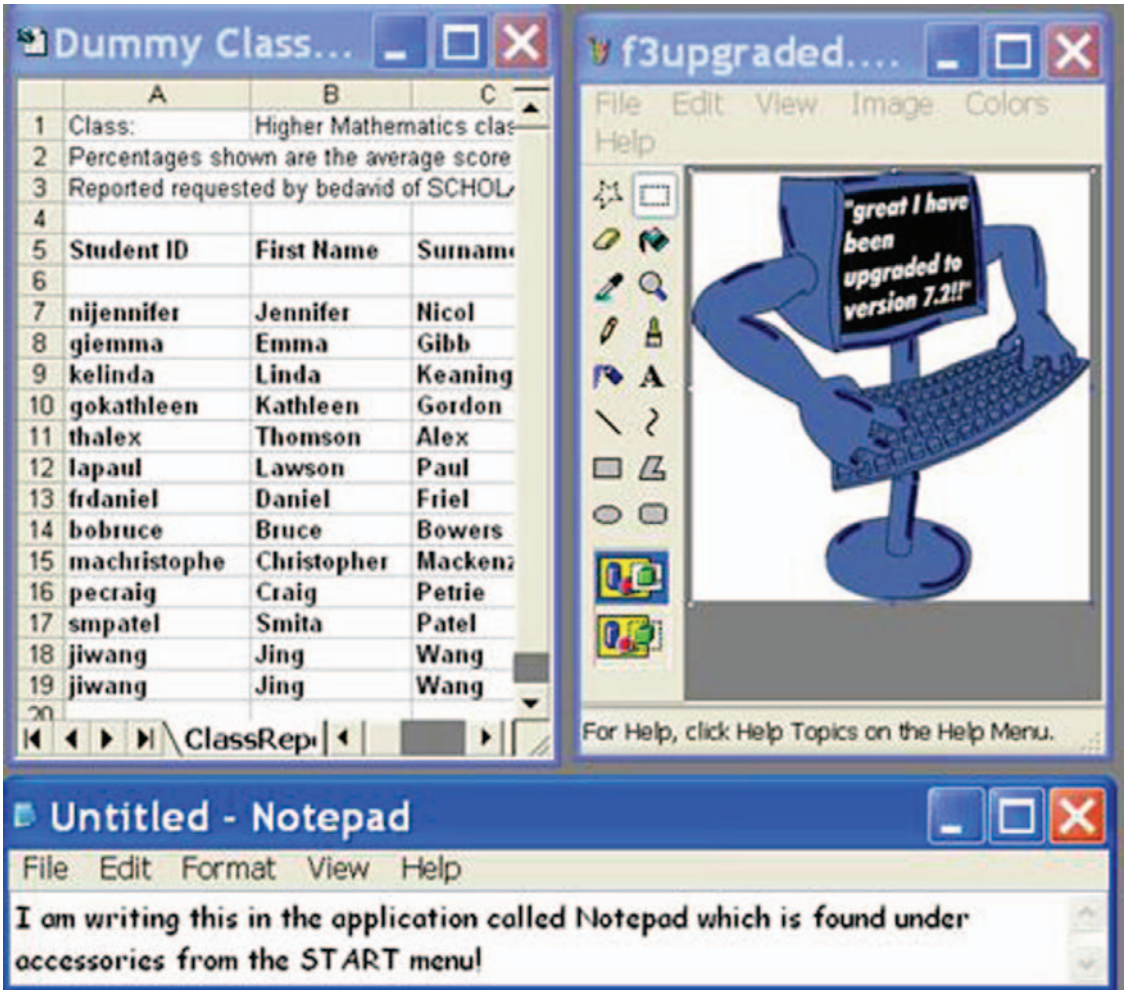


Figure 2.9: Tiled windows

The major design issue surrounding windows is whether to make them tiled, overlapping and whether or not to make them scrolling. This depends mainly on the application being developed. The common approach is to try to model the windows to be consistent with the rest of the interface.

2.8 Icons

An icon is a pictorial representation of an object or action. Icons can represent objects that users want to work on or actions that users want to perform. A unique icon also represents an application when it is minimised. Care must be taken when designing icons. The pictures must be carefully drawn so that they are understandable by users. The purpose of the icon must also be clear to users; hence great emphasis must be paid on the choice of picture used in the icon.

Figure 2.10 shows some icons representing actions in the Microsoft Word application. Many of the previous screenshots also show icons.

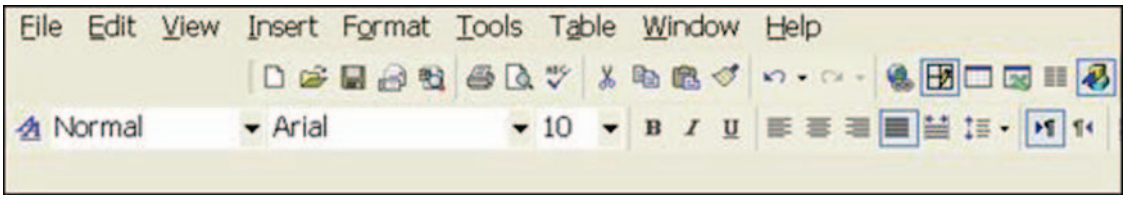


Figure 2.10: Examples of icons

2.9 Menus

Menus have been previously defined and discussed. The Graphical User Interface can support all types of menus. The more common types found in this interface are pull-down menus, menu bars, scrolling menus and pop-up menus. The pull-down menus of a Graphical User Interface can also be hierarchical. The same issues discussed in the design of menus in section 2.5 apply here as well

2.10 Pointers

A pointer is a symbol displayed on the screen that is controlled by a pointing device, such as a mouse. It is used to point at objects and actions users want to select. The pointer is the tool used to drive the GUI. Pointers are usually designed in the shape of an arrow to point to different selections. Pointers can also change shape. This is done to provide feedback to the user. For example, when a long operation is being performed, the pointer changes to an hour-glass or stopwatch to indicate to the user that the application is still functional but a specified operation will take some time to complete. When designing an interface to incorporate shape changeable pointers, emphasis must be paid on the shape of the pointer to suit the operation or mode the user is put into when the pointer changes shape.

Figure 2.11 shows examples of mouse pointers:

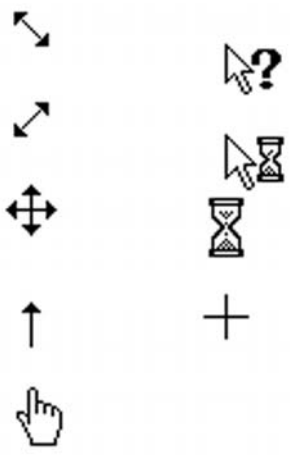


Figure 2.11: Mouse pointers

2.11 Alerts and Warnings

Alerts and warnings occur as pop-up windows that appear to notify the user of an event that requires a response. The user can choose between overriding the event (cancel or close) or by performing the required action.

Figure 2.12 shows an alert situation generated by closing an application. Figure 2.13 shows a warning produced by a notebook OS running on battery power:

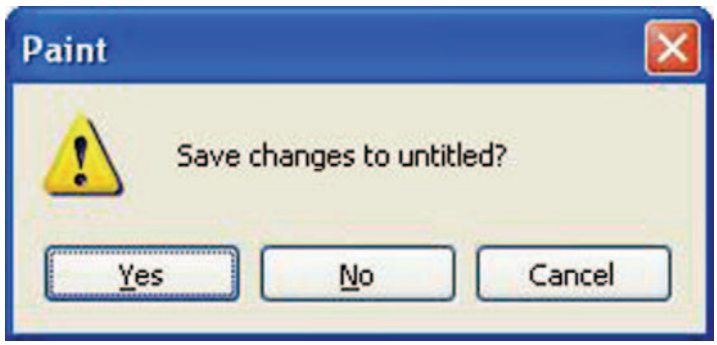


Figure 2.12: A program alert (not serious)

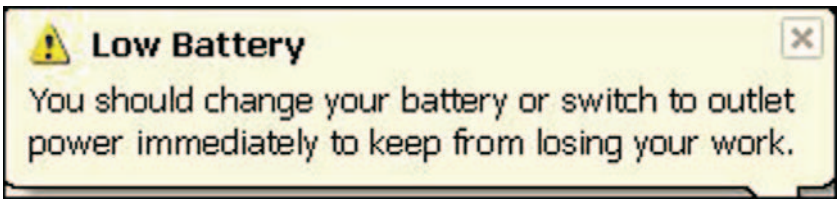


Figure 2.13: A more serious system alert

2.12 Dialogue Boxes

A dialogue box is a fixed sized moveable window in which users provide information that is required by an application so that it can perform a user request. Figure 2.14 illustrates an example of a dialogue box during a find operation in a spreadsheet:

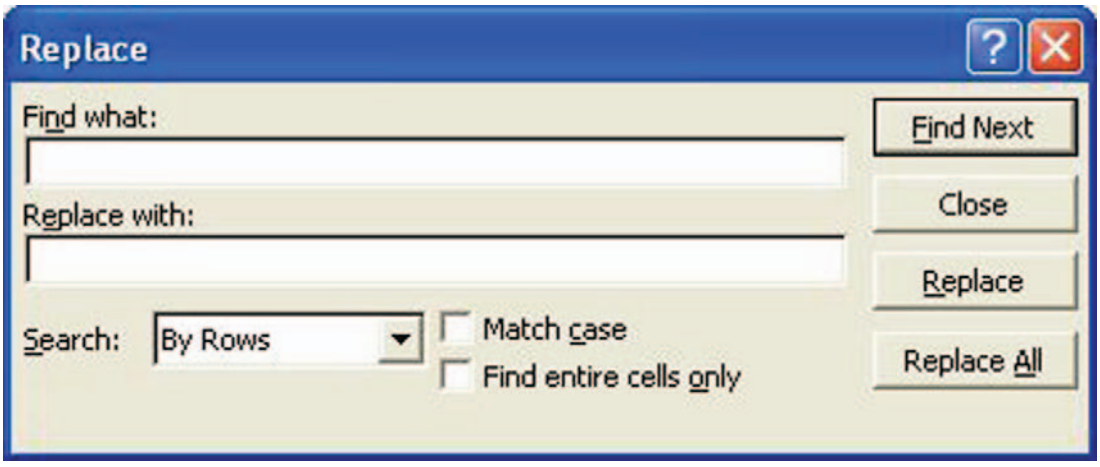


Figure 2.14: A dialogue box

2.13 Review questions

Q4: Many computers nowadays offer a graphical user inteface such as Windows.

a) Mention **two** features of such interfaces that are likely to be useful to a novice user.
b) Outline **three dis**advantages of this type of interface.

Q5: Consider the interface of

1. an application you are familiar with on the computer you are using (for example a word processor, or Web browser)
2. a mobile or cordless phone you use
3. a domestic or office appliance you use (microwave oven, photocopier, washing machine, etc.)

List the components of the interface, including those which enable input of text, numbers, or range-limited continuous or discrete parameter settings, and the parts which present feedback to the user (screen, LCD indicators, sound generators, etc.) For each, indicate which are the dominant interaction modes (visual, symbolic, speech, tactile, auditory, etc.)

2.14 Processing Capabilities of Graphical User Interfaces

The graphical nature of graphical user interfaces gives it the ability to perform a variety of different operations, including the combination of text and graphics that were never possible before using one type of interface. Some of these operations are listed below.

- What You See Is What You Get (WYSIWYG) editing
- Image Scanning
- Processable Graphics
- Animation and Support for Multimedia
- Porting of documents or files across different applications
- Provision for users with disabilities

What You See Is What You Get (WYSIWYG) Editing

WYSIWYG editing refers to the representation of an image on screen as being an exact image of the end result (as might be output on paper). This was previously possible but only with text-based documents done on word processors. Graphical user interfaces take WYSIWYG editing further since the interface screen can now provide reliable images of text and graphic outputs.

Animation and Support for Multimedia

The manipulation of graphics to produce moving images is called animation. Animation was originally found in game applications. However many businesses now use animation for presentation and marketing purposes. Animation is also necessary for

video conferencing which is used in some companies. A graphical user interface is also the predominant interface for Web browsers since animations using Flash, Shockwave and Java plug-ins are important for interactive activities.

Porting of documents or files across different applications

Sometimes it is necessary to use a file created in one application in another. In the past this required explicit code to perform file conversions so that this can be accomplished. This type of operation can be easily accomplished in a system with a GUI interface. Since a GUI treats all files as objects, it makes it possible to embed one object within another. When this is done, a link to the application that created the embedded object is made. Therefore whenever the embedded object is to be manipulated, the application that created that object is executed as a process separate from the application maintaining the whole document.

Provision for users with disabilities

The WIMP environment is ideally suited for adjusting components for users with visual, hearing and mobility disabilities. Special keyboards and mice can be adapted to use the WIMP system. In terms of visual impairment, icons can be made larger and in contrasting colours for easier use. Figure 2.15 shows normal icons and enhanced icons:

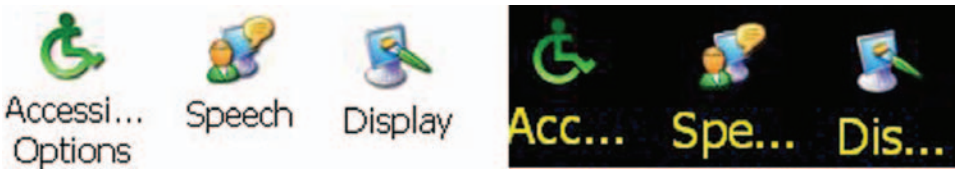


Figure 2.15: Normal and enhanced contrast icons

2.15 Advantages and Disadvantages of Graphical User Interfaces

Overall, the advantages of GUIs are:

- Users feel in control of the computer and are not intimidated by it.
- Typical user learning time is short.
- Users get immediate feedback on their actions so mistakes can be detected and corrected quickly.
- Icons help users to recognise what objects are meant to represent.
- Can facilitate the transfer of skills from one WIMP based operating system to another. For example, using Windows 2000 on a PC is very similar to operating the Mac OS 8 system. Although there may be small differences in symbols used to represent icons or command names in menus, the dialogue between the user and the system is essentially the same.

The disadvantages of GUIs are:

- These interfaces are processor intensive and memory demanding which imposes a high overhead on the system. Powerful and expensive processors are needed to adequately support these interfaces.
- Sometimes a longer sequence of steps is necessary to perform certain operations in a GUI than it is for another type of interface. This can be annoying for experienced users of computers.
- Users with vision or motion disabilities may have trouble navigating in a GUI. In some cases, special coloured filters can be used to partially alleviate this problem.
- Screens can be become cluttered and difficult to navigate.

All that said, GUIs today offer more functionality such as Plug and Play, increased responsiveness, stability, multitasking and enhanced Web support.

2.16 Special purpose interfaces

With the ever increasing development of hardware and software, many specialised user interfaces are now available. Table 2.2 summarises some areas of use:

Table 2.2: Special purpose interfaces

Interface	Areas of Use	
Touchscreens Also found in PDAs, PC Graphic tablets and Interactive whiteboards. Also found in some bank ATMs	Museums	Allows users to navigate menu system of art works, etc. by touching screen with finger.
	Photographic outlets.	Customers can now produce their own prints by navigating the menu and sub-menu printing system.
	Travel centres	Travel options can be obtained by touching the required icons on screen.
Speech	Visually and physically disabled users	Users can get audible sounds and words from the computer.
Natural language	Speech input systems	The most natural interaction style. Speech recognition software is improving greatly but is still not perfect.

For further references on GUIs you may wish to follow the links:

http://toastytech.com/guis/guitimeline.html

http://en.wikepedia.org/wiki/history-of-the-graphical-user-interface

2.17 Review question

Q6: Find examples of as many widgets as you can in two or three of the more common software tools (browser, word processor, spreadsheet, etc.) on the computer you are using. Is it obvious to you how to interact with these? Did you have difficulty guessing their use and purpose the first time you encountered them?

2.18 Summary

In this topic various interface designs have been described and illustrated where possible with suitable graphics and diagrams.

You should now be aware of the following objectives:

- the nature of interface designs including textual, menu driven and graphical;
- the relative merits of each in a computing environment.

End of topic test

Q7: What is meant by the term Human Computer Interface?

Q8: User interfaces have become more and more oriented towards the needs of users. Give **two** aspects of interfaces to explain why this is so.

Q9: Some computer systems use a graphical user interface, while others use a command-driven interface. Give **one** characteristic feature and one advantage of each interface.

Q10: A menu system is much faster for the expert user.

a) True
b) False

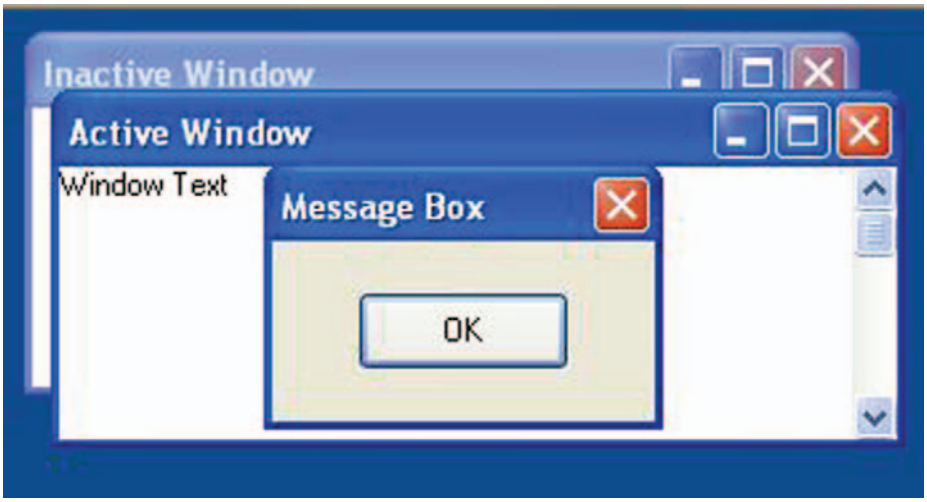
Q11: Which one of the following statements is true:

a) The processing of graphics is faster using a GUI.
b) A WIMP environment allows for no errors being made.
c) Command-driven interfaces are no longer used.
d) None of the above.

Q12: Which one of the following statements is FALSE?

a) GUIs are processor intensive and screens can be cluttered.
b) Novice users find GUIs intimidating since there are too many options.
c) Sometimes a longer sequence of steps is necessary to perform certain operations in a GUI than it is for another type of interface.
d) The learning curve for using a GUI is short.

Q13: Consider the following figure:

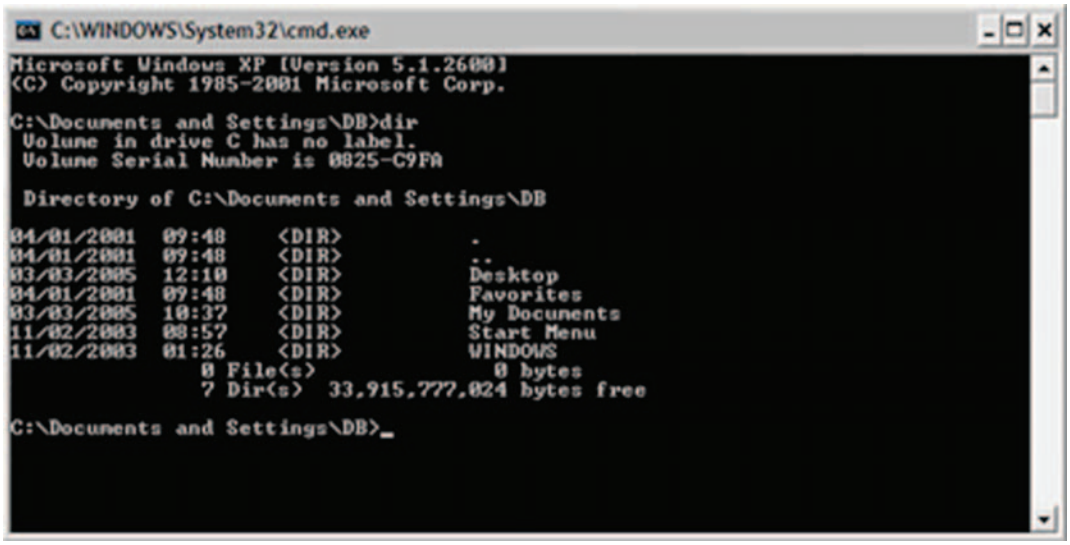


It is an example of:

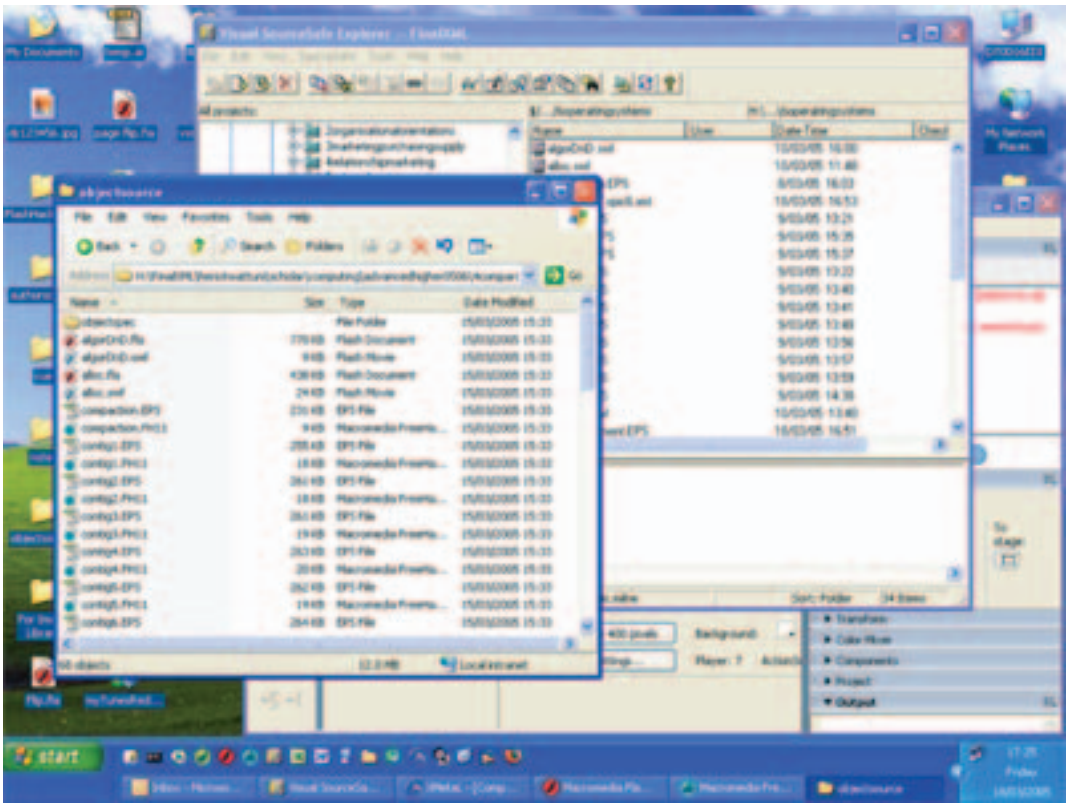
a) A dialogue box with tiled windows.
b) An alert box with overlapping windows.
c) A menu system with overlapping windows.
d) None of the above.

Q14: Which of the following operating systems is the odd-one-out?

a) MS Dos



b) Windows XP



c) Windows 98

d) Mac OS X

Topic 3

Software development languages and environments

Contents

Learning Objectives

After studying this topic, you should be able to:

- *describe object-oriented languages;*
- *compare object-oriented with procedural, declarative, event-driven and low level languages;*
- *explain trends in language development (low level to high level, 4th generation).*

3.1 Introduction

This topic introduces the concepts behind object-oriented languages with a brief history of their philosophy and development. They are then compared to the more conventional programming languages such as procedural, declarative and event-driven paradigms. Comparative features of low level languages are also covered in some detail.

Language development is a fairly dynamic process and this topic will examine the trends from low level organisation to high level constructs, including 4th generation languages.

3.2 Object-oriented languages

The first object-oriented language called Simula 67 was developed in the 1960s by Nygaard and Dahl of the Norwegian Computing Centre in Oslo during research into event-driven systems in computer simulations. Simula 67 contained most of the important concepts and techniques used in all present-day object-oriented languages.

Research progressed into the 1970s with the development of the language 'Smalltalk' and the subsequent emergence of current languages such as C++, Visual Basic, Delphi, Java and many others.

The popularity of such languages was enhanced by the increasing use of graphical user interfaces (GUIs) and the development of windows applications. Today these are the areas in which object-oriented techniques predominate.

3.3 Why object-oriented?

Conventional methods of programming invariably used imperative languages such as Pascal, Basic, or C. The program would be written in terms of procedures, which would process, or operate, on, the data passing through. This was still the von Neumann approach to computing. Such programs were sequential, starting at the beginning of the code and running to completion, branching and looping according to the program instructions.

A fundamental flaw of this type of programming was that large programs became more complex and tended to be unmanageable even allowing for the fact they were modular and well designed. Also global variables could be accessed and possibly changed by every part of the program during execution so results might not be valid. It was extremely difficult to code program modules or procedures in isolation that might have referenced global variables in the complete program.

Today computer programming is generally done in, and for, a windows environment. Windows applications can contain millions of lines of source code and this can be problematical in terms of project management. Not surprisingly this is an extremely error-prone situation where possibly many thousands of modules have to be integrated into a complete, working program. Even allowing for efficient top-down or bottom-up development techniques, and the best will in the world, problems are bound to occur. For example, should the project specification or a fragment of code be altered, this

would have serious repercussions on the development schedule and probably incur a complete re-write of a large chunk of the code.

Windows is an event driven environment and user interaction is required in the form of a mouse click or a key press to represent the processing. Procedural languages do not possess the necessary constructs to deal with this style of programming and are generally unsuitable for building windows applications.

Object-oriented programming techniques overcome these difficulties and make it easier for the programmer to implement good programming practices.

3.4 Object-oriented concepts

Object-oriented programming is a programming methodology, or paradigm, in that it compels the programmer to think in a different way about software and its development. Object-oriented programming enables the creation of software that can be more readily understood and shared with others. Unlike more traditional programming methods that are based on concepts such as data flow or some form of logic, object-oriented programming directly models the program.

The object-oriented language model makes use of the following concepts:

1. Objects
2. Classes
3. Inheritance
4. Encapsulation
5. Polymorphism

3.4.1 Objects

As the name suggests object-oriented programming is a language model centred around objects and, just as important, the data associated with them. An object is a logical unit that contains both data and the code that manipulates it. The two are regarded as a single unit, unlike the procedural viewpoint where data and code are separate.

This idea stems from the attempt to mirror object-oriented programming techniques with attempting to solve real-life events employing everyday objects. You do not need to look far to see examples of objects: a program, a bank statement, a tree, a cat, a car, a computer, a house, a person are all examples of everyday objects.

Such objects share two characteristics. They have:

- a state (data)
- a behaviour (operations)

Table 3.1 shows some examples:

Table 3.1: Object characteristics

Object	State	Behaviour
Dog	Size, name, colour, breed	Barking, playing, sniffing, hunting
Car	Make, model, colour, engine size	Accelerating, braking, reversing, gear changing
Tree	Species, shape, size, leaf colour	Growing, swaying, rustling, budding

For example, consider the desk in front of you. It has a location - you know where it is. It has a certain size and colour (Figure 3.1). It cost a certain amount of money. However, if we move the desk, then we change the location property. If we paint the desk we change the colour property etc. When we change the attributes of the desk (location, colour, size, cost) we do not change the fact that is it still a desk.

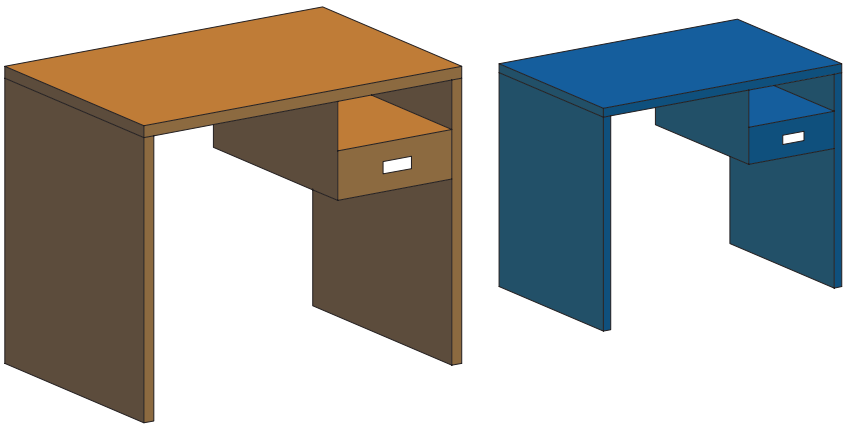


Figure 3.1: A large, brown desk and a small, blue desk.

In a similar fashion windows objects (command buttons, dialogue boxes, text boxes and so on) also contain two parts:

- attributes (i.e. data)
- operations (also called methods).

A schematic view of an object is shown in Figure 3.2, where the set of attributes of the object constitute its state.

Figure 3.2: A Schematic View of an Object

Applying this representation to an everyday object such as a car and a GUI object such as button the equivalence between them can be shown in Figure 3.3:

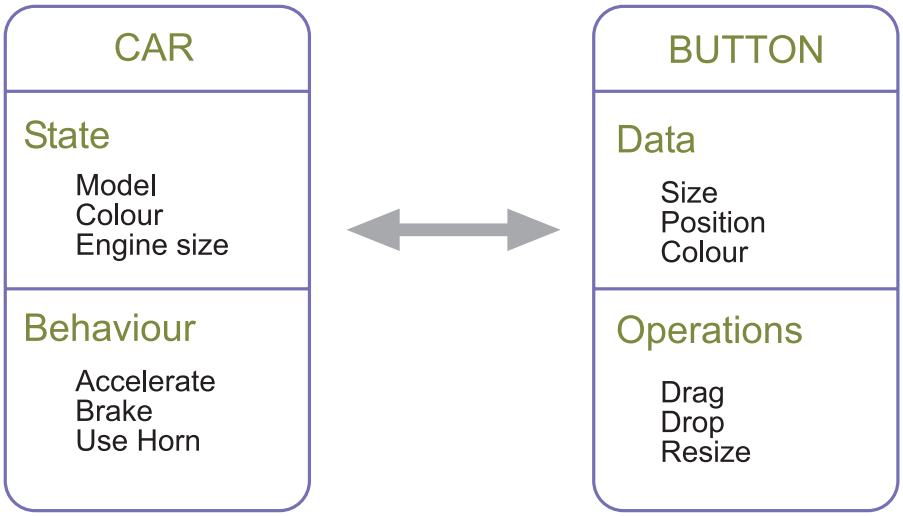


Figure 3.3: Objects

Objects on their own are, however, not very useful. Instead an object appears as a component of a larger application that will contain many other objects. For example the car object is just an assembly of metal and rubber sitting idle in a garage. By itself it is incapable of any activity. The car becomes useful when another object (a person) interacts with it and starts the engine.

Objects communicate with one another using messages and this will be discussed further under the section on encapsulation.

Exercise

Compile a list as in Table 3.1 of four GUI objects with data and operations.

3.4.2 Review questions

Q1: Give **two** reasons for the development of object-oriented languages.

Q2: For a language to be classified as object-oriented it must satisfy at least **three** requirements. What are these requirements?

Q3: Explain what is meant by an object and how it is represented in an object-oriented language.

Q4: State the attributes and operations for a Windows application icon.

3.4.3 Classes

Objects of the same kind share many characteristics. For example, a car is just one of many cars; a cat is one of many types of cat and so on. Because of this, a blueprint or template can be produced to accommodate all cars and one to contain all cats. Such a blueprint is called a **class**.

Using object-oriented terminology, a car object is an **instance** of the class of objects known as cars and a cat object is an instance of a class of objects known as cats.

Suppose there now exists a class called 'CAR' that contains instances of types of car such as four-wheel drive, saloon, hatchback and limousine. The class 'CAR' would become a **super-class** or **base class** and the various types become **derived classes** or **sub-classes**.

Figure 3.4 illustrates the base class CAR:

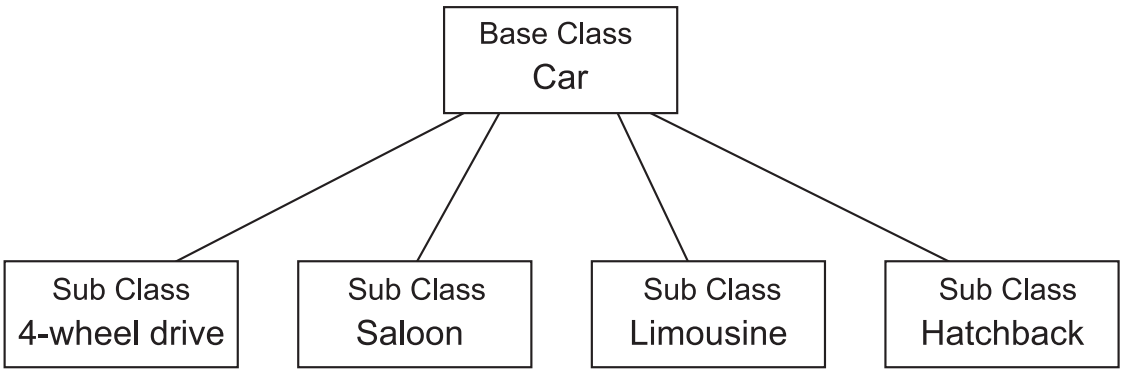


Figure 3.4: Class diagram

In each case the sub-classes all have common features based on the super class.

Just as large programs are sub-divided into modules, much of the art of object-oriented programming is determining the best way to divide a program into an economical set of classes. In addition to speeding development time, suitable class construction results in far fewer lines of code, which effectively means less errors and lower maintenance costs.

Also object-oriented languages depend greatly on the concept of **class libraries**. These are sets of classes that can be used by developers as common building blocks for complex applications, rather akin to module libraries that are used in procedural programming.

Exercise

Extend the class diagram in Figure 3.4 to include two new sub-classes of:

- the class *Hatchback*
- the class *Saloon*

3.4.4 Inheritance

We have already seen that objects of the same kind share many characteristics and belong to a specific class (type of car, species of cat etc.). It is possible to redefine a new type of object from an existing class by adding additional features without modifying the original class. In programming terms this opens up the possibility of reusing existing code and is known as inheritance. Re-use of code is probably one of the most important and strongest features of object-oriented languages.

Recall Figure 3.1 showing the two coloured desks. If we create a class called furniture and describe it then the desk can be made a member of this class. We can also add another class called table. In so doing both the desk and table will inherit all the attributes and operations of the class furniture (shaded portions in Figure 3.5) with additional features.

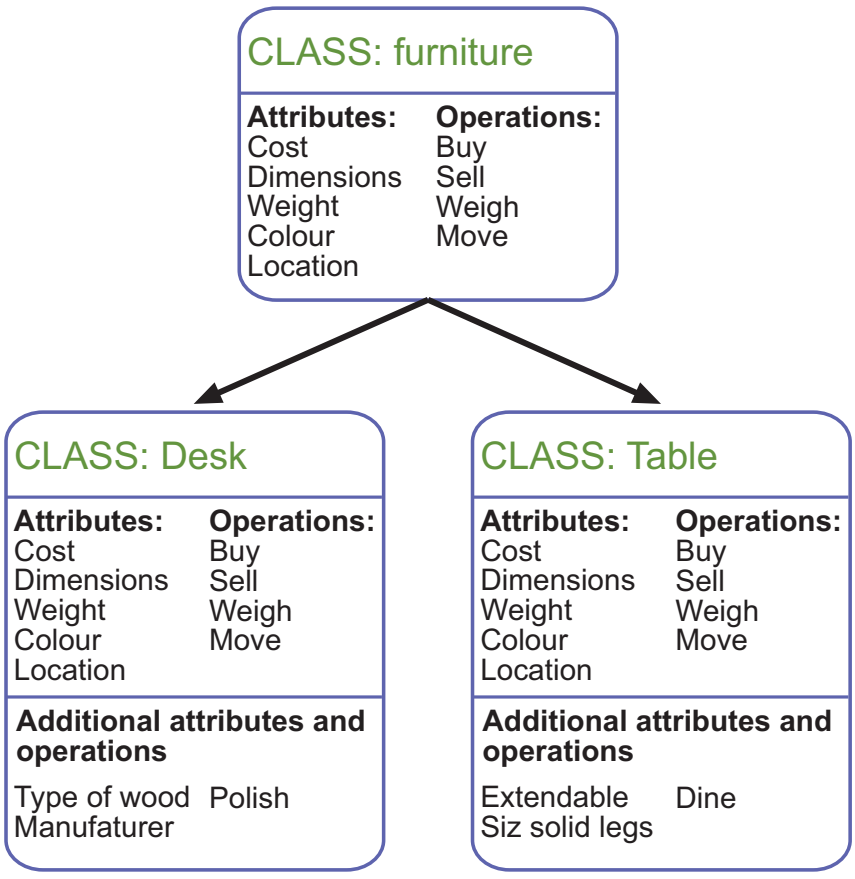


Figure 3.5: Example of inherited Classes

As another example consider the class 'DOG'. This could be viewed as being a member of the super class 'CANIS' that contains other dog species, all possessing basic, identical behaviour patterns.

Figure 3.6 shows this as a hierarchy called an **inheritance diagram**:

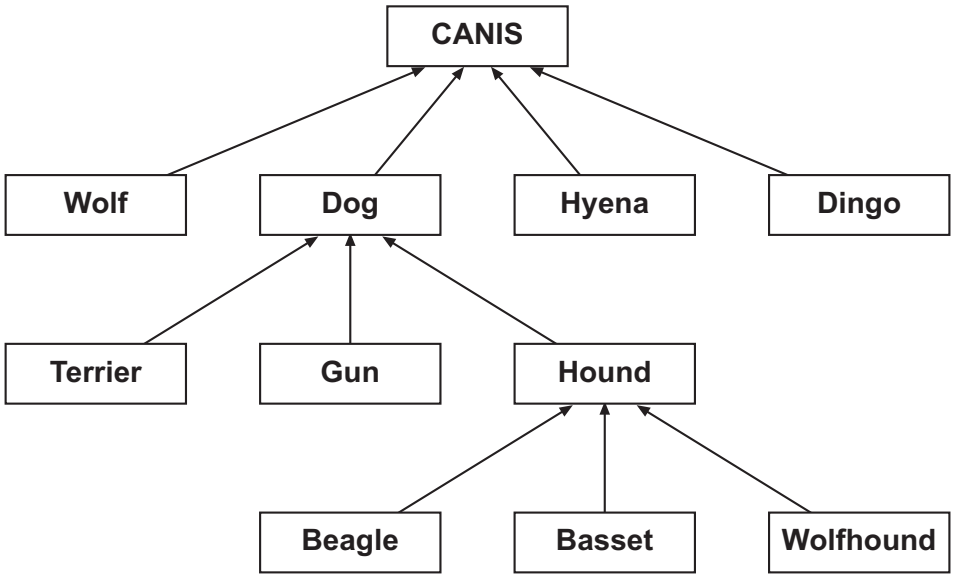


Figure 3.6: Inheritance Diagram

Note the directions of the arrows in the diagram. This is part of the diagramming 'nomenclature' for inheritance diagrams and effectively says that each sub-class *belongs* to the classes above.

Exercise

In a payroll application, for example, a class *Employee* might be defined with data such as *Name*, *Address*, *National Insurance number*, *Date of birth*. The application represents each person on the payroll by an object of the *Employee* class. For a large organisation, this might be extended to define a class *Department* and this would handle a collection of *Employee* objects (an array, for example), and offer operations such as listing the staff in the department and calculating the monthly pay bill. Each department in the organisation would therefore be represented by an object of the class *Department*.

Construct an inheritance diagram showing all the classes and sub-classes referred to in the payroll application.

This makes object-oriented programs easier to modify. Existing classes that have properties similar to what is required in the new program can simply be reused.

Moreover, if an error is found and fixed in the super class, it is automatically fixed in all derived classes!

This is significant when you consider that a large project, programmed using object-oriented methods could contain dozens of inheritance levels and hundreds of derived classes.

3.4.5 Review questions

Q5: Define the term class in an object-oriented language.

Q6: Suppose that there exists a class called *BankAccount*. List **three** possible data items and **three** corresponding methods of this class.

Q7: What is inheritance and why is it an important feature of object-oriented programming?

Q8: Give three examples of object-oriented languages that you have come across in your studies.

3.4.6 Encapsulation

You have already seen that an object contains data together with the operations that can be applied to the data. In object-oriented programming, objects interact with each other by use of **messages**.

The only aspect that an object knows about another object is the object's **interface**. In object-oriented programming an interface is quite an abstract idea. Just as a spoken language is an interface between two people for them to communicate through, so then is the interface of an object through which it communicates with other objects. In some object-oriented languages an interface is simply regarded as a protocol or an interface protocol.

This interface allows messages to pass between the object and the outside world. Values can be sent in to an object (usually by parameter) and the object can send values back (usually the return values of methods). See Figure 3.7:

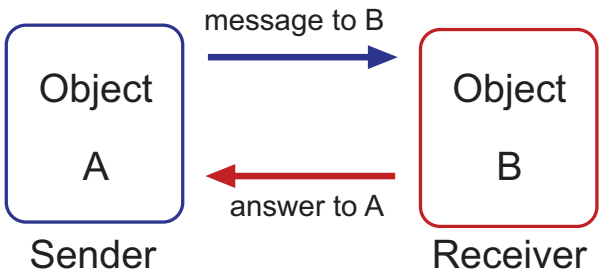


Figure 3.7: Messages and interfacing of objects

Good design often means no direct access is allowed to an object's data. As such the data is protected from unregulated alteration. The content of an object's methods is also hidden: the outside world can use an object's methods but cannot interfere with them. This means that objects cannot change the internal state of other objects in unexpected ways. This is called **encapsulation** or **information hiding**.

You may be wondering at this stage about all this concern regarding changes to the software. We live in the real world and during program development changes do occur to programming code, whether inadvertently or by need, resulting in lost development time. All too often software developers tend to tweak their code by enhancing it here and there in the hope that is will execute faster, for example.

There is a saying among programmers to the effect that *"software is not written, it is re-written!"*

Through encapsulation developers can carry out the functionality of an object to their applications and manipulate it through the object's interface, but they are barred from enhancing or changing the code or making it perform illegal operations.

3.4.7 Polymorphism

Two or more classes derived from a base class are said to be **polymorphic**. Polymorphism allows two or more objects to respond to the same message in different ways. Objects derived from a base class will have similar characteristics but can also have unique properties of their own as well.

An analogy of polymorphism to daily life is how students respond to a school bell. Every student knows the significance of the bell. When the bell (message) rings, however, it has its own meaning to different students (objects). Some students go home, some go to the library, and some go to other classes. Every student responds to the bell, but how they response to it might be different.

One of the classic examples in object-oriented programming is the 'shape example'. Here there is a base class called *Shape* with derived classes called *Circle*, *Square* and *Triangle*.

Figure 3.8 shows the inheritance diagram:

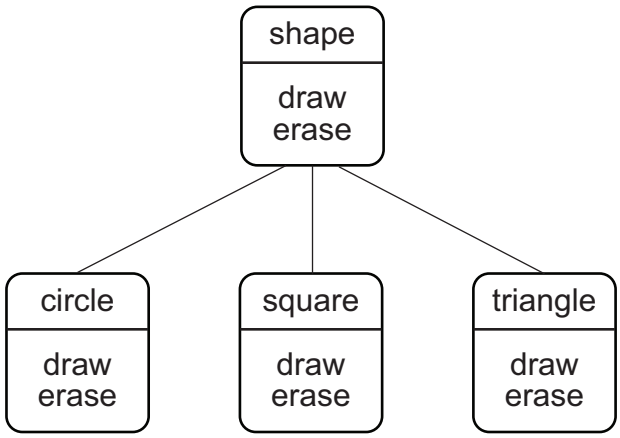


Figure 3.8: Inheritance Diagram for class Shape

When the message 'draw' is passed to each sub-class they will respond in different ways according to their unique code. Here, the results will be a circle, square and triangle.

If we wished to add another shape such as a rectangle then the present inheritance diagram can be extended to include the object *rectangle* complete with code.

Conclusion

Although some of the concepts of object-oriented languages may seem difficult to grasp at first sight, it is nevertheless important to realise that many benefits accrue in using the techniques. The more important ones are:

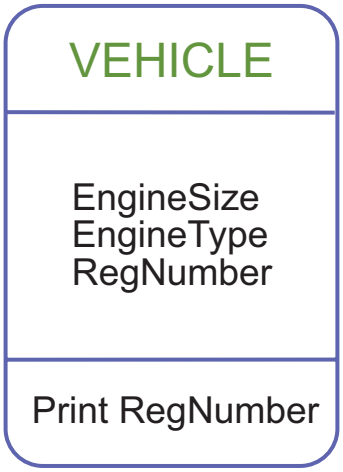
- Classes allow the programmer to create new classes that are not already defined in the language itself.
- The definition of a class is re-usable in other applications thereby reducing coding time and development time.
- Encapsulation provides a way to protect the data from accidental corruption.

If you wish to read more on object-oriented programming then you might find the following link to be useful:

http://www.eecs.utoledo.edu/~ledgard/oop/mainpage.html

3.4.8 Review questions

Q9: The following diagram is that of a class called VEHICLE:



Using the diagram explain what is meant by *encapsulation*.

Q10: What is meant by the terms *message* and *interface* in relation to objects?

Q11: Explain the term *polymorphism* and give an example.

3.5 Comparison of object-oriented with other language types

Computer programming is concerned with the use of computers to solve problems. Programming methods or paradigms represent different approaches both to the computers and the problems they attempt to solve.

This part of the topic compares object-oriented methods to those of procedural, declarative, event-driven and low level.

3.5.1 Procedural languages

In the procedural paradigm, the program is considered as a collection of blocks or modules of code (in general, *procedures*) that serve to process collections of data. Programs carry out their processing according to a defined sequence of events until the program eventually terminates.

Similarities of object-oriented languages with procedural:

1. Object oriented languages have all the high-level control structures for iteration and selection available to procedural languages.
2. Object oriented languages operate by changing the values of variables by using assignment statements. In low level terms, they change the values in the locations in memory that are set aside for data.
3. Both are problem specific languages.

Differences of object-oriented languages with procedural:

1. Object-oriented concepts might be difficult to learn by programmers with procedural programming experience.
2. In procedural languages a program module is, generally speaking, a free standing unit that can be compiled and executed on its own. In object-oriented languages, a module is always part of a class hierarchy.
3. In object oriented programming, data and code are not regarded as separate. Both are encapsulated into an object that consists of data and relevant operations or methods (equivalent to procedures and functions). The difference is that these methods belong to objects rather than to the program at large. If you develop an object in one program and then make use of it in another, the object will bring its methods with it.

   As an example, consider a list of records.

   In a procedural language, this would be declared as data, and modules would be written to carry out operations on the list: create it, add records to it, remove records from it, sort it, and search in it, and so on.

   In an object-oriented language, these methods are part of the object: instead of calling a procedure to sort the list, the list is told to sort itself.

Table 3.2 and Figure 3.9 illustrate the differences:

Table 3.2: Procedural view of list

List	Operations
Record 1	Create
Record 2	Add record
Record 3	Modify record
Record 4	Delete record
.....	Query list
Record 99	Sort list
Record 100	Print list

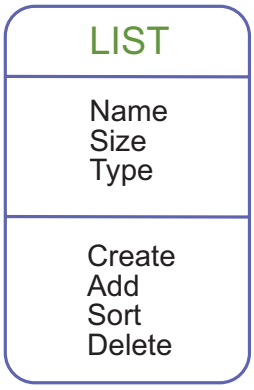


Figure 3.9: List object

4. Assignment statements may look the same but their interpretation is different.

   Consider the following assignments:

```
Number1 = 5
Number2 = 7
Sum = Number1 + Number2
```

   In a procedural language what this is saying is:

   "Take the variable Number1 which has the value 5 and add to it the variable Number2 which has the value 7, using the + operator. Place the result, 12 into the variable called Sum."

   In an object-orientated language the meaning would be:

   "Take the object Number1, which has the value 5 and send it the message "+" which includes the object Number2, which has the value 7. Perform the operation and create a new object, Sum to contain the value 12."

5. If a program written in an imperative language like Pascal encounters a problem and crashes then that is the end of the program. In an object-oriented language an error occurs in the code for a specific object then only that object will be affected and no others. The program will therefore continue to operate as normal.

Procedural and object-oriented languages are often grouped together under the heading *imperative*. (Such terms are, of course, used by people, and people's views will differ: some prefer to regard imperative (procedural) and object-oriented languages as distinct.).

Some imperative languages that have developed into an object-oriented environment are listed in Table 3.3:

Table 3.3: Derived object-oriented languages

Imperative	**Object-oriented derivative**
C	C++
Pascal	Pascal++
Modula-2	Modula-3
Basic	Visual basic

Exercise

List some other examples of present day object-oriented languages.

3.5.2 Declarative languages

The declarative paradigm treats the computer in an entirely different way. The computer is treated not as a machine that processes data but as a machine that can perform *logic* and produce answers. The programmer is concerned to declare the *form* of the result to be achieved, rather than to tell the computer *how* to achieve it.

In imperative languages, the programmer sets out sequences of steps for the program to follow. In declarative languages, the programmer declares *relationships* between items, in the hope that the system will produce a result.

Most declarative programming is carried out in one version or another of a language called **Prolog** (short for **pro**gramming in **log**ic).

A typical program consists of a *database* and a set of *rules*. The items in the database are known as *facts*. The general term for facts and rules is *clauses*. A fact consists of a single assertion with no conditions: a fact is always true. A rule consists of a goal, whose truth is dependent upon a set of conditions or sub-goals. The goal of a rule is referred to as its head and the sub-goals as its body. The database and the rules are submitted to a program known as the *interpreter*. The programmer sets goals, and the interpreter, following the given rules, seeks for these goals in the database.

Figure 3.10 illustrates this process.

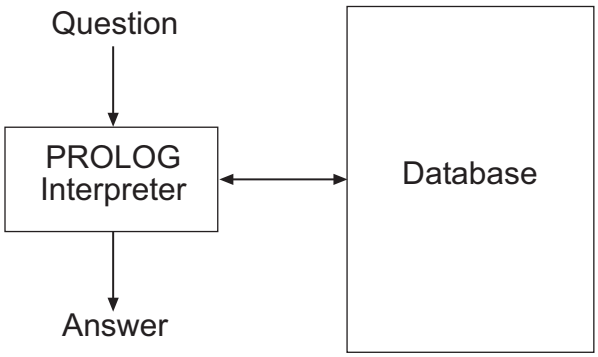


Figure 3.10: Declarative environment

Consider the information in Code 3.1 relating to the planets that has been **asserted** into the Prolog database. The 3rd arguement relates to distance in millions of miles the planet is from the Sun and the last number represents the number of moons the planet

has.

```
planet(saturn,large,900,atmosphere,rings,19).
planet(earth,small,92,atmosphere,none,1).
planet(mercury,small,35,none,none,none).
planet(pluto,small,3700,atmosphere,none,1).
planet(venus,small,68,atmosphere,none,none).
planet(mars,small,140,atmosphere,none,2).
planet(uranus,large,1800,atmosphere,rings,21).
planet(jupiter,large,512,atmosphere,rings,17).
planet(neptune,large,2800,atmosphere,rings,8).
```

Code 3.1: Planets database

We will now use the database to answer the following queries:

a) Find all planets with no atmosphere.

The structure of the query is

```
?- planet(Planet,_,_,none,_,_).
```

Here we are asking the database to return the names of all the planets that match the 4th responce as 'none'. Variables of no interest to the query are represented by the underscore character and this is referred to as the **anonymous variable**. Note the variable 'Planet' is capitalised and this means that a value for that variable will appear in the result.

The query returns the only planet with no atmosphere - Mercury.

```
Planet = Mercury
yes
?-
```

b) Find all planets with rings and moons.

```
?- planets(Planets,_,_,_,rings,Moons),Moons>0 * Input query to Prolog *

planet = saturn                                * Prolog output *
planet = uranus
planet = jupiter
planet = neptune
Yes
?-
```

Exercise

Using the planets database in Code 3.1, write queries to:

1. find all planets that are greater than 100 million miles from the Sun.
2. find all planets that are small, have an atmosphere, no rings and contain moons.

Similarity of object-oriented languages with declarative:

1. Because Prolog always operates on a database there is a mechanism built into it for searching the database. A tenuous similarity could be the case with object-oriented languages where the searching process is part of the object

Differences of object-oriented languages with declarative:

1. Declarative languages have none of the rigid control structures such as loops (`For..Next`, `While`, `Repeat`), `IF..Then` conditions.
2. All data structures in declarative languages are dynamic meaning that they only exist at run time. This makes the database more effective in dealing with the assertion and querying of rules and facts. Much use is made of the dynamic data structure called a stack (see Topic 6) to store current situations of goals and sub goals during recursive searches.
3. Declarative languages make use of type free variables that can represent widely different data structures. This allows any variable in a clause to represent a number, a string, a list of numbers, a list of strings, a list of strings and numbers, etc.
4. Variables can be reassigned in object-oriented languages but not in declarative. In Prolog, for example, once a variable has been given a value it is unchangeable.
5. The scope of a variable is within the rule it occurs in. There are no global variables in declarative languages.
6. An assignment is an instruction to the computer to give a variable a value. In a declarative language this is called **instantiation**. Whereas, the same word in oo implies creating a new object.

Recall the small program to add two numbers and store the result in the variable sum? In a declarative language, like Prolog this would become as shown in Code 3.2:

```
?-Number1 = 5, Number2 = 7, Sum is Number1 + Number2.
Number1 = 5
Number2 = 7
Sum = 12
?-Yes
```

Code 3.2: Prolog code

Notice the syntax in the first line. In the 4th line the variable Sum is instantiated to the value 12.

The language Prolog was extended in 1994 to **Prolog++** to include a fully functioning, object-oriented environment.

For example, the following statements, shown in Code 3.3, relate to two dynamic data structures, the stack and the queue and are from Prolog++:

```
class stack.
   inherits buffer          % inherits all characteristics of buffers
   ...
end stack.

class queue.
   inherits queue,          % inherits from general queues
   ...
end queue.
```

Code 3.3: Prolog++ code

Further reading

Should you wish to find further information on Prolog++ the following text is recommended:

Prolog++: The Power of Object-Oriented and Logic Programming, by Chris Moss. Published by Addison-Wesley, 1994

3.5.3 Review question

Q12: The following statement is from an article by Tim Rentsch, Professor of computer science at the University of Southern California:

"...Object oriented programming will be in the 1980's what structured programming was in the 1970's. Everyone will be in favour of it. Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programmer will practice it (differently). And no one will know just what it is."

From your knowledge of object-oriented languages explain whether you agree or disagree with the statement.

3.5.4 Event-driven languages

Historically, event driven programs dealt with such areas as embedded systems and control systems where input was usually through various types of sensors. Support for event driven programming was invariably grafted on to existing languages at the time to extend their functionality to deal with this environment.

With the ever increasing use of graphical user interfaces (GUIs) in computers today, new **event-driven** languages such as Visual Basic have emerged to accommodate the new style of programming. Also since GUIs such as MS Windows and Apple Mac OS are object based then object-oriented techniques feature heavily in this environment and these will be discussed later. In fact the early applications of object oriented programming were focussed on event driven systems.

As you are probably aware, conventional programming techniques are less suitable for the management of GUIs since there are no sequences of instructions to execute nor are there predefined pathways in the execution of the code. There are no start or finish phases either to program execution. In event driven environments the user executes what is known as an event procedure such as clicking a mouse or hitting a key to run part of the code.

Event driven programming is also referred to as **asynchronous programming** since

the computer waits for events to occur and responds to them as they happen. Such input events would include:

- keyboard events - key up, key down
- mouse events - mouse up, mouse down
- window events - window resized, window dragged

An event driven 'program' consists of a number of smaller units or procedures that respond individually to events, triggered by the user. These procedures are called **event handlers** and are executed by a special program called a **dispatcher**, which is part of the operating system, as shown in Figure 3.11:

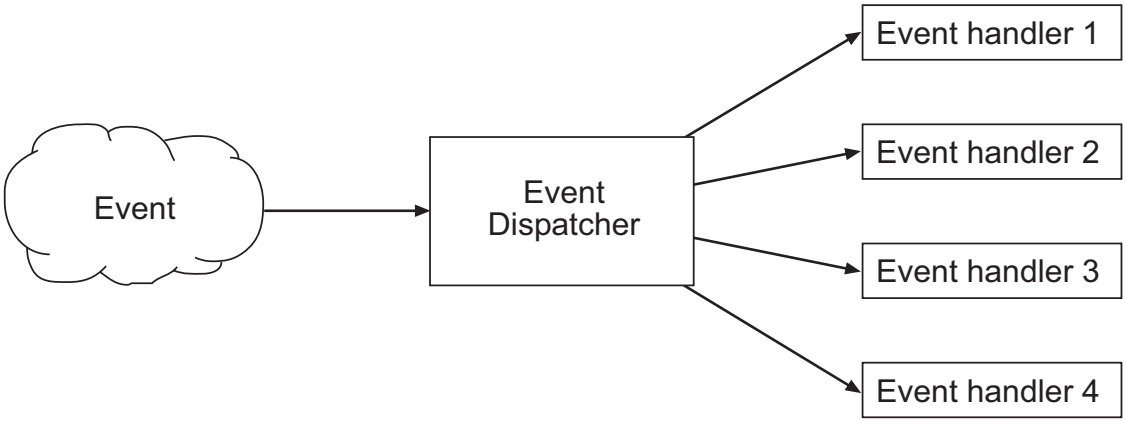


Figure 3.11: Control of events

The event dispatcher is called whenever an event is triggered. The dispatcher then forwards control of the event over to the appropriate event handler procedure.

Each event has a unique event handler. It is customary to associate each handler with two concatenated names to reflect the type of object and its action.

Table 3.4 shows some examples:

Table 3.4: Examples of event handlers

Event type	Handler required
Mouse	mousePressed
	mouseReleased
	mouseClicked
	mouseExited
	mouseEntered
Mouse motion	mouseDragged
	mouseMoved

The event dispatcher can be regarded as a large programming loop that waits for an event to occur, much in the same way as the operating system might wait for an interrupt to happen, through a polling system, for instance, and then respond accordingly.

Should an event be triggered while others are being processed then it is put into an **event queue**. All pending events in the queue will be dealt with, in order, only when all current events have been processed.

The following code fragment is represented in Code 3.4:

```
while true                          // loop forever until something occurs
while (empty(event.queue));         // wait for an event to happen
event = pop(event.queue);           // look at first item in queue
event = type(event.information);    // process event using appropriate event
                                       handler
```

Code 3.4: Event dispatcher loop

As previously mentioned object oriented programming techniques are evident in GUIs and work in harmony with event driven procedures. Languages like C++, Java and Visual Basic support the main constructs such as objects, classes and methods.

Objects that can be sources of events in a GUI include buttons, fields, scroll bars, graphics, check boxes, text areas, labels, frames etc.

It is quite difficult to distinguish between object-oriented and event driven programs. Quite a few languages can be categorised as being both! Visual Basic, for example is classified as an event driven language but it also supports object-oriented concepts. It is safe to say that all object-oriented languages are also event driven.

Summary

1. Event driven programs do not have a set sequence of instructions to execute.
2. Event programming and object oriented programming are inextricably linked in GUI systems.
3. Languages like Java and Visual Basic provide support for event-driven programming through the use of objects, classes and methods.

4. Inputs to event-driven programs come from event sources that require individual event handlers to process them. Some rules usually need to be set about which event has priority if multiple events are triggered at the same time.

3.5.5 Low level languages

At low level the focus is more on the computer rather than the problem to be solved. There is only one programming language that any computer can understand and execute and that is its own native binary machine code. This is the lowest possible level of language in which it is possible to write a computer program, consisting entirely of strings of binary 0s and 1s.

An enhancement to binary code was assembly code that used simple mnemonics to represent the various actions: load, add, sub, jump etc. but these varied from machine to machine as each had a different **instruction set** related to a specific processor. This defined what actions a computer could perform or not.

Both machine code and assembly code are machine dependent and are referred to as **machine-oriented languages**. Both require a detailed knowledge of every aspect of the computer's architecture, processor instruction set, registers, addressing modes, I/O details, memory layout and much more.

The programmer's intention, in choosing to work with low-level languages, is to write programs that run very fast and make efficient use of the computer's resources. Such example would include cases where the speed of a program (screen displays, device drivers) or the efficiency of its use of computer resources (memory), is of primary importance. A well written assembly program can optimise for speed and memory usage much more effectively than that of high level compilers.

Programmers in this paradigm have no high-level features available to them, either for program structure or for data structure, and have to do everything for themselves. An idea that is nowadays strongly stressed is that code should be easy to read. Low level programs are by no means easy to read and, as a result, maintaining and adapting them can be extremely difficult.

Summary

Low level languages:

1. Are machine dependent
2. Are difficult to program and debug
3. Are not compiled nor interpreted
4. Demand a large expense of time on the part of the programmer
5. Contain no high level constructs
6. Execute extremely fast
7. Produce lines of code that have a 1:1 relationship with machine instructions

3.5.6 Review questions

Q13: There is much more of a direct association of object-oriented procedures with event-driven languages in today's computing environments. Discuss.

Q14: In low level languages the main centre of attention is the computer itself. Give **three** reasons why this is the case.

Q15: In spite of this, low level programming is still used in today's computing environments. Give **two** areas where this might be the case.

3.6 Trends in programming language development

Before the existence of high level programming languages as we know them today, computers were programmed one instruction at a time using binary, octal or hexadecimal code. This was a dull, monotonous task with lots of errors being created in the process. The only people who could work in this code were mainframe engineers. Software development at this time was extremely expensive often costing many times as much as the computer itself. This led to the development of assemblers and assembly languages that made programming somewhat easier but this was still the domain of computer specialists.

Later, with the widespread use of computers, individuals wished to solve larger application problems. Assembly and low-level methods were rather inadequate in the resources they offered the programmer; it was difficult to describe a problem in terms of the operations required by the computer to solve the problem within such a mundane framework.

As a result a number of 'high-level' programming languages began to appear from the mid 1950s onwards, each supporting facilities and control structures for specific fields of application. Since the required resources were not supported in hardware, high level languages had to be developed to support them.

The trends in computer languages development can be viewed from various perspectives:

1. As a series of discrete generations, although there is some dispute as what regards the various generations.
2. As a chronological progression or timeline of events.
3. When a computing event initiates some sort of change that dictates future strategies.

Options 2 and 3 form the basis of the following discussion.

3.6.1 Machine code

The first code used to program a computer was called machine language or machine code. Machine code could also be called the oldest programming language dating back in the early 1950s, when it was the only means of programming available.

Machine instructions were in binary form consisting of opcodes, short for operation codes and addresses.

A 16 bit instruction may have the following structure depending on the machine architecture:

```
0 0 0 1   0 0 0 1 0 0 1 1 0 0 0 1
0 0 1 1   1 1 0 0 1 0 1 1 1 0 1 0
1 0 1 0   0 1 1 0 0 1 0 0 1 1 1 0

Op Code   Address
```

While they actually ran very fast, low level programs demanded a large expenditure of time on the part of the programmer. The main problems associated with machine code programming were:

- Programs would only run on specific machine architectures and would not transfer to machines whose architecture differed in any significant way.
- Low level programs were by no means easy to read and, as a result, maintaining and adapting them was extremely difficult. For example, to insert a new instruction at a certain point, all the existing instructions needed to be moved down in memory to make room for it. This in turn meant that the addresses of jump instructions needed by selection or repetition constructs also required to be adjusted.
- Programmers found it difficult to remember the opcodes and looking them up in the system manual was a tedious task.

3.6.2 Assembly code

Assembly language programming addressed the problem of programmers remembering all the opcodes and addresses of machine level programming by using symbolic code. The symbols represented a mnemonic that corresponded to the type of operation involved such as ADD, MOV, JMP, etc.

Assembly language was arguably the forerunner of all programming languages in that it was the first tool that relieved programmers from dealing with 0s and 1s and enabled them to use meaningful names for instructions.

A typical assembly program to add two numbers might look like:

```
LDA X  {load the value in the X register to the A register
ADD Y  {add the value in the Y register to the contents of the A
        register
STA Z  {Store the result in the Z register
```

This made writing and debugging programs a good deal easier.

However there was an extra overhead in that the mnemonics had to be translated into native machine code before the processor could carry out the instructions. This was achieved using an **assembler**. An assembler is itself a program and, as it is designed to produce machine code, must be written by a machine language programmer.

Like machine code programs, well-written assembly language programs have fast execution times and make efficient use of computer resources; each assembly code statement has a 1:1 relationship with the machine language statement.

Assembly language programs are a good deal easier to write, debug, and maintain than machine code programs.

However, assembly language programming still takes time. Given a certain specification, a programmer can easily take ten or more times as long to write an appropriate program in assembly level language, compared with one of the higher level languages that are now available.

Because programming time is now the most expensive factor in software development, assembly level programming is not greatly encouraged for general purposes. One exception is in cases where the speed of a program or the efficiency of its use of computer resources is of the first importance. Assembly language is therefore still used today in embedded systems and device drivers where there might not be much memory available and where the programmer has to make the best possible use of the given architecture.

Assembly languages developed further into the 1960s with systems that incorporated single lines of code that were translated into many lines of machine code. These programs were called **macro generators** and the most prominent of this type and also the most popular was the IBM 360 system that lasted for many years.

For further reading the following link gives an excellent discussion of low level languages:

http://www.dacya.ucm.es/luis/docencia/aetc/tae/c001p01.htm

3.6.3 High Level languages (HLLs)

With machine code and assembler languages operations being defined by the hardware, it soon became apparent that language development should now include operations more suited to the application program instead. The lack of portability between different machines was a deciding factor in stimulating the development of HLLs. Also, if assembly code could be translated into machine code then why not English?

Computer scientists and engineers discovered that computers were quickly becoming more popular worldwide. There was now an immediate need for more powerful computing languages than were currently available using assembly code.

The early 1960s thus witnessed the evolution of several high level, imperative languages, each designed for a specific area of use. These programs were block-structured (procedures) and the lines of code resembled English sentences and mathematical expressions so were much easier to read and debug. In fact COBOL was designed to be easier to read than assembly code. Each line of high level code now translated to roughly ten to twenty lines of low level code thus allowing programmers more flexibility in constructing programs.

For example, COBOL statements would look like:

```
PERFORM WAGE-CALCULATION UNTIL END-OF-FILE.

DIVIDE TOTAL BY NUMBER GIVING REMAINDER RESIDUE.
```

Table 3.5 summarises the features of the four most influential languages of their time, three of which have undergone modification and are still in use today, as we'll soon see:

Table 3.5: Early development of HLLs

Language	Developed by	Applications	Features
ALGOL	Algol 60 Committee	Universal	1st 'universal' language used to describe algorithms. Close to standard mathematics notation Block structured Portable across machine architectures Compilable to machine code.
FORTRAN	John Backus	Numerical	Programs more readable Easier to train new programmers Reasonably fast development time
COBOL	Grace Hopper	Business and commerce	Programming in 'natural' English Programs very readable File handling procedures Programmers required specialised training to write in COBOL
LISP	John McCarthy	List processing and AI systems	Common data structure for data and programs Uses recursive techniques

Early proposals for high level languages were said to be controversial. Critics said that they would yield programs with unacceptably slow execution times since they would have to be translated to machine code using compilers, which were fairly complex programs in their own right, but supporters argued that a significant reduction in coding and debugging time would result.

Both statements turned out to be true! In the case of FORTRAN the slow compilation speed was offset by the reduced time to program the application.

The use of languages such as ALGOL started the programming revolution and offered programmers many advantages including:

- The use of statements and keywords using English words
- Access to high level control structures, such as selection and iteration
- The use of high level data structures, such as **arrays**, **records** and **files**
- The capability of defining types and dynamic data structures such as **stacks** and

queues.

Programming at this time (1960s/70s) was mainly in the realm of scientists and engineers but this was to change dramatically.

3.6.4 The computing explosion

Progression through the 1960s and into the 1970s witnessed a dramatic increase in the number of programming languages being developed, estimated to be in the hundreds. There now seemed to be a need to write easier and faster error-free programs in areas other than that of science or engineering.

In education the language BASIC was developed to train students in programming while the language Pascal appeared later as a derivation of ALGOL.

The influential language 'C' developed by Dennis Richie at the Bell Laboratories in America emerged from the UNIX operating system and it is still an important teaching language today in many of its different guises.

Languages like COBOL, FORTRAN and LISP were metamorphisised into more powerful variants to keep them competitive in the current markets and also to take into account faster processing techniques and enhanced computer architectures. They did however offer backwards compatibility with their earlier versions to accommodate the use of legacy code.

Other notable factors were:

- Compiler techniques were becoming more refined so larger and more complex programs could be translated.
- Escalation of new hardware technologies (e.g. networking, cheaper memory)
- Expansion of specialised software applications (graphics, word processing, spreadsheets).

In some cases existing languages were enhanced with so many features that they became so complex and unwieldy to use. 'Language bloat' became a common feature of the times. However it was argued that the complexity of a language was directly proportional to the increasing complexity of the problems they had to solve. Consequently programs tended to run with reduced efficiency but with decreasing costs and increasing speeds of hardware meant this was well tolerated at the time.

By the end of this extremely productive era programming languages existed in every sphere of activity: business, commerce, industry, scientific, teaching, artificial intelligence, simulations, graphics, translation, etc.

3.6.5 Review questions

Q16: The high level language FORTRAN was created in 1954 and it is still in use today. Give **four** reasons why you think this is the case.

Q17: What other early high level languages do you think were influential in shaping trends in programming? Name the fields for which they were developed.

3.6.6 4GLs

Throughout the 1970s hardware technology evolved with relentless haste. Application software dominated the scene and methods of manipulating the data within and between applications were not conducive to imperative language techniques.

The first 4GL language was **Forth**, developed in 1970 for use in scientific and industrial control systems. 4GLs were typically developed to meet the special needs of data processing, with such applications as databases, spreadsheets and graphics applications. They were non-procedural and designed in such a way that users could specify the nature of the problem in a simple manner without having to understand the computer processing involved.

A typical database command in 4GL would take the form

```
FIND ALL RECORDS WHERE NAME IS "SMITH"
```

and the database would be queried until either a match was found or not found.

If, on the other hand, a list of values had to be arranged in numerical order, it is evidently much easier to use the command 'SORT' from the application rather than writing the code in an imperative language to achieve the same objective. In a spreadsheet, for example, Table 3.6 shows how a list of marks can be sorted in ascending or descending order by clicking on the appropriate icon as shown in Figure 3.12

Table 3.6: Sorting of spreadsheet marks

Name	**Mark**
Thomson Peter	93
Gerrard Tom	90
Jones Robert	89
Phillips James	71
Best Jenny	67
Calder Ben	66
Jenkins Rosie	55
Smith Wendy	51
Beatty Colin	50
Smart Mary	45

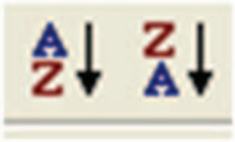


Figure 3.12: Ascending and descending sort button

Most 4GLs were written for specific purposes and incorporated the following features:

- database application tools (dBase, PARADOX, ACCESS macros)
- end-user tools like query languages (SQL (structured query languages), Postscript)
- report generators (RPG);
- application generators, sometimes referred to as **RAD** (Rapid-application development)

4GLs have evolved into such languages as Visual Basic for Applications (VBA) and are

responsible for many of the powerful in-built features that can be accessed from a menu or by clicking on an icon in present-day GUI systems.

3.6.7 Beyond 4GL

The terms 4GL and now 5GL are deemed to be rather imprecise terms but nevertheless they are used in the literature to define stages in computer language development. In essence no one really knows what comes after 4GL but we can see for ourselves the languages that are in use today.

Considering 5GLs there are various threads that exist:

- they are natural language systems involved in artificial intelligence and expert systems;
- 5GL is programming that uses a visual or graphical interface to create source code that can be compiled using a conventional high level language compiler;
- they encompass event driven, object-oriented programming and GUIs.

In the 1980s vast efforts were being put towards two large projects in artificial intelligence, namely:

1. the ADA language development in America and
2. Prolog development - the Japanese "Fifth Generation" Computer Project

It was thought that these two languages would dominate programming language development well into the 21st century. This is partly true but other developments took centre stage.

With the advent of GUIs:

- event driven, visual programming languages (Visual C, Visual Basic, Visual FoxPro) emerged to take advantage of these powerful interfaces;
- object-oriented languages (C++, Turbo Pascal, Java, Visual COBOL) also flourished under GUI environments.

Also with the rapid use of the Internet and the WWW, web-based languages were developed to enhance existing software or as languages in their own right. These include scripting languages (JavaScript, VBScript) and markup languages (HTML, XML) and many more.

It is interesting to note at this stage what the criteria might be for making a language popular. In a paper by R.P Gabriel "The end of history and last programming language" he makes the following main points:

- The language should be available to run on a wide variety of hardware
- It should be easy to learn
- It should be simple to implement

- Require modest computer resources
- Its code should be efficient at running any program of varying complexity

Does such a language exist?

Programming languages have developed greatly over the past 45 years and have been influenced by many factors throughout that time. Initially the need was to get away from machine dependent programming and to make languages more portable. Economic factors saw the costs of hardware becoming cheaper and more powerful and this heralded an upsurge in language production, notably procedural, for specialised areas. As compiler techniques improved more complex problems could be coded and solved in shorter times, this led to declarative and object-oriented paradigms being produced. Languages then became application centred and were supported by 4GLs. With the advent of GUIs, existing languages were revised and new languages developed such as event driven and scripting systems.

3.6.8 Review questions

Q18: What is meant by the term *4th generation language (4GL)* and how did they come about?

Q19: What, in your opinion is meant by 5GL?

Q20: A software developer chooses to use an object-oriented approach with the aid of a visual programming language. Describe **two** advantages which their use might have over the use of a traditional, procedural language

3.7 Summary

This fairly extensive topic has described the nature of object-oriented languages and the structures and concepts involved in their use. Object-oriented languages are then compared to other language types with relevant summaries included. The trends in programming language development are discussed and brought up to date with 4GLs and beyond.

By the end of this topic you should now be aware of the following objectives:

- an awareness of the concepts involved in object-oriented programming and some of the OOP languages that are in use today;
- an appreciation of the differences between object-oriented languages and other programming paradigms, in particular imperative, declarative, event-driven and low level systems;
- be aware of the trends in programming languages development should be recognised as a timeline from the early 1950s to the present day.

End of topic test

Q21: In object-oriented programming what concept allows objects of a derived class to have the same characteristics of its associated base class?

Q22: A programmer chooses to use an **object-oriented** approach, using a **visual programming** language. State what is meant by the terms in bold.

Q23: Give **one** advantage of using an object-orientated approach and one advantage of using a visual programming approach.

Q24: Which one of the following statements is false?

a) A translator converts source code into an object code
b) A different compiler is required for each high level language
c) An assembler converts assembly language mnemonics into binary
d) A machine code program can be executed by any CPU

Q25: Name **two** programming languages that can handle events.

Q26: Give **two** reasons for the abundance of programming languages today.

Q27: Considering 4GLs,which one of the following statements best describes them?

a) They were developed to meets the needs of data processing
b) They are non-procedural
c) They lie behind many applications
d) All of the above

Q28: 5GLs have taken over from 4GLs to allow programming using object-oriented concepts.

a) True
b) False

Topic 4

Software Testing and Tools

Contents

Learning Objectives

After studying this topic, you should be able to:

- *explain module, component and beta(acceptance testing);*
- *describe debugging techniques: dry runs, trace tables (tools), break points;*
- *describe the use and advantages of Computer-Aided Software Engineering (CASE) tools.*

This topic takes an in-depth look at large software system testing procedures before the finished project is deployed commercially. This is followed by a review of debugging techniques that might be used by all types of programmer when testing their software. Finally a description of the uses of CASE tools is investigated together with their relative advantages and disadvantages

4.1 Software testing in more detail

Software testing involves a series of processes in order to verify that:

- the finished software product meets the original specification.
- the software is robust.
- the software is reliable.

Testing can only find errors or bugs in a program, not prove that there are none!

There is the story about Thomas Edison of light bulb fame. After a thousand failed attempts at making an electric light, he said he was making progress because he now knew 1000 ways of how not to make an electric light! Testing can be bit like this.

Testing should be systematic. It's certainly not enough to run the program, insert a few values and see what happens. For large commercial systems that might use expensive, exhaustive testing methods, errors will still be found.

Test data also needs to be planned, and the testing process needs to be continually recorded in a test log. A typical test log might consist of these four items:

- input
- reason (for choosing that input)
- expected output (expectations being formed on the basis of the specification)
- actual output

The actual test data that is selected will depend on the testing strategy that is adopted.

4.1.1 Test strategies

The purpose of test data is to determine that the system behaves as expected. Tests may be used for different purposes. These include:

- module testing, on all components of the system;
- integration testing, where the components function as a single entity;
- system testing or black box testing;
- acceptance testing or beta testing, ensures that the system is ready for operational use.

Software testing is a complex and expensive issue. There are a number of different testing approaches that are used ranging from the most informal ad hoc testing, to formally specified and controlled methods. The following table contains some of the many generally accepted classifications amongst the many that exist and the ones that we are interested in at this stage are highlighted in Table 4.1 :

Table 4.1: Scope of testing procedures

By scope	**By purpose**	**By life-cycle phase**
Component testing	Correctness testing	Requirements testing
Module testing	Performance testing	Design testing
Integration testing	Reliability testing	**Acceptance testing**
System testing	Security testing	Maintenance testing

Testing ’By Scope" starts at component level then proceeds through subsequent testing phases, each one more complex than the previous, until the final system is tested as a whole.

Testing "By purpose" entails the whole reason for testing in the first place, namely that the finished program is correct and runs to the original specification.

Finally testing by "Life-cycle phase" includes the final software development test of all, namely acceptance testing when the software is deemed to fit for purpose and sold to the client as a viable product.

4.1.2 Component testing

Components, sometimes referred to as units, are the building blocks of software applications and the first rung of the testing "by scope" ladder. They are defined as pieces of software code such as subroutines, procedures or functions that can be compiled and executed on their own. Usually a component will be the work of a single programmer who will also be in good position to test the code and ensure it functions correctly. Hence the testing of a component will usually be at a fairly basic level and use white box testing techniques.

White box testing, also referred to a glass box testing or structural testing is so called since the actual component code is seen and it is the workings of this code that are tested. Programmers often use white box testing as they proceed with the program to ensure that it is working as it should under test cases.

The component can be tested in **static mode** or **dynamic mode**. In static mode the component does not require to be executed. Instead a detailed catalogue of reviews and inspections is compiled for each module detailing program information. The more rigorous of the two modes are the inspections that contain a precise framework for rigorously checking both component documentation and code. Using this method supporters claim dramatic decreases in the errors found in the final software system.

These methods are sometimes referred to as *Fagan inspections*, named after the programmer who introduced them while employed with IBM.

In dynamic testing the component program is executed using test values for the usual normal, boundary and exceptional cases.

Once a component has been thoroughly tested and is working to specification it may be combined with other working components to form modules and the process is repeated until a complete software system has been created.

4.1.3 Module testing

Modules are a collection of dependent components, procedures or functions designed to be parts of the main program. They are not complete programs in themselves, and as such, they cannot run independently. It often happens that a module becomes ready for testing before the rest of the main program is in a condition to supply it with realistic data. In such a case, it is common practice to write a small program, often called a **driver** or a **test-harness** to run the module and supply it with appropriate test data and an interface under a test simulation environment.

To save time driver programs are now commercially available; they scan the module source code, analyse the inputs and outputs and automatically generates the test code necessary to construct a test-harness, complete with the required input and output drivers.

Use is also made of stubs. A **stub** is a module that has the appropriate interface but does not contain a lot of code. It might contain, say, no more than a line of code to display what the module will do when it's been completed or to return an arbitrary or random value of the appropriate type. The system as a whole is laid out using stubs to ensure that, from the first, the overall structure is correct. The stubs are then converted into fully functional modules.

Needless to say these software resources free up valuable time for the program developers in that they can spend more time on coding and less time on testing.

With sizeable applications, containing a large number of modules things become a little more complicated, however. The system may have to be divided into sub-systems that are developed separately. Modules are created and tested, within their sub-systems. When all modules have been tested, the sub-system as a whole is then tested. Following this the numerous sub-systems are brought together, or integrated, and the entire system is tested. At this stage any errors existing between the sub-systems are identified and rectified and when complete, the program is said to be correct in as much that it runs to the original specification.

At system level the type of testing would be black box testing. Exhaustive white box testing would possibly take years to accomplish at this level.

Black box testing takes the program specification as the sole source of test cases. If the system stands up to black-box testing, it is passed as acceptable.

One advantage of this approach is that the developers who test a system can be different from the programmers who create it. In some cases, program testers don't even know the programming language in which the program has been written. They read the specification, create test cases, and test the program and, if any output does not match the expectation as defined in the specification, it is sent back for what's sometimes called a bug-fix.

Finally, the system is installed on the client's site, and is subjected to what is called **acceptance testing**: if it passes, it's accepted.

The testing process involves feedback. Sub-system testing, for example, might indicate errors in a module, which would need to be debugged and tested again before the sub-system can be re-tested. Figure 4.1 represents these processes in context:

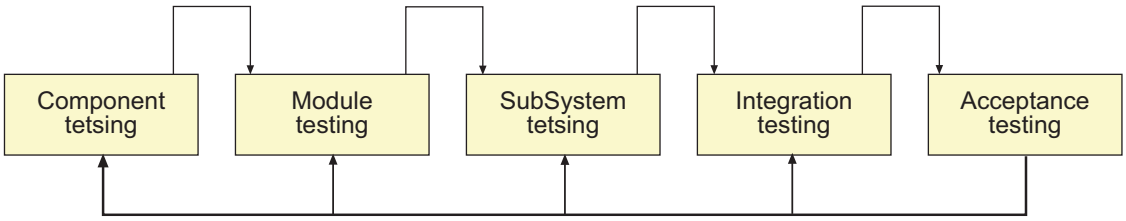


Figure 4.1: Progression of testing diagram involving feedback

With commercial software projects, the usual strategy is to test the software twice. The software is first tested within the organisation and this is known as **alpha testing**. This is followed by the final phase in the testing process which is **acceptance testing**, also known as **beta testing**.

Alpha testing can be regarded as "does the software work?"

Acceptance testing is more for the client to say, "are we willing to pay for the software?"

4.1.4 Alpha testing

Alpha testing is the stage of the development cycle where the software is first able to run but it may not contain all the features that are planned for the final version. Alpha testing usually takes place on the software developer's premises

Alpha testing is typically done for two reasons:

1. to reassure clients that the software is in a working condition but not released to them
2. to find errors that may only be found under operational conditions

Alpha testing is performed on an early version of the software. However since it will not have all the intended functionality it will have core functions and will be able to accept inputs and generate outputs in accordance with user specifications.

Usually, the most complex or most used parts of the code are developed more completely during the alpha phase, in order to enable early resolution of design questions that might arise. Nevertheless due to the very nature of testing an entire system, Alpha testing traditionally occurs quite late in the software development cycle. Unfortunately in many cases, time for testing is squeezed very tightly at the end of the development. Thus compromises in the quality of testing procedures are made

Ideally Alpha testing should be conducted with as much independence as possible from the development team and any of the personnel that performed other forms of testing up to that point. Independent scrutiny can bring a fresh view point to the project.

However in the real world, not many organisations have the resources for an independent Alpha test team. Consequently testers have to wear both hats. An alternative is to co-opt personnel closer to the customer, such as consultants and implementers to do the testing but this, consequently adds to the overall costs of the project.

4.1.5 Beta (acceptance) testing

Acceptance testing is the highest level of test in the software development cycle and is a test for a software product prior to commercial release.

It confirms that the program is as near correct as possible in relation to the requirement specification. It follows the sequence of alpha testing if a program has been developed for use by particular clients, and is installed on their site. The clients use the program for a given period and then report back to the development team. The process might be iterative, with the development team making adjustments to the software. When the clients regard the program's operation as acceptable, the testing stage is complete.

The organisation generates a build of the software, with the production standards of a released product. Ideally the software should be packaged with the documentation and other artefacts that are also to be delivered. Installation documentation and the basic operational documentation are especially important in this respect.

Acceptance testing involves sending the product to beta test sites outside the company for real-world exposure or offering the product for a free trial download over the Internet or sending it out on CD.

If the software is to be a commercial product then it is sent to a variety of independent users who agree to test the software and report and log any faults or defects. The program will then undergo modification and an updated version submitted for re-testing. The process will be repeated until the software developers believe that the product is ready for distribution. Finally any minor bugs that come to light will be corrected by software upgrades or patches.

How the testers are selected depends on the software and the environment into which it is to be released. For example if the project is a very large system for a government department, then the beta testers should be drawn from the team implementing the system and experienced end-users. Conversely a mass market graphics product may be tested by many hundreds of artists, drawn at random from interested parties.

Finally there has to be a mechanism for the beta testers to log incidents or defects and report back to the developing team.

The positioning of the final product within the software development cycle is closer to market than that of Alpha testing due to the increased independence of the testers. In the real world, however, commercial pressures will often determine the beta release. For example if a release date has been sent to the press, embarrassment will ensue if it slips. Thus beta testing may start before the software is ready.

Even at this late stage in the process, functionality delivered in the beta release may not be what is in the final product due to the under-resourcing of testing procedures or other organisational pressures. However if major modifications are needed, then something has gone seriously wrong with that organisation's development processes and the client has every right to take legal action.

4.1.6 Review questions

Q1: What is meant by component testing?

Q2: What are the difficulties encountered in module testing and how are they overcome?

Q3: Explain the term alpha testing and where it is performed during the development process.

Q4: Describe **three** main features of acceptance testing.

Q5: What **three** processes does the process of software testing hope to achieve?

4.2 Debugging methods

In the following section all examples of the debugging methods are exemplified in Visual Basic. However the general principles outlined still apply to any programming environment.

A bug is a fault in a program that causes it to function abnormally. Debugging is the process of locating and fixing errors in a program, and programs that help to do this are called "debuggers".

Most software development environments offer a range of debugging tools for help with programs that produce unexpected output when executed, but the amount of support in this respect can vary from language to language.

Generally speaking interpreted languages have little need for extra debugging programs (but some do) since an error will cause the program to halt at or near an offending line when the program is run. Compiled languages however produced object code which is then executed independently of the program source code. The first task of a debugging tool is to link the object code to source code in order to locate the error in the source code when the program is run. A special part of the compiler called a **link loader** is responsible for this.

Note that with the debugging feature activated a program will be compiled somewhat slower so it is only used when absolutely necessary when other methods of debugging have failed.

Debugging tools usually take the form of:

- Dry runs
- Trace tools
- Watch statements
- Break points

4.2.1 Dry run

Bugs can be extremely difficult to locate and correct. The temptation is to stay at the computer and tweak the code until the program runs correctly. If this is done unsystematically, it can change the program from one that only had a single minor error to one that has multiple errors. If the bug does not become apparent readily, usually the best course is to get away from the computer and conduct a dry run, if not of the entire program, at least of the part of the program that seems to be causing the trouble.

The program might be giving wrong output or performing an infinite loop which is a common programming error.

A dry run, sometimes referred to as desk checking, is based on a listing of the code or algorithm. The programmer works through the code manually, using pencil and paper and takes over the role of the computer. Test data is entered to check that the program or part of the program, subroutine or function is working correctly.

The trace table (see Table 4.2) contains the program variables and each row of the table will display their values as the logic of the algorithm unfolds. The actual output can then be checked against expected output and any discrepancy will, hopefully, reveal the error.

This technique is useful for locating logic errors in a program and is only practical for fairly simple programs and small amounts of data.

Consider the following Visual Basic program fragment in Code 4.1 to calculate the average of a list of marks input by the user:

```
Private Sub CmdRun_Click()
âĂŸAverage marks program
Mark = Sum = Counter = Average = 0
Do While Mark <> 999
  Mark = InputBox("Enter a mark then 999 to end")
  Sum = Sum + Mark
  Counter = Counter + 1
Loop
Average = Sum / Counter
PicResult.Print Average
End Sub
```

Code 4.1: Visual Basic program fragment

After the variables are initialised a loop is entered where the user is asked to enter a mark. The loop terminates when the dummy or rogue value 999 is entered to signify the end of the input. The value of the average mark is then output.

A trace table can be constructed to show the values of the variables as the program executes in 'dry run' mode. See Table 4.2

Table 4.2: Trace table

Loop	Counter	Mark	Sum	Mark = 999?	Average
0	0	0	0	False	
1	1	52	52	False	
2	2	68	120	False	
2	3	47	167	False	
3	4	50	217	False	
5	5	999	1216	True	243.2

Not the answer that was expected. It seems that the rogue value 999 has become involved in the calculation and it shouldn't have been!

Inspecting the code reveals that the while loop is testing the value of mark too late in the program to terminate the action of the rogue value.

The remedy is to copy the statement

```
Mark = InputBox("Enter a mark followed by 999")
```

and place it before the `while` loop so that the value can be checked before the loop is entered. Also the present statement is moved down to the end of the loop so that the values of `Sum` and `Counter` are unaffected.

The revised code in Code 4.2 now looks like:

```
Private Sub CmdRun_Click()
'Average marks program
Mark = Sum = Counter = Average = 0
Mark = InputBox("Enter a mark and end using 999")     'New statement
Do While Mark <> 999
  Sum = Sum + Mark
  Counter = Counter + 1
  Mark = InputBox("Enter a mark and end using 999")   'New position
Loop
Average = Sum / Counter
PicResult.Print Average
End Sub
```

Code 4.2: Revised Code

This gives the expected answer of 54.25 for the average with the values of `Counter` and `Loop` exiting when their values = 4.

Dry Run Exercise

Consider the following fragment of a program (Code 4.3) which finds the sum of two valid numbers:

```
Dim first As integer, second As integer

Sub getValidNumber (ByRef number As integer)
Dim numberOk As boolean
     do
           print "Input a number"
           numberOk = number > 0
           if not numberOk then
              print "Make it positive"
           end if
     loop until numberOk
end sub

function sum (a as integer, b as integer)as integer
     sum = a + b
end function

'Main program
     getValidNumber first
     getValidNumber second
     print("The total is "; sum (first, second))
```

Code 4.3: Sum program

Perform a dry run on the program using the data shown and construct a trace table to show the program output.

Data: a = 6, -6, 0, 2, 3, 1: b = 0, 3,3, 8, -7, 4, 6

4.2.2 Trace tools

Some programming languages have TRACE facilities as a debugging feature.

Tracing gives access to otherwise invisible information about a program execution. Apart from allowing the programmer to step through the program line by line (see watch and breakpoints), and stop at given points to examine variable contents, more enhanced tools exist that allow investigation of memory locations and, in particular, the contents of the stack.

Programs that contain large number of procedures use the stack to store all their procedure calls during program execution. By examining such data any errors occurring in the order of procedure or function calling from the main program can be checked and corrected.

The tool can be toggled between TRACE On and TRACE Off. With the TRACE on activated the program will execute slightly slower and produce extra output.

4.2.3 Program Watch

A watch takes an identifier and displays its value as the program progresses. The programmer steps through the code, one statement at a time, and the value of the variable being traced is displayed on the Watch screen, an example of which is shown in Figure 4.2:

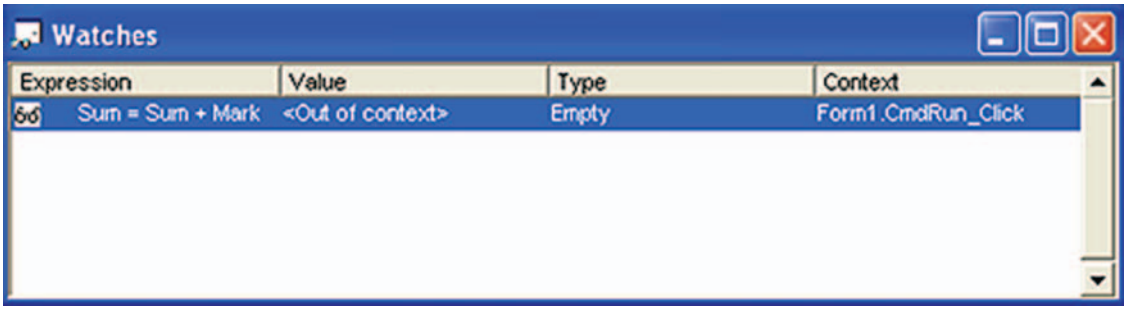


Figure 4.2: Watch screen

Many programmers prefer to put in watches for themselves in the form of output statements that cause the value of a variable to be displayed at the points in a program where a bug is thought to exist. However the drawback to this method is that it extends the number of lines of code and increases programming time in inserting and deleting the statements.

To set a watch in Visual Basic load any program and access the Debug menu. You will see the following drop-down menu as in Figure 4.3:

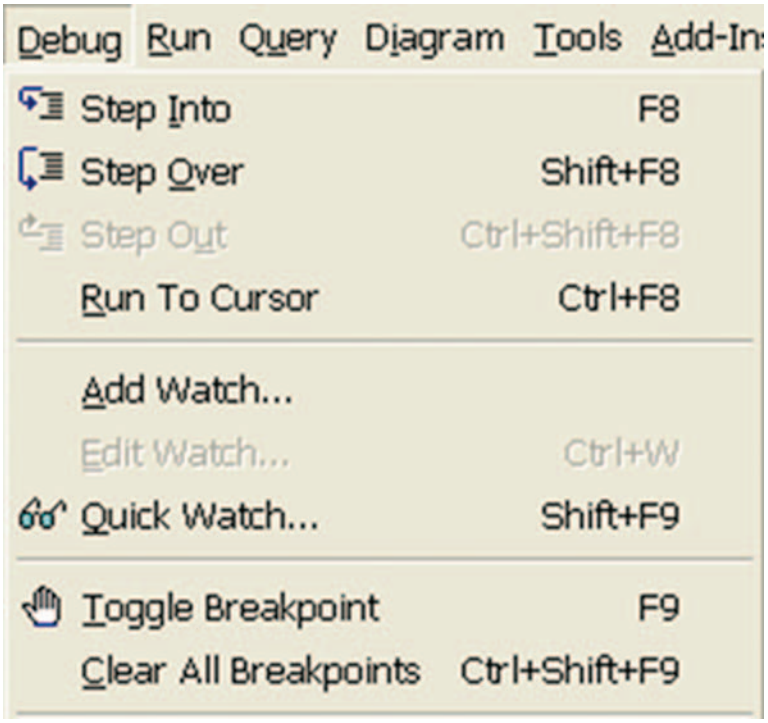


Figure 4.3: Visual Basic Debug menu

Choose the `Add Watch` feature and a dialog box will appear, as in Figure 4.4:

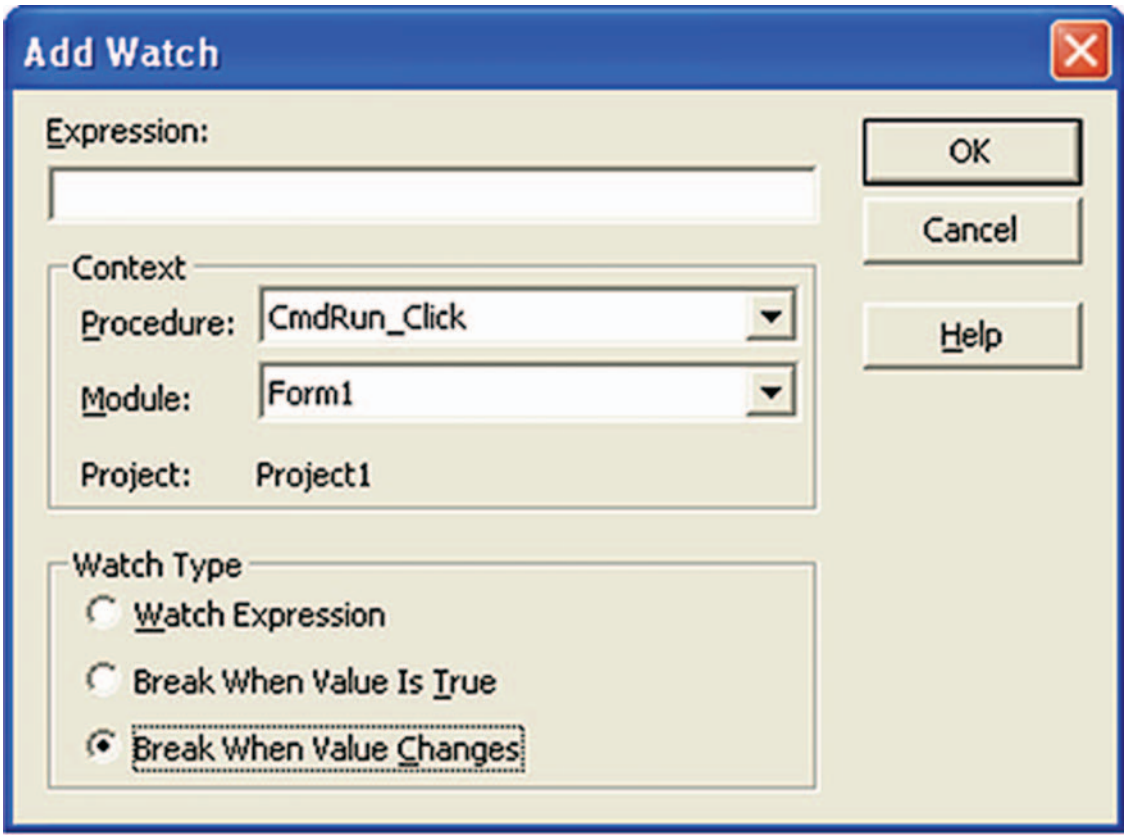


Figure 4.4: Watch dialogue box

In the `Expression:` box type in the statement that you want to watch and click on OK.

When he program is now run in debug mode it will focus on the 'watch' statement and allow you to step through the program line by line by pressing function key F8.

Figure 4.5 illustrates what is happening. Notice also that by placing the cursor over any of the variables, the value of that variable is shown, in this case `Mark = 34`

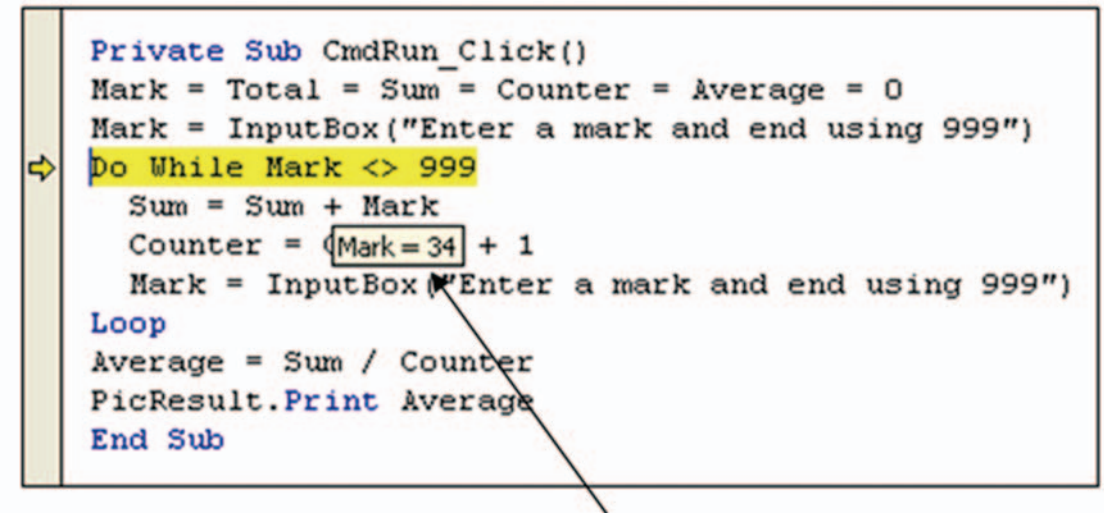


Figure 4.5: Visual Basic watch

Exercise

Practice creating 'watch' statements in a program either in Visual Basic or the programming language of your choice.

4.2.4 Breakpoints

A breakpoint is a marker set within the code of a program to halt program execution at a predefined spot. The statement or variable expression responsible will be highlighted and can be inspected while the program is temporarily interrupted. The program then continues, either to completion or until it hits another breakpoint.

Breakpoints are often used with traces. To set a breakpoint in Visual Basic, access the Debug menu.

Load any Visual Basic program and highlight a line of code or click in the margin. From the Debug menu choose `Toggle Breakpoint` and this will set a marker at that point.

Alternatively simply click in the margin opposite the line of code.

The example shown in Figure 4.6 is the Average marks program with a breakpoint added:

```
Private Sub CmdRun_Click()
Mark = Total = Sum = Counter = Average = 0
Mark = InputBox("Enter a mark and end using 999")
Do While Mark <> 999
  Sum = Sum + Mark
  Counter = Counter + 1
  Mark = InputBox("Enter a mark and end using 999")
Loop
Average = Sum / Counter
PicResult.Print Average
End Sub
```

Figure 4.6: Breakpoint set

When the program is run it will now pause at that position. To remove the breakpoint click on the coloured dot. By pressing function key F8 the program can be continued in 'Step Into' mode, line by line or the debug option may be cancelled by pressing function key F5.

Exercise

Practice creating 'breakpoints' in a program either in Visual Basic or the programming language of your choice.

4.2.5 Review questions

Q6: Explain the difference between the testing of software and debugging of software.

Q7: In programming a developer might make use of break points and watch statements. Explain the meaning of each of these terms.

Q8: Use a trace table to show how the values of the variable *Count*, *Number1* and the condition *Count* $>$ *4* when the following statement is executed:

```
Number1 = 6
Count = 0
Repeat
    Number1 = Number1  +  Count
    Count  =  Count  +  1
Until Count > 6
```

4.3 CASE tools

CASE is an acronym which stands for Computer Aided Software Engineering. It refers to collections of software programs that are designed to automate the various phases of the software development cycle.

The implementation of new systems requires many different complex tasks to be organised and completed correctly and efficiently. Information generated during the various phases has to be kept in synchronisation; a typical development environment requires that system design be closely related to resultant source code and be described by customary documentation, and that all of these areas be under centralised version control. The tools that support the individual tasks of design, coding, documentation, and version control must be integrated if they are to effectively support this kind of scenario.

Using previous 'paper and pen' techniques important information could very easily be lost between the software stages in a vast jumble of paper documentation, thus delaying progression of the software project.

The use of CASE tools eases the task of coordinating these activities from analysis right through to implementation.

Commercial CASE tools are now widely available and vary extensively in functionality and capability, but there are a set of features that are commonly found in most.

The following basic categories are typical of CASE tools available today:

1. Diagramming tools that represent data models according to system specifications.
2. Screen and Report Generators for creating system specifications.
3. Data Dictionaries that contain a history of changes made to a system
4. Code Generators to be able to generate code from data Diagrams themselves.
5. Documentation generators that make the code more readable.

4.3.1 Development of CASE tools

Since the early days of writing software, there has been an awareness of the need for automated tools to help the software developer. Initially the focus was on program support tools such as document production, translators, compilers, assemblers and so on. As computers became more powerful and the software that ran on them grew larger and more complex, powerful tools with increased functionality were therefore required. Figure 4.7 shows this development:

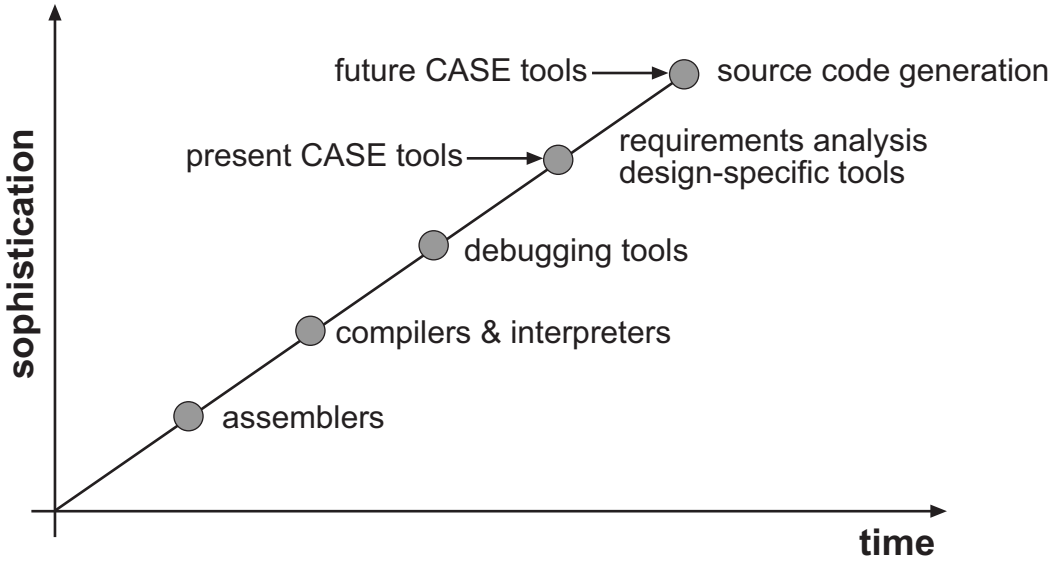


Figure 4.7: Development of CASE tools

It is said that the "holy grail" of CASE tools is to completely construct the source code directly from the requirements analysis but this is still a long way off!

4.3.2 Categories of CASE tools

CASE tools can be divided into two main groups - those that deal with the first three parts of the system development life cycle (preliminary investigation, analysis, and design) and are referred to as Front-End CASE tools or **Upper CASE tools**, and those that deal mainly with the implementation, testing and installation are referred to as Back-End CASE tools or **Lower CASE tools**. This is shown in Figure 4.8:

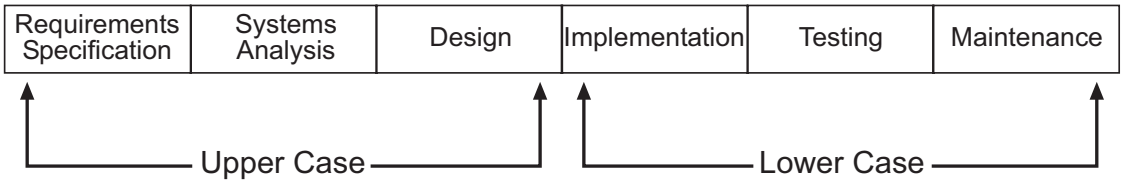


Figure 4.8: Categories of CASE tools

Developers tend to believe that the use of upper CASE tools have more importance than than the use of lower CASE tools in the software development process and that more time spent in the initial stages is beneficial to the overall process.

4.3.3 Upper (Front-End) CASE tools

These are basically general-purpose analysis and design specification tools.

During the initial stages of the system development, analysts are required to determine system requirements, and analyse this information to design the most effective system possible. This task is usually accomplished by an analyst using graphical methods such as data flow diagramming and structure charting techniques. Manual completion of these tasks makes it very tedious to have to redraw some of the diagrams each time a change is made to the system, for example. However small the change that is made to one diagram, this will probably require many changes to be made throughout all the existing documentation.

In very large systems, unless these changes are well documented, they could be lost or forgotten about thus leading to an erroneous representation of the system which, in turn might lead to significant problems during the implementation phase.

Computerised CASE tools would allow for these types of changes to be made automatically, very quickly and accurately. Information shared throughout the flowcharts and documentation stages would then be checked against each other to ensure that there is agreement.

Some large projects produce excessive amounts of documentation, some of it running into hundreds, if not thousands, of pages. Although development methodologies provide some structure to the documentation, even the best methodology and the most meticulous analyst cannot organise, index, and cross-index the information sufficiently to make it entirely useful. There is always the need to see the information differently from the way in which it is presented. Even with the best organisation and indexing methods, hardcopy documentation (printed on paper) is difficult to revise and still more difficult to keep up to date.

In order to overcome these and other purely mechanical problems involved with producing and maintaining proper documentation, many firms rely on automated data dictionaries. These data processing system products are specifically designed to hold, maintain, and organise analytical information; they come equipped with flexible facilities for producing a wide variety of reports on the dictionary contents.

A **Data dictionary** is an automated tool for collecting and organising the detailed information about system components. Data dictionaries maintain facilities to document data elements, records, programs, systems, files, users, and other system components. A dictionary will also have facilities to cross-reference all system components to each other and contain details of:

- systems environment;
- audit trails;
- reports;
- forms;
- functions;
- processes.

By using upper CASE tools developers can now be much more productive in the analysis and design stages of the development compared to using conventional paper and pen methods.

One of the ultimate goals of upper CASE tools is to refine the requirements analysis and design specification process to such an extent that most of the application code, around 75%, could be generated automatically. This would be a further enhancement to give analysts and designers more time in the initial stages of the development.

4.3.4 Lower (Back-End) CASE tools

Lower CASE tools focus on the architecture of the system and its implementation and maintenance. These tools are also effective in helping with the generation of the program code and are referred to as code generators.

A **code generator** is a tool that enables automatic generation of program code directly from analysis and design specifications including design documents, structure diagrams and reports. Generating code this way ensures that all the code is produced with identical naming conventions, documentation etc. No two developers could make that guarantee! See Figure 4.9.

In 2005 the automatic code generation process can produce about 45% - 50% of source code. The future aim is to increase this to 80% and beyond but this may take some time yet.

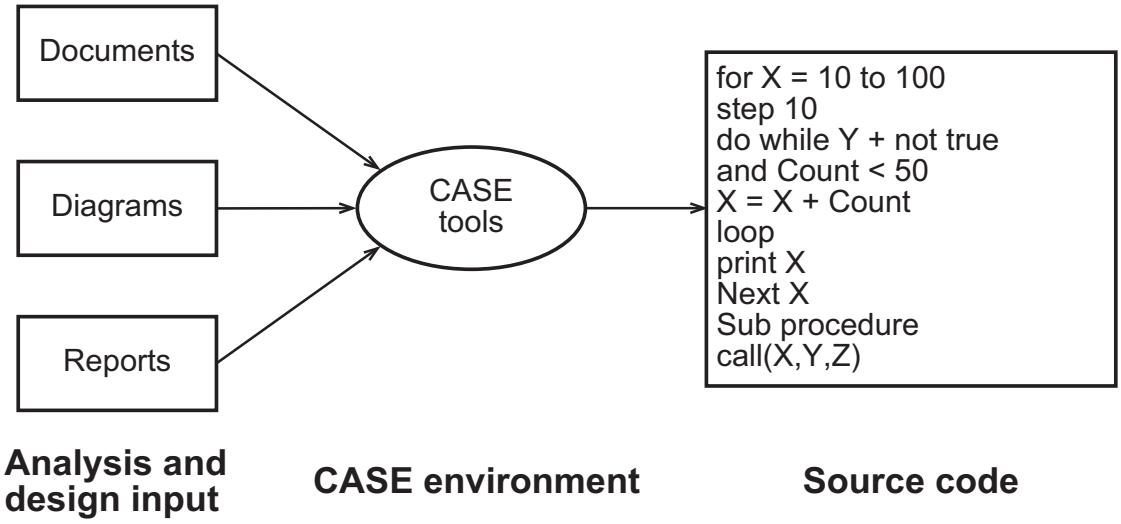


Figure 4.9: Code generation

Code generators also produce a high quality of code that is easy to maintain and is portable to different hardware platforms. They also have the feature that they are able to interact with the upper CASE tools. Information produced by the upper CASE tools can be accessed using the code generators to aid in the development of the code.

A further enhancement is the use of **document generators**.

A document generator is a CASE tool that generates technical documentation from source code comments. If this were to be done manually then problems would exist by different programmers using differing formats for their comments. However by using a CASE code generator the comments will automatically conform to a standard format.

This allows programmers to browse, edit, document and understand program source code in any specified language. Also the output can be in a variety of formats such as text or HTML.

4.3.5 Object-Oriented CASE

CASE tools are well-supported in object-oriented programming systems (OOPS). The principles of objects, classes and inheritance are fully supported and advocates of OOPS maintain that development issues become more apparent and easier to understand.

Earlier problems with OOPS focussed on the models created for software development systems; the models differed in notational techniques. This was overcome by the use of UML. The **Unified Modelling Language** (UML) created in 1995, became the new standard for producing diagrams and charts. All current CASE tools have now adopted UML. UML is used for specifying, visualising, constructing, and documenting the elements of software systems, as well as for business modelling and other non-software systems.

UML diagrams can be fairly complex entities. To view a selection of screen shots folow the following links:

http://www.visual-paradigm.com/sdevsScreenshots.php

http://pigseye.kennesaw.edu/~dbraun/csis4650/A_D/UML_tutorial/interaction.htm

Today CASE tools offer automatic code generation from the UML Diagrams created in the analysis phase. The CASE tool produces a framework for the code which contains objects and class definitions. They produce what is known as an **executable prototype**. This is an executable source code program obtained directly from the analysis Diagrams and specifications in UML. One of the main features of this process is **reverse engineering** where some tools analyse existing source code and reverse-engineer it into a set of UML Diagrams.

Figure 4.10 illustrates a typical object-oriented CASE environment:

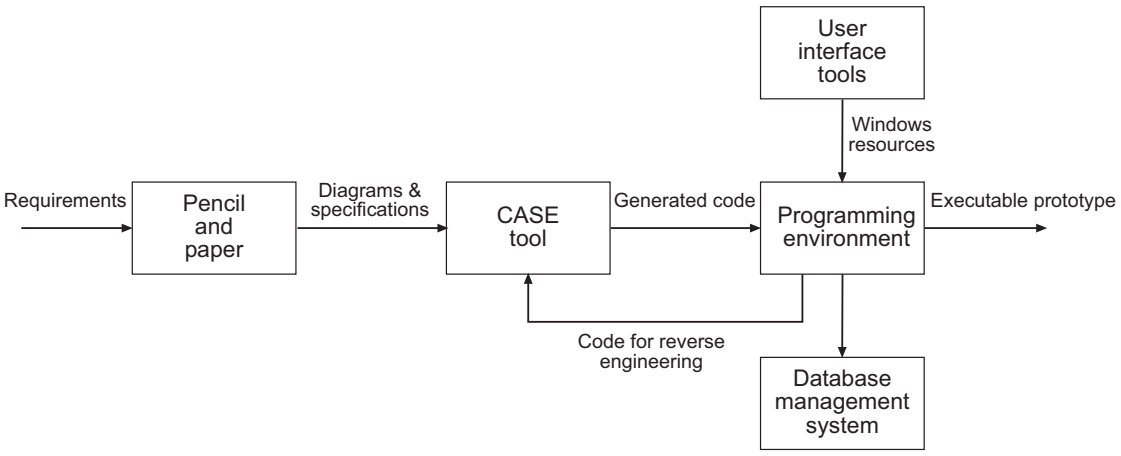


Figure 4.10: Object-oriented CASE environment

CASE tool developers claim to incorporate best practices into their products. In conjunction with UML techniques developers also employ **Rational Unified Process** (RUP) methodologies. This is a development process, now supported by IBM to deliver best proven practices during each stage of a project. Using RUP the risks are effectively lowered during all stages of the software development process as seen in Figure 4.11.

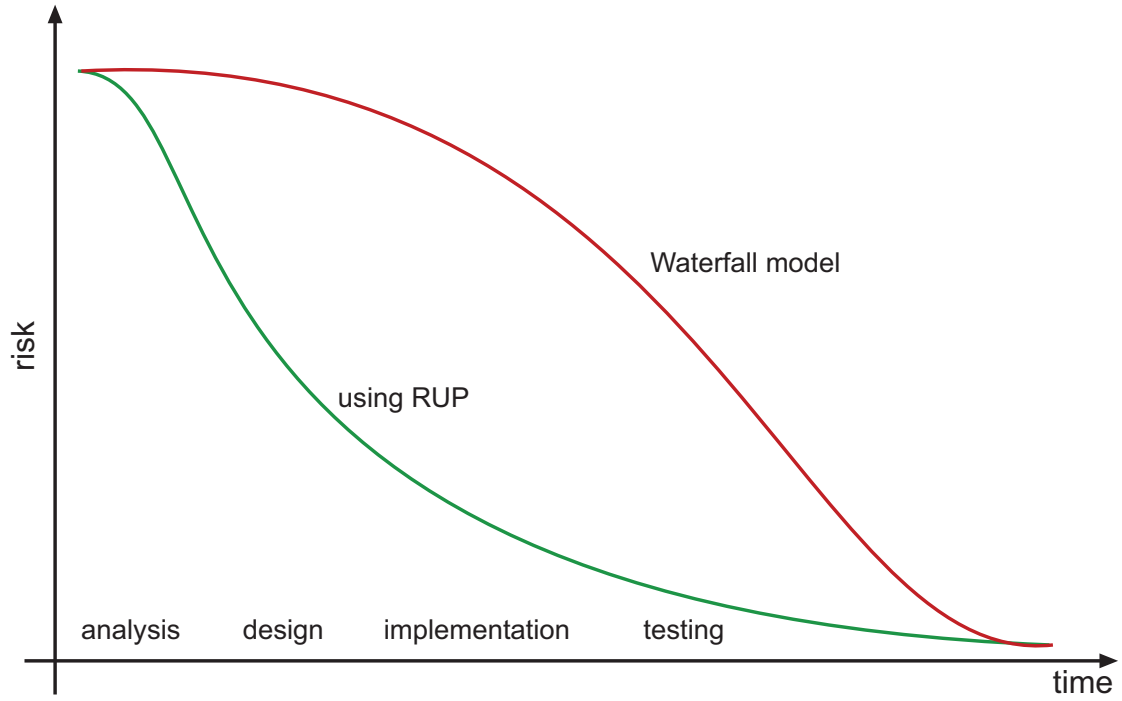


Figure 4.11: Lower Risks with Rational Unified Process (RUP)

4.3.6 Advantages of CASE Tools

1. **Increased Speed**

   CASE Tools provide automation and reduce the time to complete many tasks, especially those involving Diagramming and associated specifications at the design stage. This ultimately speeds up the entire development process and hence will increase productivity in the long term.

2. **Increased accuracy**

   CASE Tools can provide ongoing debugging and error checking which is vital for early defect removal. Less effort and time are consumed if corrections are made at an early stage of the development process such as the design stage. As the system grows larger, it becomes difficult to modify with error detection becoming increasingly more difficult and costly in terms of time and cost.

3. **Reduced costs and maintenance**

   As a result of better design, better analysis and automatic code generation, automatic testing and debugging, overall systems quality improves. There is also better documentation. Thus, the net effort and cost involved with maintenance is reduced. Also, more resources can be devoted to new systems development. CASE provides re-engineering tools which can be important because they make

this process more efficient and less expensive by determining which of the older parts of the system can be reused.

4. **Better Documentation**

   By using CASE Tools, vast amounts of documentation are produced along the way. Most tools have provision for comments and notes on systems development and maintenance. An important aspect of this is the **CASE Repository** or **encyclopaedia**. This takes the form of a developers' relational database which is regularly updated with information relating to software development projects. Such information could include specifications, codes and definitions that can be re-generated to form multiple diagrams. There will also be an audit trail of people, data, processes and technologies used that refer to a specific phase of a development project.

5. **Better communication**

   By using a set of CASE tools, information generated from one tool can be passed to other tools which, in turn, will use the information to complete its task, and then pass the new information back to the system to be used by other tools. This allows for important information to be passed efficiently and effectively between planning tools. Results of one stage should be available to another resulting in 'forward' integration. When using the old methods, incorrect information could be passed between designers or could simply be lost in the shuffle of papers.

4.3.7 Limitations of Case tools

Although CASE tools are becoming more popular in software development environments their uses can be subject to some limitations.

1. **Choice**

   It is estimated that over 1000 CASE tools are in existence today (2005) and the decision of which one will best fit a company's needs is not an easy one. The failure or success of the tool is relative to expectations. The evaluation and selection of a CASE tool is a major project in itself and should not be taken lightly. Time and resources need to be allocated to identifying the criteria on which the selection is to be based. Success with CASE will most likely occur when developers and managers choose tools based on methodologies similar to already in place within the organisation.

2. **Costs**

   CASE tools are not cheap! In fact, most firms engaged in software development on a small scale do not invest in CASE tools because they think that the benefits of CASE would be unjustifiable in cost terms alone. The cost of equipping every systems developer with a preferred CASE tool kit can be quite high. Hardware, systems software, training and consulting are all factors in the total cost equation. Although, there are an increasing number of good open source tools.

3. **Training**

   In most cases, programmer productivity may fall in the initial phase of implementation, because users need time to learn the technology. In fact, a CASE

consulting industry has evolved to support uses of CASE tools. The consultants offer training and on-site services that can be crucial to accelerating the learning curve and to the development and use of the tools.

CASE tools are a relatively new technology but have one of the highest growth rates compared to any segment in the computer industry. Many companies have begun buying copies of tools; in effect adopting CASE technology as their software productivity strategy. Progressive businesses are already changing how they think about developing, maintaining and enhancing their systems, and are finding strategic advantage in doing so.

For more information on CASE tools you may wish to use the following links:

http://www.uwm.edu/People/derek/course/Tools/CASE/keydfd.html

http://educ.queensu.ca/~compsci/units/casetools.html

http://www.cs.utexas.edu/users/almstrum/cs370/tlee/r1.htm

4.3.8 Review questions

Q9: What is a CASE tool and why were they developed?

Q10: Differentiate between upper CASE and lower CASE tools.

Q11: List **three** advantages that a software developer might find in using CASE tools and also **three** limitations that they might have.

4.4 Summary

The topic has described large scale testing procedures within software development and the various strategies employed. This can be a fairly extensive process. Debugging techniques offer important tools to the programmer including CASE tools that are now becoming more of an integral part of software development.

By the end of this topic you should now be aware of the following objectives:

- that software testing is a complex and expensive process involving many procedures before the final product is released to the client;
- that debugging tools in software development can aid the programmer to locate and fix errors in programming;
- that CASE tools are becoming increasingly accepted into most aspects of software development.

End of topic test

Q12: Alpha testing is performed on the client's premises.

a) True
b) False

Q13: Alpha testing finds errors that can only be found under operational conditions.

a) True
b) False

Q14: Alpha testing is performed on the finished product prior to implementation.

a) True
b) False

Q15: The best tool for a programmer to test the logic of a program is:

a) Dry run
b) Breakpoint
c) Watch statement
d) None of these

Q16: Give an example of an error that debugging software will not be able to find

Q17: State the main purpose of a CASE tool.

Q18: Which one of the following is associated with modern case tools?

a) Unified modellling language
b) Executable prototype
c) Reverse engineering
d) All of the above

Q19: CASE tools are not in widespread use because they are too expensive

a) True
b) False

Topic 5

High level programming language constructs 1

Contents

Learning Objectives

After studying this topic, you should be able to:

- *Describe and exemplify the following constructs in pseudocode and an appropriate high level language:*
  - *Files: File handling: write, read, delete item, create new file;*
- *Describe and exemplify the following variable types:*
  - *2-D arrays.*

The practical aspects of this topic extend to file handling skills where sequential and random types are explained and implemented. The topic also expands on the work done in Higher Computing in dealing with static data structures, arrays. The concept of 2-dimensional arrays are introduced and implemented in both pseudo code and a high level language..

5.1 File handling

An important aspect of any programming language is its ability to access and manipulate files as part of its I/O system.

There are three types of data file that can be used to store information, namely:

1. sequential
2. random
3. binary

In this topic emphasis will be focussed on sequential files since they are more straightforward to implement in a programming language.

Random access files are dealt with later in the topic when the data structure is discussed. Manipulation of records requires the use of random access files for the tasks of creating, reading and writing.

Binary files are similar to sequential files but the data types can vary. They may contain characters, numerical data or the contents of an executable file. Images can also be stored in binary files. Although they offer greater flexibility the files have no defined structure and are somewhat more difficult to program. They will not be discussed further.

5.1.1 Sequential files

Sequential (text) files have a universal standard format and are used extensively in simple text editors. The Windows *Notepad* application, for example creates simple text files. Even numerical data is stored as a string, for example 5.76 would be stored as "5.76". Such files use disc space efficiently but are difficult to update - hence they are only used for text.

Sequential files are read from beginning to end and so the files cannot be read and written to simultaneously.

A sequential data file can be thought of as a 1-dimensional array with each array location storing a single byte of data, such as an ASCII character. As an example, the data referring to a name and telephone number, `TOD A. 225 3625`, would be stored in memory as:

"	T	O	D	"		A	.	"	,	"	2	2	5		3	6	2	5	"	¶	EOF

The name is contained in a field which is separated from the telephone field using a comma.

There are also hidden ASCII control codes: the end of the field is signified by a paragraph symbol. This represents both a carriage return (CR) and linefeed (LF), equivalent to pressing the Return key in an application like a word processor. The EOF marks the end of the file.

The complete information represents a data structure called a **record** and this will be discussed later.

The manipulation of files involves three stages as shown in Table 5.1:

Table 5.1: File processes

Opening a file	If the file does not exist then it is created and then opened by the operating system, which reserves a portion of memory for the file.
Processing a file	Once a file is opened it can be written to or read from, or both in the case of random and binary files. Writing to a file will save it to backing store.
Closing a file	Once a file has been opened and processed it must then be closed. When a file is closed the operating system releases the memory that it reserved for the file.

In Visual Basic a file is opened or created using the Open command. This is achieved by using the statement:

```
Open "FileName" For FileType As #ChannelNumber
```

Each file created must have a file name and a file number for identification.

The command opens a file called `FileName`, the name of the file to be opened or created and allocates it a `ChannelNumber` for identification. Every file that is opened has a different channel number, starting at 1 and increasing by 1 for every file that is opened. Visual Basic does this automatically. The use of the hash symbol, `#` is a Visual Basic directive to the computer filing system.

The argument `FileType` determines what type of file is being processed and also the type of access. Table 5.2 shows the options:

Table 5.2: File options

Input	The file is opened for input which is read only.
Output	The file is opened for output which is write only or create.
Append	The file is opened for adding new data to an existing file. The default setting in Visual Basic.
Random	The file is opened for random access which is reading or writing one record at a time.
Binary	The file is opened in binary mode.

The first three options refer to sequential files only. Note that if the argument is missed out in the statement then `Random` is assumed by default.

You can also specify the path where the file will reside on the hard drive. For example, the statement:

```
Open "c:\My Documents\trial.txt" For Output As #1
```

will create a text file called `trial.txt` in the 'My Documents' folder on the C: drive.

5.1.2 Manipulating sequential files

Although the file options `create`, `open`, `read`, `write` and `delete` are all explained separately, the code for each form part of a complete program that can be seen in Code 5.4. The program form is shown in Figure 5.1. You may wish create this first before proceeding with the code.

Opening (creating) a sequential file

Before data can be written to a sequential file, the file must be created. Characters are then 'printed' to the file using a simple text window and the file is then closed. The following algorithm will achieve this:

```
1. Enter filename
2. Open file for writing
3. Input information
4. Save to file
5. Close file
```

The corresponding Visual Basic code is shown in Code 5.1. To show the output a text box is used.

```
Option Explicit
Dim Filename As String
'File manipulation program
'Create file
Private Sub cmdCreateFile_Click()
Filename = InputBox$("Please enter a filename")
Open Filename For Output As #1          'Create file
Print #1, Text1.Text
Close #1
End Sub
```

Code 5.1: Open(create) file

The program will create a file and open it ready for input of text in the text box, `Text1.Text` 'Filename' is any valid name that is input by the user via the `InputBox$` function.

Each time text is entered it can be saved to the same file, in which case existing data will be over-written. Alternatively it can be saved as a new file.

NB. In Visual Basic if an existing file is opened using the `Output` option then the contents will be wiped even if nothing is written to it!

This is where the `Append` option becomes useful. By using `Append` a file can be opened and data added to it. The original data cannot be changed.

Reading a text file

Once a file has been created using the program in Code 5.1, it can be read at any time. The following algorithm will achieve this:

```
1. Enter filename
2. Open file for reading using input
3. Close file
```

The corresponding Visual Basic code is shown in Code 5.2:

```
Private Sub cmdRead_Click()
 Filename = InputBox$("Please input a filename")
 Open Filename For Input As #1                    'Read file
 Text1.Text = Input$(LOF(1), 1)                   'Length of file
 Close #1
End Sub
```

Code 5.2: Read file

The program will open the file assigned to the string `Filename` for input through `channel 1`. The contents of the file are read into the `text1` window using the `Input$` statement. The `LOF(1)` part gives the length of the file to be input, a single character at a time. Finally the file is closed.

NB. When you create the text window `Text1.Text` ensure that the property `Multiline` option is set to `True`. This ensures that text wrapping occurs within the text window.

Deleting a file

Any file may be permanently deleted using the `Kill` statement within Visual Basic. The file is not sent to the recycle bin.

The following algorithm will achieve this:

```
1. Get file
2. Delete file? Yes/No
3. If 'yes' then
4. Kill file
```

The VIsual Basic code for this is shown in Code 5.3:

```
Dim Response As String
 Response = MsgBox("Are you sure you want to delete", vbYesNo)
 If Response = vbYes Then
 Kill Filename
End If
```

Code 5.3: Delete file

The variable `Response` will take the value `Yes` or `No` depending on user input. Note the Visual Basic construct `vbYesNo` that is identical to text input "Yes" and "No" which saves time!

If the response is `Yes` then the file is permanently deleted.

To test this out, incorporate the above code and associate this with the button called `Delete`. Now run through the following steps:

1. Create file and give it a name
2. Type in some text
3. Close the file
4. Clear the text box and load the saved program. Your text should appear.
5. Delete the file and clear text box
6. Load file that you saved

A run-time error should occur because the file cannot be found; it has been deleted. Success!

Here is the completed Visual Basic program, Code 5.4, containing all the file features:

```
Option Explicit
Dim Filename As String

'**************************************
'Sequential File manipulation program   *
'This program illustrates file commands *
'                                       *
'November 2004                          *
'**************************************

Private Sub cmdOPEN_Click()
'OPEN FILE FOR WRITING
 Filename = InputBox$("Please enter a filename")
 Open Filename For Output As #1
 Print #1, Text1.Text
 Close #1
End Sub

Private Sub CmdREAD_Click()
'OPEN FILE FOR READING
 Filename = InputBox$("Please input a filename")
 Open Filename For Input As #1
 Text1.Text = Input$(LOF(1), 1)
 Close #1
End Sub

Private Sub CmdDelete_Click()
'DELETE FILE
 Dim Response As String
 Response = MsgBox("Are you sure you want to delete", vbYesNo)
 If Response = vbYes Then
  Kill Filename
 End If
```

```
End Sub

Private Sub cmdClear_Click()
'Clear text box
 Text1.Text = " "
End Sub

Private Sub cmdExit_Click()
'End program
 End
End Sub
```

Code 5.4: Complete file program

A screenshot from the program is shown in Figure 5.1.

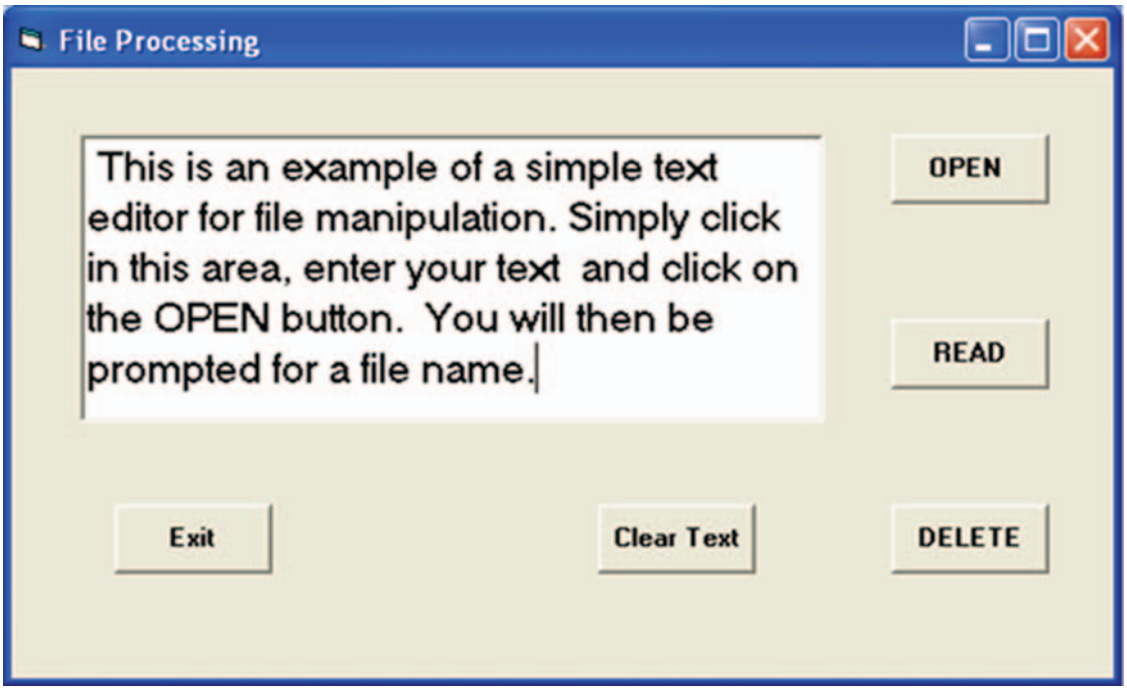


Figure 5.1: Program output

Exercise

Extend the program by including the `Append` option as a new button on the form. The code for this is shown in Code 5.5:

```
Private Sub CmdAPPEND_Click()
'OPEN FILE FOR APPENDING
 Filename = InputBox$("Please input a filename")
 Open Filename For Append As #1
 Print #1, Text1.Text
 Close #1
End Sub
```

Code 5.5: Append option

When you use the Append option you will not see much happening! It has to be done in the correct order as foillows:

1. Run program and enter some text
2. Click on Open
3. Clear text
4. Click on Read to ensure file exists
5. Enter further information and click on Append
6. Clear text area
7. Click on Read and the updated file should appear

You should now be able to create, write to, read from and delete a sequential file.

Exercise

One of the limitations of the previous program is that if you forget the name of the file to open or append then you need to go searching! If there was some way to catalogue the disc area then that would allow you to see the files that have been saved.

By using a Windows option called the `CommonDialog` filter, a directory listing may be achieved. The `CommonDialog` filter is a Windows construct that allows Visual Basic programs to interact with the Window operating system.

For this to happen the CommonDialog control box has to be added to the project form. This is done as follows:

1. Open Visual Basic
2. Access the Toolbox from the menu
3. Choose Components from the Project menu
4. In the Controls tab tick the box next to Microsoft CommonDialog and click on OK.

You will see a new icon in the `Toolbox`: 

5. Drag this icon on to the form and it will work away in the background

In Visual Basic open a new project and type in the code as shown in Code 5.6:

```
Option Explicit
Dim Filename As String
Dim Channel_Number As Integer

'*****************************************************
'Program for file manipulation using Windows interface *
'                                                      *
'November 2004                                         *
'*****************************************************

Private Sub cmdLoad_Click()
'Using the OPEN dialogue box
 CommonDialog1.Filter = "All files (*.*)
           |*.*|Text_files(*.txt)|*.txt|VB_Text_files(*.frx)|*.frx"
 CommonDialog1.ShowOpen
 Filename = CommonDialog1.Filename
 Open Filename For Input As #1
 Text1.Text = Input$(LOF(1), 1)
 Close #1
End Sub

Private Sub CmdSave_Click()
'Using SAVE AS dialogue box
 CommonDialog1.Filter = "All files (*.*) |*.*|Text_files(*.txt)|*.txt"
 CommonDialog1.ShowSave
 Filename = CommonDialog1.Filename
 Open Filename For Output As #1
 Print #1, Text1.Text
 Close #1
End Sub

Private Sub CmdClear_Click()
 Text1.Text = " "
End Sub

Private Sub CmdExit_Click()
 End
End Sub
```

Code 5.6: Files under Windows

The program contains two main lines of code to load (open) and read a file.

The `CommonDialog1.Filter` is used to set the type of files to be catalogued. In this case most options are given with each type separated by the pipe character, '|'. You may choose your own file types to view.

The lines `CommonDialog1.ShowOpen` and `CommonDialog1.ShowSave` determines which Windows dialogue option is used i.e. `Open` and `Save`.

Finally the `Filename` is prefixed with `CommonDialog1` to ensure the correct path when loading and saving the file.

Figure 5.2 shows the directory access in action, loading the file called `Planets.txt` that you met in a previous topic:

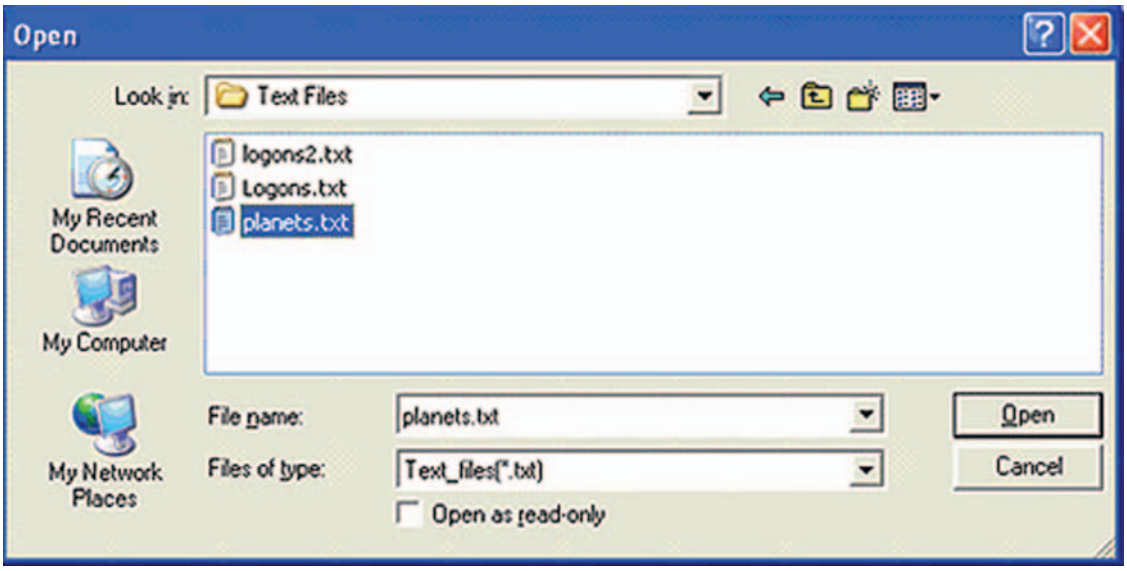


Figure 5.2: CommonDialgue in action

Figure 5.3 shows the program run.

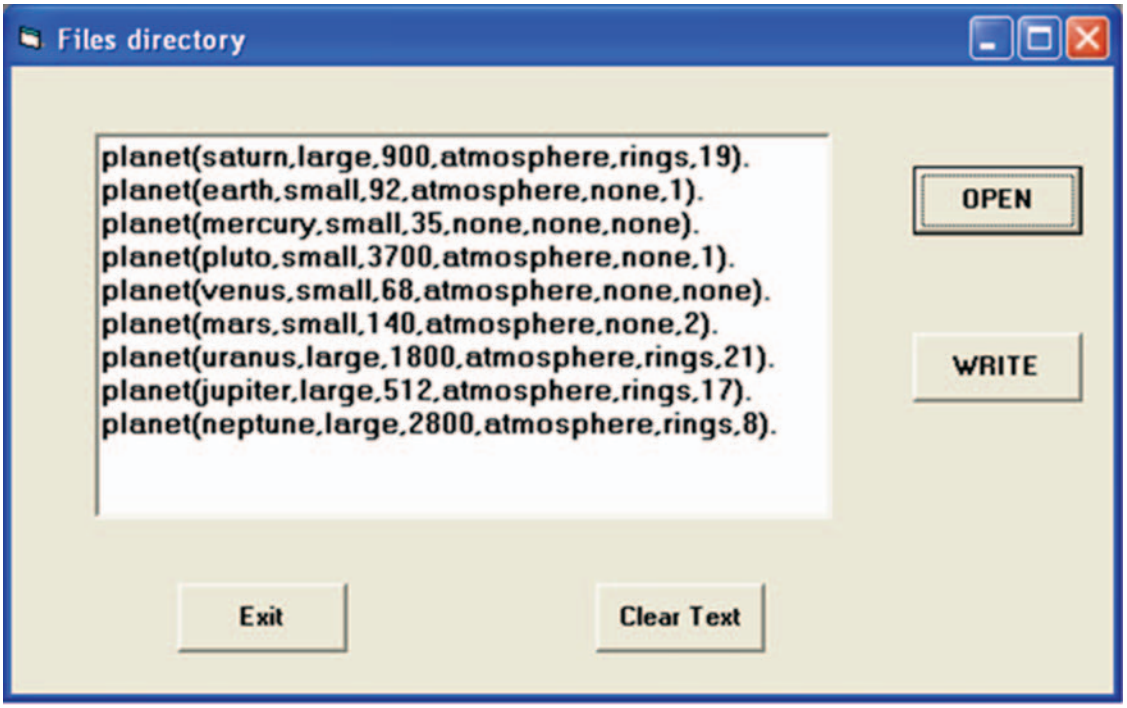


Figure 5.3: Program run

Run the program and experiment with files of your choice. Remember to set the `multiline` property of he text box to `true` to allow text wrapping.

5.1.3 Review questions

Q1: What is a sequential file?

Q2: Explain the processes of creating a sequential file and reading a sequential file.

Q3: Show in the form of a diagram how the following information, relating to golf handicaps would be stored as a sequential file in computer memory:

Vardon Hubert: 28

Sayers Bill: 6

Player Gill: 20

Trevino Louis: 1

5.2 Arrays

In the Higher Computing course we looked at the 1-dimensional array, also referred to as a *vector* or *linear list*. They could store a single row or column of values. Such arrays are useful for holding lists of data that can easily be manipulated during, for example, searching and sorting of data.

For example in Visual Basic an array called `Month` containing 12 data items as shown in Table 5.3 would be declared in the form

```
Dim Month(0,11) As Integer
```

The data actually relates to the average monthly temperatures in the Mediterranean, beginning with January = 15°C in location `Month(0)`.

Table 5.3: Average monthly temperatures

15	15	16	19	23	28	30	31	28	24	20	17

In software applications arrays with two or more dimensions can exist in computer memory. Remembering that computer memory is made up of contiguous memory locations multi-dimensional arrays can only exist as linear lists.

Here we will only consider 2-dimensional arrays.

5.2.1 2-Dimensional Arrays

A 2-dimensional array or a **matrix** is a table of 1-dimensional arrays the size of which depends upon the number of *rows* and *columns*.

As an example consider the following data relating to weather information that was collected during a school outdoor project week, which is stored in a **9 x 2** array i.e 18 locations. Data was collected and stored over an eight-day period in the array called Weather_records as shown in Table 5.4:

Table 5.4: Dimensional array - Weather_ records

Data 26/05/04	**Site 1** **Inside forest**	**Site 2** **Outside forest**
Maximum temperature (°C)	12	13
Minimum temperature (°C)	6	7
Wet temperature (°C)	10	11
Dry temperature (°C)	12	13
Humidity (%)	67	74
Soil temperature at 2cm (°C)	11	11
Soil temperature at 10cm (°C)	10	9
Rainfall (mm)	0.1	0.25
Light intensity (lux)	540	8440

The array now requires two index numbers to access any element.

Each data item or element of the array can be uniquely identified using two index numbers, the same as in a spreadsheet, by looking at the cell reference - Row number and Column number. In the array `Weather_records`, the array name and (*Row*, *Column*) values make up the cell reference.

For example, the value `6`, representing the temperature at `Site 1`, is in location `Weather_records (1,0)`.

Consider now a more general example. In Figure 5.4 an array of dimensions 3 x 4 called `Primes` has been created containing the first 12 prime numbers.

		column index			
		0	**1**	**2**	**3**
row index	**0**	1	2	3	5
	1	7	11	13	17
	2	19	23	29	31

Figure 5.4: 2-Dimensional array

The value `17` is stored in location `Primes(1,3)`

Location `Primes(0,0)` contains the value `1`

Exercise

Check and verify the following statements:

The location `Primes(0,2)` contains the value `3`

The location `Primes(2,1)` contains the value `23`

The value `31` is stored in location `Primes(2,3)`

The value `19` is stored in location `Primes(2,0)`

Exercise

With reference to the array Weather_records (Table 5.4):

Q4: What is the location of the value 67?

Q5: What value is held in location(0,1)?

Arrays Interaction for Two-Dimensional Arrays

There is an interactivity online to demonstrate two-dimensional arrays

5.2.2 Implementation of a 2-dimensional array

We will consider the following actions on 2-dimensional arrays:

1. initialise the array
2. read data into the array
3. output the contents of an array
4. search the array for data items
5. sort data items

5.2.3 Initialising an array

Recall that initialising an array simply means setting all the array elements to zero, null or a specified value. For small arrays this can be done by assigning the data to the array locations. For a 2 x 3 array called `List` this would be:

```
List(0,0) = 0
List(0,1) = 0
List(0,2) = 0
List(1,0) = 0
List(1,1) = 0
List(1,2) = 0
```

For larger arrays this is best achieved by implementing a loop structure in the programming language.

For a 2-dimensional array, two such loops would be required, one to deal with the rows and the other for the columns.

For example to initialise a 6 x 6 array called `Numbers` the following general algorithm can be employed that uses two `for` loops, since the array boundaries are known:

```
1.  For Row_index = 0 to 5
2.   For Column_Index = 0 to 5
3.      Numbers(Row_index,Column_Index) = 0
4.   Next Column_index
5.  Next Row_Index
```

Exercise

In a programming language, code a program to initialise the 2-dimension array `Numbers` using the algorithm below.

```
1.  For Row_index = 0 to 5
2.   For Column_Index = 0 to 5
3.      Numbers(Row_index,Column_Index) = 0
4.   Next Column_index
5.  Next Row_Index
```

5.2.4 Reading data into an array

Whereas initialisation sets all array elements to be identical, the same method can be used to input data. For example in the previous 3 x 4 array example called *Primes* we can input the prime values into the array using the following algorithm:

```
1. For row = 0 to 2
2.  For column = 0 to 3
3.   Prime(row,value) = Input value {1 2 3 5 7 11 13 17 19 23 29 31}
4.  Next column
5. Next row
```

Exercise

Code a program to input data into the 2-dimension array `Prime` using the algorithm below.

```
1. For row = 0 to 2
2.  For column = 0 to 3
3.   Prime(row,value) = Input value {1 2 3 5 7 11 13 17 19 23 29 31}
4.  Next column
5. Next row
```

When you implement this code you won't see much happening since data is only being read into the array. There is no output.

5.2.5 Outputting array data

With a 1-dimensional array there is only one output, namely a linear list. With a 2-dimensional array, such as Primes, however we would like the output to be in tabular

form showing the data in rows and columns, instead of what would be produced without any formatting such as:

```
1 2 3 5 7 11 13 15 17 19 23 29 31
```

The following algorithm, using formatting commands will accomplish this:

```
1. For Column = 0 To 2
2.  For Row = 0 To 3
3.   Print Tab(Row*5); Prime(Column, Row); "    ";
4.  Next Row
5.  Print
6.  Print
7. Next Column
```

Lines 1 and 2 are the normal loop structures to access the data.

Line 3 contains formatting to give more spacing between the values. The `Tab` function takes the value `Row` on each pass through the loop and by multiplying this by a value, in this case `5`, produces uniform spacing of the output. This has been achieved by trial and error. The semicolon at the end of line 3 suppresses output to a new line.

Lines 5 is required to direct output to a new line once the first row of values have been output. Line 6 is purely cosmetic to produce a more pleasing result.

The complete Visual Basic program code is shown in Code 5.7:

```
Option Explicit
'*********************************************************
'This program will input values to a 2-dimensional array *
'and output the contents                                  *
'                                                         *
'November 2004                                            *
'*********************************************************
Dim Prime(3, 4) As Integer
Dim Row As Integer
Dim Column As Integer
'Input values
Private Sub CmdRUN_Click()
For Column = 0 To 2
   For Row = 0 To 3
     Prime(Column, Row) = InputBox("please enter values")
   Next Row
Next Column
'Output results
For Column = 0 To 2
   For Row = 0 To 3
     PicOutput.Print Tab(Row * 5); Prime(Column, Row);
   Next Row
   PicOutput.Print
   PicOutput.Print
Next Column
```

```
End Sub
```

Code 5.7: Formatted output

Figure 5.5 shows sample program output:

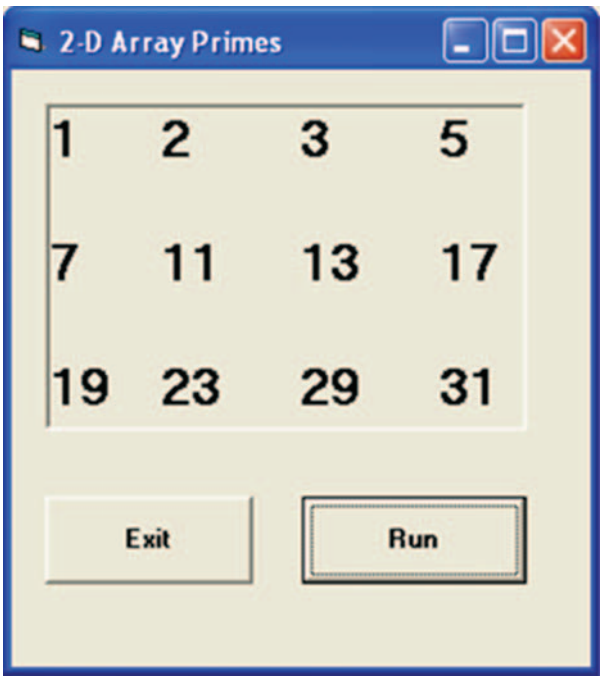


Figure 5.5: Program output

You may like to experiment with `Tab` values and font sizes to obtain the output that you prefer.

Extension work

In the Higher Computing course one of the examples was to read the days of the week into an array called `Days`.

Suppose we want to extend this to hold the dates for a month as well and storing the results in a 2-dimensional array.

Problem: Create a program that will input the days of the week and also the dates for the month of `May 2005` and store the results in a 2-dimensional array called `month`. The output is to be in the form of Table 5.5 :

Table 5.5: Days of the month, May 2005

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Exercise

What changes would have to be done to the `Month` program to output dates for the month of `February 2005`?

5.2.6 Review questions

Q6: Explain the importance of 2-dimensional arrays. By referring to newspapers, magazines television etc., list **five** examples where 2-dimensional arrays feature.

Q7: Explain the process of initialising a 3 x 4, 2-dimensional array.

5.3 Summary

The topic has introduced file handling techniques and the concept of 2-dimensional arrays and their uses in computing.

By the end of this topic you should be aware of the following objectives:

- appreciate file handling techniques such as create, write, read and delete;
- describe the structure and uses of 2-dimensional arrays;
- implement the structures in pseudocode and a high level language.

End of topic test

Q8: Which one of the following statements regarding sequential files is true?

a) They contain text only and access is fast
b) They contain text and graphics but access is slow
c) They contain text only and access is slow
d) They contain text and graphics and access is fast

Q9: For every file that is opened it must also be:

a) Written to
b) Closed
c) Read from
d) Deleted

Q10: A sequential file can be read and written to but not simultaneously. Why is this case?

Q11: Consider the following data that has to be stored in a suitable data structure:

London, Aberdeen, Glasgow, Bath, Portsmouth, Inverness, Dundee, Oxford, Ayr

If the data were to be searched and sorted then the most convenient data structure would be:

a) 1-dimensional array
b) Sequential file
c) 2-dimensional array
d) None of these

Q12: Information relating to the data has now to be added. To allow for this the most convenient data structure would be:

a) 1-dimensional array
b) Sequential file
c) 2-dimensional array
d) None of these

Q13: The following questions refer to the 2-dimensional array:

6	9	0	3	1
12	0	4	10	2
5	11	7	15	8

Assuming indexing begins at 0, the array would be declared as:

a) Array(5,3)
b) Array(2,4)
c) Array(4,2)
d) Array(3,5)

Q14: The value 10 is held in location:

a) Array(1,3)
b) Array(2,4)
c) Array(3,1)
d) Array(4,2)

Q15: The location Array(2,3) contains the value:

a) 4
b) 7
c) 10
d) 15

Q16: Why are arrays initialised?

Q17: It is impossible to have arrays with more than two dimensions.

a) True
b) False

Topic 6

High level programming language constructs 2: Data structures

Contents

Learning Objectives

After studying this topic, you should be able to:

- *Describe and exemplify the following data constructs:*
  - *stack, queue and record*

This topic introduces two important dynamic data types, namely the stack and the queue and discusses their uses in computing. Further file handling techniques are covered with specific reference to the data structure called a record.

Static data structures, like arrays have a fixed size in memory. Their size is dependent upon the amount of data that they store. Dynamic data structures however can increase and decrease in size during the execution of a program. Two such data structures, important in computing are the stack and the queue, often used in programming languages for the temporary storage of data. They can be considered as special cases of linear lists or 1-dimensional arrays.

6.1 The stack

The stack is an extremely important data structure in the workings of a computer.

The stack is useful where a situation calls for the most recent item to be accessed such as in programming where procedures are called. Each call is placed on the stack pending processing. They are also used to store the code produced during the compilation of high level languages and also to store the intermediate results of calculations. When a running program is interrupted the status of the program and the contents of all the registers are stored on top of a stack.

A stack may be represented in computer memory as a 1-dimentional array with the bottom at a fixed location and a variable **stack pointer** to the current top location which is movable. The stack pointer is a special register that is updated each time the contents of the stack changes.

Data elements can be added or deleted only from the top of the stack, akin to a real pile of plates or coins. Figure 6.1 illustrates the concept:

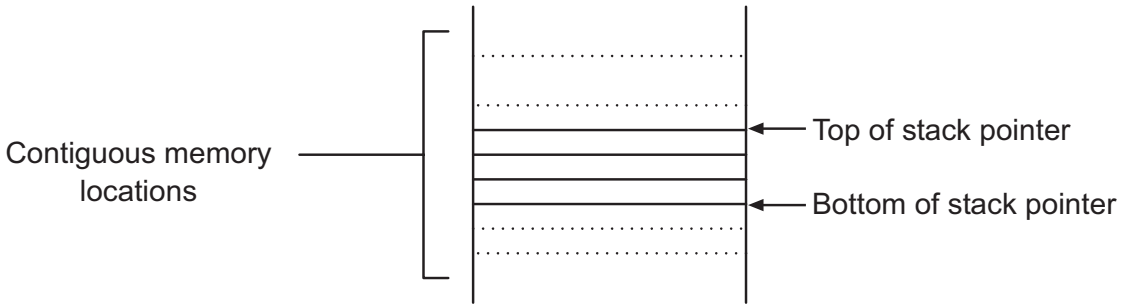


Figure 6.1: Representation of a stack

Customarily a stack has two operations associated with it:

1. **Push**: an item is added to the top of the stack, increasing the stack size by 1.
2. **Pop**: the top item is taken from the stack, decreasing the stack size by 1.

Examples

1. Suppose the following Push operations were performed with variables A, B and C. Figure 6.2 illustrates the situation:

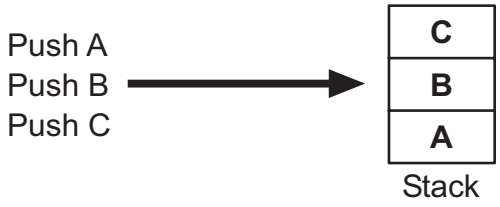
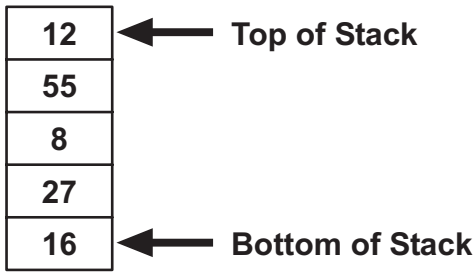


Figure 6.2: Stack operations

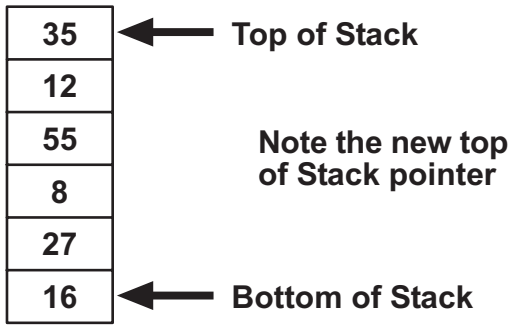
If you then did three `Pops`, you would retrieve the variables in the order:

```
C  B  A
```

2. Consider the integers `16,27,8,55 and 12`. Pushing them into a stack in the order given would produce:



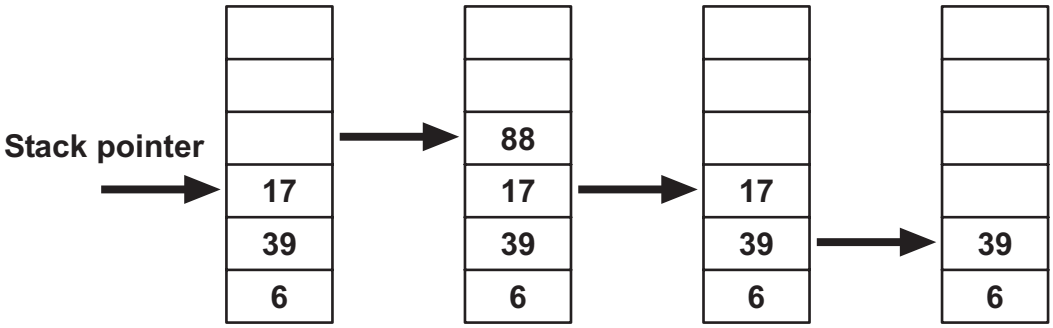
If the number 35 is to be added to the list then it is *pushed* onto the top of the stack, the situation now looks like:



Using this system the last number in is always the first number out. A stack is therefore called a **LIFO** structure (**L**ast **I**n **F**irst **O**ut).

Exercise 1

Consider the following stack sequence:



Explain the stack operations in terms of push, pop and pointer changes.

Stack underflow

In Exercise 1, if a further two pop operations were executed, the top of stack would become less than the bottom of the stack. The stack is now empty. Attempting any further pop operations before any new data have been pushed onto the stack is an error known as **stack underflow**. Any stack operations should test whether a further pop would produce stack underflow and report an error if necessary.

Stack overflow

As a stack size is usually limited, this may produce a **stack overflow** if the maximum size is exceeded. As with stack underflow it must never be allowed to happen otherwise termination of a computer program or application will occur.

The use of recursion imposes significant overheads on stack manipulation, in some cases leading to stack overflow. The LOGO teaching language relies heavily on recursion as does nested procedure and subroutine calls during program execution.

A number of computer languages are stack-oriented where they use the stack for basic arithmetic operations and temporary storage of variables. Declarative and event-driven languages are prime examples.

Exercise 2

Q1: Given the input stream A, B, C, D, E, F write down the sequence of stack operations (Push for stack, Pop for unstack) which would produce the output stream C, B, D, E, F, A

Q2: For each of the six data items 1, 2, 3, 4, 5, 6 on the input stream, which of the following permutations are possible output streams using a stack?

1. 1, 2, 3, 4, 5, 6
2. 2, 4, 3, 6, 5, 1
3. 1, 5, 2, 4, 3, 6
4. 4, 2, 1, 3, 5, 6
5. 1, 2, 6, 4, 5, 3
6. 5, 2, 6, 3, 4, 1

6.2 Implementation of a stack

The following algorithms illustrate the two processes of pushing and popping data items. In each case testing for the conditions of stack full (overflow) and stack empty (underflow) must be implemented otherwise the running program may terminate abnormally.

Push a new item on to a stack

```
If Stack_pointer > Maximum then
  Output 'Stack Overflow'
Else
  Stack_pointer = Stack_pointer + 1
  Stack(Stack_pointer) = Data item
EndIf
```

Pop an item off the stack

```
If Stack_pointer < Minimum then
  Output 'Stack Underflow'
Else
  Data item = Stack(Stack_pointer)
  Stack_pointer = Stack_pointer - 1
EndIf
```

Extension Work: Reverse Polish notation

If you look at some modern calculators on sale today many do not have an '=' sign button. Neither do computers! Arithmetic expressions are worked out in a logical fashion using **Reverse Polish**, a name derived from Jan Lukasiewicz who was a Polish mathematician. The stack is ideally suited to this form of calculation.

Consider the simple addition of the two integers 5 and 7 to give 12.

In a simple calculator you would perform the steps: `5 + 7 =`

In a graphics calculator the steps would be: `5 7 +`

The latter expression is in Reverse Polish, also referred to as **postfix notation** since the operator (+) comes after the data. In terms of stack operations it will look like:

```
Push 5
Push 7
Push +
```

The expression is read from left to right and once an operator is pushed on to the stack the calculation is performed. The top of the stack will always contain the result of the arithmetic calculation as shown in Figure 6.3 :

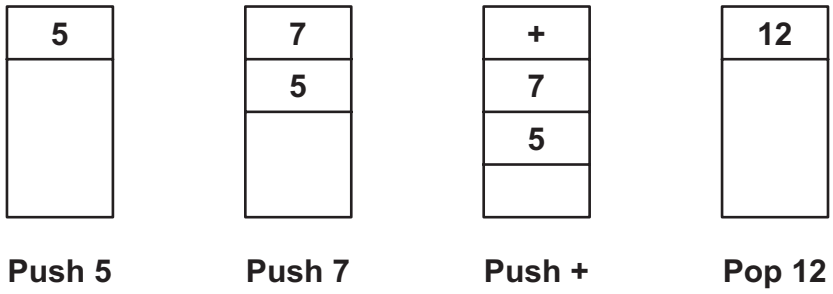


Figure 6.3: Stack operation

Now consider the more complex expression `(6 + 2) x (5 - 1)`

Using the rules of precedence i.e. *brackets, exponentiation, multiplication and division, addition* and *subtraction*, the sequence would be as follows:

```
Push 6
Push 2
Push + to give 8 on top of stack
Push 5
Push 1
Push - to give 4 on top of stack
Push x to give 32 on top of the stack
```

Thus the expression `(6 + 2) x (5 - 1)` in Reverse Polish is: `6 2 + 5 1 - x`

Convert the following expressions into Reverse Polish:

Q3: `((a + b)/c) + (d - e)`

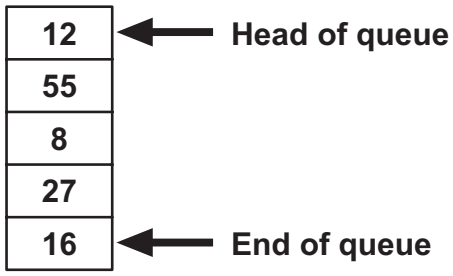
Q4: `9 + (7 - 3) x (4 + 5)/2`

6.3 The queue

A queue is also a 1-dimensional array (linear list) similar in structure to a stack but data items are inserted and deleted at different ends. Using this system the first number in becomes the first number out. It is therefore referred to as a **FIFO** structure (**F**irst **I**n **F**irst **O**ut).

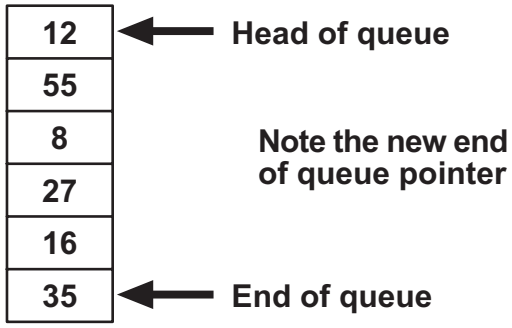
Two pointers are used, one to point to the head of the queue and another to point to the end of the queue.

Consider the numbers `12, 55, 8, 27, 16`. Putting them into a queue the list now becomes:

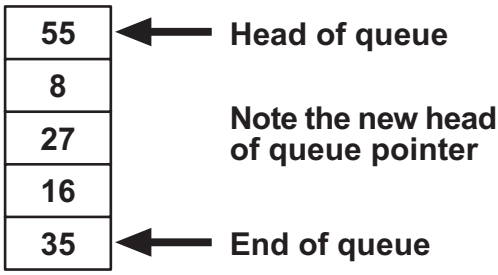


If the number 35 is to be added then it joins the end of the queue (**pushed**).

The queue now becomes:

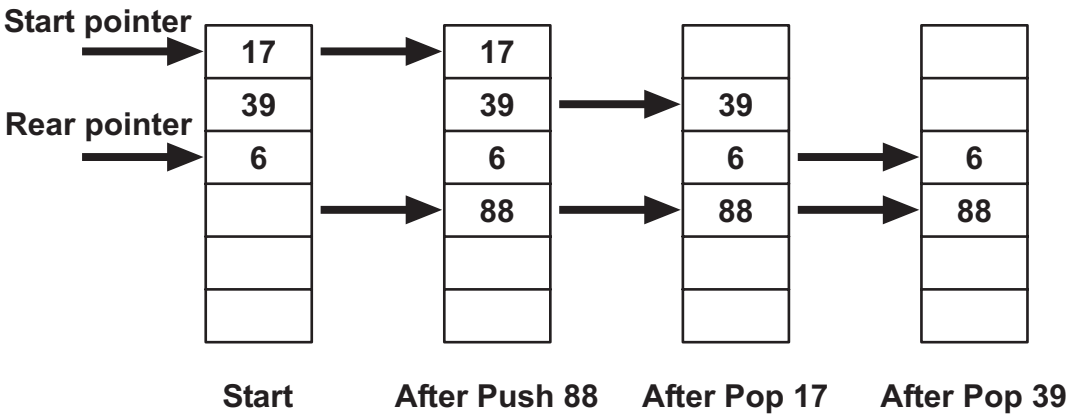


If a data item has to be removed from the queue then it is popped from the head of the queue. In this case if 12 is popped then the situation becomes:



An important aspect to realise here is that the data itself does not move but merely the pointers to the head and end of the queue.

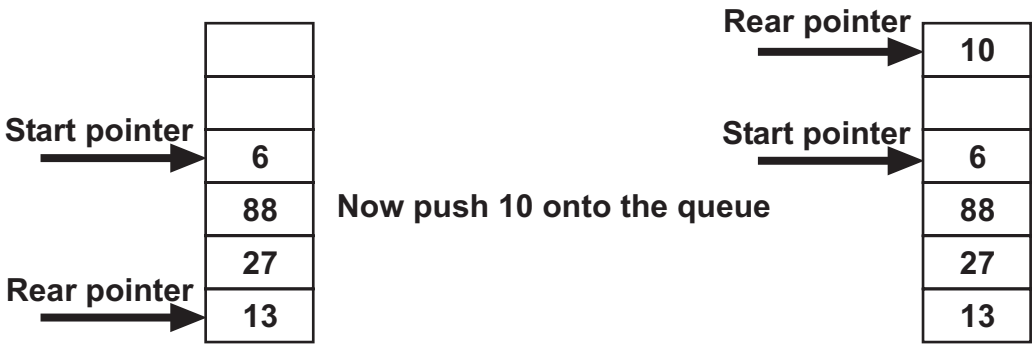
Consider the following sequence:



1. 88 is added to the end of the queue(pushed);
2. Rear pointer = Rear pointer + 1;
3. 17 is removed from the queue (popped);
4. Start pointer = Start pointer + 1;
5. 39 is removed from the queue (popped);
6. Start pointer = Start pointer + 1;

Suppose now that 27 and 13 are now added to the queue. At this point the rear pointer is referencing the end of the queue. We now want to add the value 10.

The situation would be:



There are no free locations at the end of the array, but at the start of the array there are free locations. The value 10 can be added there, with pointers updated appropriately. The array is now treated as a **circular queue**, with the first element following on from the last element in the array.A circular queue arrangement is such that when there is no space at the bottom of the queue you simply start again at the top. The rear pointer now indicates the item 10 which is now at the end of the queue, while the start pointer is unchanged.

With this organisation of start and rear pointers the representation of an empty queue and the detection of when the queue becomes full is tricky. Clearly, when the start

queue contains a single item, the start and rear pointers should coincide and refer to the single item in the queue. Thus the empty queue is represented by the rear pointer being one less than the start pointer, so that when the first item is added the pointers coincide. However, the same position could be reached by repeatedly adding items to the queue (for example, by adding another item after the 10 in the previous queue). So a full queue and an empty queue would be indistinguishable. A common way out of this difficulty is to have the position reached after the value 10 has been added to represent a full queue. This means that one location of the array will always be unused and the maximum number of items that can be stored in the queue is one less than the size of the array.

Queues are used when multiple printing jobs have to be processed and also during the scheduling of tasks in a multitasking environment. It is also an important structure in event-driven languages where events are placed in a queuing system to wait being processed.

6.4 Implementation of a queue

The following algorithms illustrate the addition of an item to the end of a queue and the removal of an item from the head of a queue. It is assumed that the queue is stored in an array with identifier 'queue' and that the array subscripts run from 1 to Maximum (inclusive).

Adding an item to the queue

```
If Rear = Maximum Then
  Rear = 1
Else
  Rear = Rear+1
EndIf
If Rear = Start-1 Or (Rear = Maximum and Start = 1) Then
  Output 'Queue Full'
Else
  queue(Rear) = data
EndIf
```

Removing an item from a queue

```
If Rear = Start-1 Or (Rear = Maximum and Start = 1) Then
  Output 'Queue Empty'
Else
  Data = Queue(Start)
EndIf
If Start = Maximum Then
  Start = 1
Else
  Start = Start + 1
```

Notice that if the full or empty queue conditions are satisfied, the start and rear pointers have been corrupted. More involved programming is required to maintain their consistency if the queue is required to ignore attempts to add items when it is full or remove items when it is empty.

An alternative implementation would maintain the rear pointer referring to the free space *after* the queue, so that the empty queue would be represented by the start and rear pointers being equal, and the full queue would be represented by the rear pointer being one less than the start pointer, taking into account any wrapping round. The insertion algoritm would change to increment the rear pointer *after* the time had been copied into the array.

6.5 Review questions

Q5: Describe fully the workings of a stack and explain why it is an important structure in computing?

Q6: Describe fully the workings of a queue and explain why it is an important structure in computing?

6.6 Records

Whereas an array can contain variables of the same type a record can contain different types and this makes them more complex to manipulate. Records are associated with databases and the following fields as shown in Table 6.1 are typical:

Table 6.1: Record fields

Field	Type
Surname	String
FirstName	String
Sex	Character
Address	String
PostCode	String
Telephone	String

A record can be therefore be regarded as a 2-dimensional array of data of different types and Table 6.2 illustrates this, showing three records:

Table 6.2: Records

Surname	First name	Sex	Address 1	Post code	'Phone
TOD	Andy	M	35 Brookside Drive	TY7 8UK	225 3625
BOYD	Mary	F	27 The Grange	OB7 RF12	335 2901
BELL	Charles	M	2 Bellside, Hunterstown	HT5 WA 3	213 1157

Each record is terminated by a CR, LF (indicated by *) and EOF control codes. In computer memory the records will be stored similar to the following:

Record1	*	Record2	*	Record3	*	Record4	*	Record5	*	EOF

Records can either be fixed or of variable length. There are advantages and disadvantages of each but for this particular topic we will only deal with fixed length records.

When allocating storage space, therefore the filing system must know the number of bytes in advance. For example, taking the fields in Table 6.1 we can roughly estimate the number of bytes for each field as shown in Table 6.3:

Table 6.3: Field lengths

Field	Bytes
Surname	25
FirstName	15
Sex	1
Address	50
PostCode	10
Telephone	14

This produces a record length of 115 bytes. Even if the information in a record did not total 115 bytes, it would be padded out with characters to that value. A file containing 100 such records would equate to around 10k to 12k allowing for space between blocks of records on backing store.

Exercise

Using your own personal data estimate the record size you will require based on Table 6.3.

6.7 Implementation of a record

A record is the smallest piece of information that can be manipulated using a random access file. Like a sequential file random access files store information as characters, one byte per character. Numbers however are stored in their native format.

Although random access files are less efficient in using disc space they do provide faster access to the information than sequential files since each record is of equal length and

can be located by its index number.

Another important aspect is that random access files can be read from and written to simultaneously.

In Visual Basic a record is defined using a Type statement that takes the form:

```
Type varType
  Variable1 As varType1
  Variable2 As varType2
   .....................
End Type
```

where `varType` is the record name and variables represent the fields.

This means that custom data types can be created by the user and manipulated in programs just like normal variables.

As an example, for the record shown in Table 6.2, the following Type statement would be used where the record type is called `Names`:

```
Type Names
   SurName As String * 25
   FirstName As String * 15
   Sex as Char
   Address As String * 50
   Post code As String * 10
   Phone As String * 15
End Type
```

In the `Type` statement, `Names` is the record name, each field is multiplied by a value to indicate its total length. Adding the field values gives a maximum record length of 116 characters or 116 bytes. Alternatively Visual Basic will do the calculation for you using the function `Len( )` which returns the length of the entire record.

The following statement opens a random access file both for reading from and writing to:

```
Open "Filename" For Random As #ChannelNumber Len = Len(variable)
```

where `ChannelNumber` is a value assigned by Visual Basic to the file during read/write operations, and the ' Len =' is part of the Open statement, indicating to Visual Basic the length of a record in the file, and the 'Len(variable)' is a function returning the length of a record referred to by 'variable'.

For example to open a data file named Books.dat that is in a folder called `My Documents` on the hard drive, the statement would be:

```
Open "c:\my documents\Books.dat" For Random As #1 Len = Len(Books)
```

Example : Worked example

This example will take you, step by step through the formalities of writing to and reading from a random access file in Visual Basic.

The following algorithms summarise the steps involved:

Create/write random file; use of the PUT statement

```
1. create recordName and fields
2. define recordName as Type
3. declare variable as type recordName to store data
4. create file using OPEN statement
5. assign values to fields
6. write data to file using PUT statement
7. display file
8. close file
```

Read from an existing random file; use of the GET statement

```
1. Open file
2. read records using GET statement
3. display records
4. close file
```

The algorithms may be implemented in Visual Basic.

For this example we will use a file called `Books.dat` and create some records.

Create/write random file

1. The record name is `Books`
2. The record `Books` is declared with its variables :

```
Type Books                    'create record type
  Title As String * 40        'with fields and sizes
  Author As String * 30
  ISBN As String * 15
  Price As Currency
End Type
```

3. The type variable is then declared.

   ```
   Dim BookInfo As Books
   ```

4. The file is opened (created if it does not exist):

   ```
   ChanNum = FreeFile()
   ```

   ```
   Open "Books.dat" for Random As ChanNum Len = Len(BookInfo)
   ```

   The channel number is assigned automatically by Visual Basic using the inbuilt function `FreeFile()`.

5. Values are ssigned to the variables as follows:

```
BookInfo.Title = "Grumpy Old Men"
BookInfo.Author = "Stuart Prebble"
BookInfo.ISBN = "0-563-52209-7"
BookInfo.Price = 8.95
```

6. The record is written to the file

   ```
   Put #ChanNum, 1, BookInfo
   ```

   The value 1 after `ChanNum` represents the optional record number. If it is omitted then Visual Basic will store the data at the current record's position. If a record exists at that specific location, it will be overwritten. The position of a record will be indicated by a variable called `Pointer` in the complete programs that follow.

7. Display records. This can be done using a `Picture box` called `PicOutput` and is only done to show that the data has been read in correctly and that the program is working;

   ```
   PicOutput.Print BookInfo.Title;Tab();BookInfo.Author;Tab();
      BookInfo.ISBN;Tab();BookInfo.Price
   ```

8. The file is closed. This will save the data to the file `Books.dat`:

   ```
   Close #ChanNum
   ```

Read from an existing file

1. If the file `Books.dat` has been previously saved it may be opened as before:

   ```
   ChanNum = FreeFile()
   Open "Books.dat" for Random As ChanNum Len = Len(BookInfo)
   ```

2. To read and display the records, the following code is used where the variable `Pointer` accesses each record one at a time and `TotRecords` is set to the number of records in the file:

   ```
   For Pointer = 1 to TotRecords
     Get #ChanNum, Pointer, BookInfo
     Display records
   Next Pointer
   ```

3. The files is closed as before:

   ```
   Close #ChanNum
   ```

The complete program listing is shown in Code 6.1:

```
Option Explicit
'MODULE CODE
Private Type Books
  Title As String * 20
  Author As String * 30
  ISBN As String * 15
```

```
  Price As Currency
End Type
'****************************************
'Program to manipulate records           *
'                                        *
'November 2004                           *
'****************************************
Private Sub CmdREAD_Click()
 ChanNum = FreeFile()
 Open "c:\Books.dat" For Random As ChanNum Len = Len(BookInfo)
 TotRecords = LOF(ChanNum) / Len(BookInfo)  'Find number of records
 RecordHeading
 For Pointer = 1 To TotRecords
  Get #ChanNum, Pointer, BookInfo
  OutputRecord
 Next Pointer
End Sub

Private Sub CmdWRITE_Click()
 Dim Count As Integer
 ChanNum = FreeFile()
 Pointer = 1
 Open "c:\Books.dat" For Random As ChanNum Len = Len(BookInfo)
  BookInfo.Title = "Grumpy Old Men"
  BookInfo.Author = "Stuart Prebble"
  BookInfo.ISBN = "0-563-52209-7"
  BookInfo.Price = 8.95
 Put #ChanNum, Pointer, BookInfo
 RecordHeading
 OutputRecord
 Pointer = Pointer + 1
  BookInfo.Title = "How to be a gardener"
  BookInfo.Author = "Alan Titchmarsh"
  BookInfo.ISBN = "0-563-53740-X"
  BookInfo.Price = 18.99
 Put #ChanNum, Pointer, BookInfo
 OutputRecord
 Pointer = Pointer + 1
  BookInfo.Title = "Himalaya"
  BookInfo.Author = "Michael Palin"
  BookInfo.ISBN = "0-297-84371-0"
  BookInfo.Price = 20.5
 Put #ChanNum, Pointer, BookInfo
 OutputRecord
 Close #ChanNum
End Sub

Sub OutputRecord()
 PicOutput.Print Trim$(BookInfo.Title); Tab(25);
```

```
 Trim(BookInfo.Author); Tab(50); Trim$(BookInfo.ISBN); Tab(75);
 BookInfo.Price;
 PicOutput.Print " "
End Sub

Sub RecordHeading()
 PicOutput.Print "TITLE"; Tab(28); "AUTHOR"; Tab(55); "ISBN";
 Tab(74); "PRICE"
 PicOutput.Print " "
End Sub

Private Sub CmdEXIT_Click()
 Close #ChanNum
 End
End Sub
```

Code 6.1: Record manipulation

Program notes

This is the simplest implementation of a random access file to process records. In the program the three records are written to the file `C:\Books.dat` by executing the `cmdWrite()` subroutine. Normally no output would appear since the file is being created or data written to the file. To make sure the subroutine is working the `OutputRecord` subroutine is called and this prints the records to the `Picture box`, as the variable `Pointer` increases through 1 to 3. The subroutine `RecordHeading` outputs a title each time the records are displayed.

In the records output statement the `Trim$` function is used to clip unwanted spaces from the records as they are printed to the `Picture box`.

To read the contents of the file, the number of records must be known. This is done by the statement:

```
TotRecords = LOF(ChanNum) / Len(BookInfo)
```

where LOF is the length of the file.

A screenshot is shown in Figure 6.4:

Figure 6.4: Output of records

Pressing the "Read file" and "Write file" buttons give identical output, as shown.

Exercise

Code the above program and run it a few times to acquaint yourself with the manipulation of records. When you are happy with it:

1. Remove the output statement from the cmdWrite function
2. Modify existing data by adding records of your own and testing the program.

Example : Extension exercise

Although the above program illustrates the basics of record manipulation using a random access file, it not at all flexible. The program has to be modified to input further data or change the data already present. Also there should be the option of choosing which file to process as being an input parameter to the program.

The following exercise implements these enhancements and introduces some extra features of Visual Basic.

A program has to be created that inputs information relating to the members of a local golf club. The data is to be stored as a file containing records of each member, the data being input by the user during program execution.

Use a random access filing system to store the data. To input each record, a form will be used with a text box to represent each field. Because the output might be quite large a picture box might be insufficient to display the data. Also we wish to separate output from input so that the form is not cluttered. A second form will therefore be used for output.

1. The following record structure will be used:

```
Type Membership
```

```
    SurName As String * 30
    FirstName As String * 15
    MemStatus As String * 15
    Handicap As Integer
End type
```

The record name of type `Membership` is `MemberInfo`

2. The project form is created as shown in Figure 6.5. This contains four labelled text boxes representing the field names and three command buttons. Sample input is also shown.

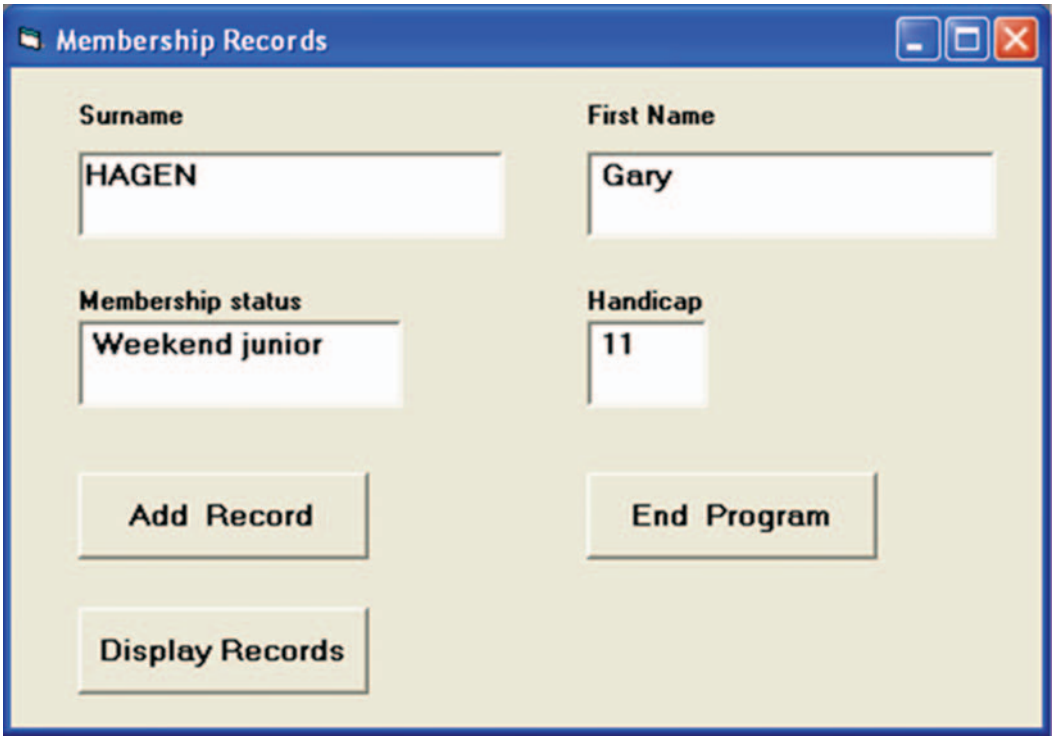


Figure 6.5: Record input form showing data

3. Each text box must now be assigned to each field. The format for this statement is:

   ```
   RecordName.Field = TxtField.Text
   ```

   For our Membership record the five statements will be:

   ```
   MemberInfo.SurName      =  TxtSurName.Text
   MemberInfo.FirstName    =  TxtFirstName.Text
   MemberInfo.MemStatus    =  TxtMemStatus.Text
   MemberInfo.Handicap     =  TxtHandicap.Text
   ```

4. The file is now opened using the statement:

   ```
   Open "c:\Members.dat" For Random As #ChanNum Len=Len(MemberInfo)
   ```

5. Data is entered into the text boxes and written to the file using the `PUT` statement.

6. To display the contents of the file containing the records a second form is added to the project. This is achieved by choosing `Project` from the menu followed by `Add Form`. This form is called `Output`. The data is displayed using the statements:

```
Output.Print MemberInfo.SurName
Output.Print MemberInfo.FirstName
Output.Print MemberInfo.MemStatus
Output.Print Membership.Handicap
```

7. The file is closed using `Close #ChanNum`

The complete code is shown in Code 6.2.

```
Option Explicit
Dim MemberInfo As Membership
Dim NoOfMembers As Integer
Dim FileName As String
Dim RecordLen As Integer
Dim TotalRecords As Integer
Dim ChanNum As Integer

'**********************************
'Record manipulation program 2     *
'                                  *
'December 2004                     *
'**********************************

Private Sub Form_Load()
 '**********************************
 ' OPEN FILE FOR WRITING           *
 '**********************************
 RecordLen = Len(MemberInfo)
 ChanNum = FreeFile
 FileName = InputBox$("Enter file name")
 Open FileName For Random As ChanNum Len = RecordLen
 NoOfMembers = LOF(Filename)\RecordLen
End Sub

Private Sub CmdAddRecord_Click()
 '************************************
 ' LOAD DATA FROM TEXT BOXES TO FILE  *
 '************************************
 GetRecord
 Put #ChanNum, NoOfMembers, MemberInfo
 NoOfMembers = NoOfMembers + 1
 TxtSurName.Text = " "
 TxtFirstName.Text = " "
 TxtMemStatus.Text = " "
 TxtHandicap.Text = " "
 TxtSurName.SetFocus
```

```
End Sub

Private Sub CmdDisplayFile_Click()
 '**************************
 ' DISPLAY CONTENTS OF FILE *
 '**************************
 Dim Count As Integer
 Heading
 For Count = 1 To Pointer
  Get #ChanNum, Count, MemberInfo       'Read record from file
  OutputRecords
 Next Count
 OutputForm.Print
 OutputForm.Print
 OutputForm.Print "Close window to continue"
 End Sub

Sub GetRecord()
 MemberInfo.SurName = TxtSurName.Text
 MemberInfo.FirstName = TxtFirstName.Text
 MemberInfo.MemStatus = TxtMemStatus.Text
 MemberInfo.Handicap = Val(TxtHandicap.Text)
End Sub

Private Sub Heading()
 OutputForm.Show
 OutputForm.Print "Surname"; Tab(18); "First name"; Tab(36);
  "Membership Status"; Tab(60); "Handicap"
 OutputForm.Print
End Sub

Sub OutputRecords()
 OutputForm.Print Trim$(MemberInfo.SurName); Tab(18);
 Trim$(MemberInfo.FirstName); Tab(38);Trim$(MemberInfo.MemStatus);
 OutputForm.Print Tab(64); Trim(MemberInfo.Handicap);
 OutputForm.Print
End Sub

Private Sub CmdEnd_Click()
 Close #ChanNum
 End
End Sub
```

Code 6.2: Membership program

Program notes

When you run the program you will be prompted to input a file name. If the file does not exist it will be created. The cursor will reside in the `Surname` box ready for input. Input a few records, each time pressing the `Add Record` button. By pressing the `Display Records` button all the records that have been input will be diplayed on the

output form. By closing this form more records may be added until the file is closed and the program terminated by pressing the `End Program` button.

If the program is run again using the same filename no records will be displayed when the `Display Records` button is pressed. However the information is still contained in the file and if new records are added, the file will be updated with the new information.

The `Sub CmdAddRecord_Click()` code clears the text boxes, calls the `GetRecord` routine that assigns the information that is input to the boxes using the `Put` command. Once the `Add Record` button has been activated, focus returns to the `SurName` box.

The largest chunk of the code deals with the display of the records using the `Get` command. The sub procedure `Heading` outputs a heading for the file on the output form. To activate this form the statement `OutForm.Show` is used The `For` loop then runs through each record up to the value dictated by the variable `Pointer` and prints each on the output form. The output is performed by the sub routine `OutPutRecords` that formats the output statements using the `Tab` function. Finally to remove unwanted spaces from each record the `Trim$` function is used. Remember that the records are fixed length format so some will contain extra "padding" to make up the common length.

A program run is shown in Figure 6.6.

Figure 6.6: Program output showing records

Exercise

You may wish to add a fourth button to the form that will open and display the contents of an existing file. If you refer to the `Books.dat` program, this should give you help.

```
Private Sub CmdLoad_Click()
 ChanNum = FreeFile()
 RecordLen = Len(MemberInfo)
 FileName = InputBox$("Enter file name")
 Open FileName For Random As ChanNum Len = RecordLen
 TotalRecords = LOF(ChanNum) / Len(MemberInfo)
 OutputForm.Show
```

```
 For Pointer = 1 To TotalRecords
  Get #ChanNum, Pointer, MemberInfo
  OutputForm.Print Trim$(MemberInfo.SurName); Tab(18);
  Trim$(MemberInfo.FirstName); Tab(38);
  Trim$(MemberInfo.MemStatus);
  OutputForm.Print Tab(64); Trim(MemberInfo.Handicap)
 Next Pointer
End Sub
```

Code 6.3: Membership program

Exercise 2

Use the data that you produced in the last Exercise and create a record structure in your programming language.

6.8 Review questions

Q7: What are the differences between a sequential file and a random access file?

Q8: What is a record and why does the filing system require to know its length?

Q9: Estimate the storage required for 100,000 records if each record contains the information shown. Provide realistic field sizes yourself. Assume all fields contain information.

- Surname
- First name
- Date of birth
- Age
- Address
- City
- Post code
- Telephone
- Mobile
- e-mail

6.9 Summary

The topic has introduced the dynamic data types, the stack and the queue both of which are used extensively in computing. File handling techniques were extended to the discussion and exemplification of records in both pseudocode and a high level language.

By the end of this topic you should be aware of the following objectives:

- describe the structure and exemplification of both a stack and queue and explain their importance in computing;

- describe the structure and exemplification of a record.

End of topic test

Q10: Which one of the following is an example of a static data structure?

a) a procedure
b) a stack
c) a record
d) a queue

Q11: PUSH and POP are procedures used when inserting and deleting items from:

a) an array
b) a sorted list
c) a sequential file
d) a stack

Q12: Stacks and queues have **two** error situations. They are _____ and _____.

Q13: Which one of the following responses is false:

a) Records and stacks are static data structures
b) Stacks and arrays are dynamic data structures
c) Queues and stacks are dynamic data structures
d) Queues and records are static data structures

Q14: Both stacks and queues are useful structures used during the execution of programs.

a) True
b) False

Q15: To store 5000 records containing information under the fields name, address and age would require approximately how many bytes?

a) 250b
b) 250Kb
c) 250Mb
d) Impossible to say

Q16: The following print jobs are held in a queue:

job 1	job 2	job 3	job 4

Front Rear

Which one of the following data structures would be best suited to reverse the elements of the queue?

a) Stack
b) Another queue
c) 1-dimensional array
d) 2-dimensional array

Topic 7

Standard algorithms

Contents

Learning Objectives

After studying this topic, you should be able to:

- *Describe and exemplify the following standard algorithms in pseudocode and an appropriate high level language*
  - *binary search*
- *describe and compare simple linear and binary search algorithms;*
- *describe and compare sort algorithms for simple sort, bubble sort and selection sort in terms of number of comparisons and use of memory;*
- *describe and exemplify of user-defined module libraries.*

In this final topic standard algorithms are revisited in the form of linear and binary searches that are compared. Sorting techniques are compared under various criteria and implemented in both pseudocode and a high level language. Finally you will have the prospect of creating your own module libraries that can be saved and used at any time in your programming work.

Searching and sorting are two important procedures used in data processing.

Searching is simply finding a 'key' item by scanning a data list until the search key is found or it is not in the list. Scanning a telephone directory or index of a book is a search routine.

Sorting techniques arrange data into some form of order such as numeric, alphabetic ascending or descending. A deck of playing cards could be sorted in order of suits, spades, hearts, diamonds, and clubs with the cards in numerical order within the suits, Ace to King.

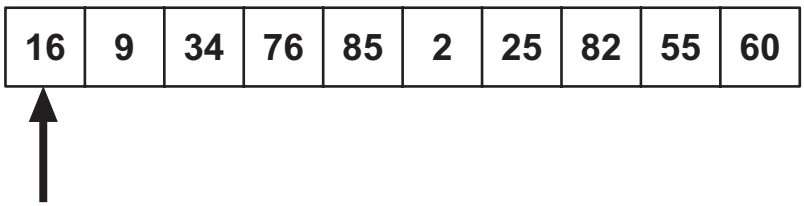
7.1 Searching techniques

There are many search techniques available and the two that we will look at are the linear or sequential search and the binary search. The linear search was introduced to you in the Higher Computing course and we want to examine the process in a bit more detail.

7.2 Linear Search

This is a reminder of what a linear search involves.

The linear search is the simplest search method to implement and understand. Starting with an array of length 10, holding say, 10 integers with a pointer indicating the first item the user inputs a search key. Scanning then takes place from left to right until the search key is found, if it exists in the list. Look at the list below:

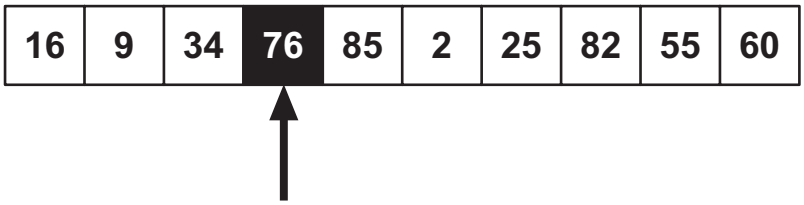


Suppose the search key is **76**.

Starting at array location 0 containing the value 16.

1. The value 16 is compared to the key, 76. Not equal to key so pointer moves on one place;
2. The value 9 is compared to the key. Not equal so pointer moves on;

3. The value 34 is compared to key. Not equal so pointer moves on;
4. The value 76 compared with key. Success! Key found at position 3 in the list.



A linear search may be represented by the following general algorithm:

```
1  Set found = false
2  Input search key
3  Point to first element in list
4  Do While (not end of list) AND (not found)
5   If array[value] = key then
6    found = true
7    Output suitable message
8   Else
9    look at next element in list
10  End if
11 Loop
12 If (not found) then
13  key not in list
14 End if
```

The linear search is not a bad algorithm for short lists. It is easier to implement than some of the other methods and the list does not need to be sorted. Indeed it might be the only method suitable for large, unordered tables of data and files.

However the linear search is not a very efficient strategy since each array element has to be compared with the search key until a match is found.

Analysis of the linear search

For comparison with other types of search routines the linear search algorithm can be analysed in terms of the average number of comparisons to find the search key. This involves some mathematics. You are not required to learn any proofs for this course.

Assume that the search operates on a list of N items.

1 comparison is required to access the 1st item

2 comparisons are required to access the 2nd item

.....

N comparisons are required to access the last item.

The average search length for N items is given by the expression:

```
(1 + 2 + 3 +.............................+N)/N
```

Using mathematics it can be shown that:

```
(1 + 2 + 3 +.............................+N)
```

```
= N(N + 1)/2
```

This means that for a linear search, the average search length approximates to:

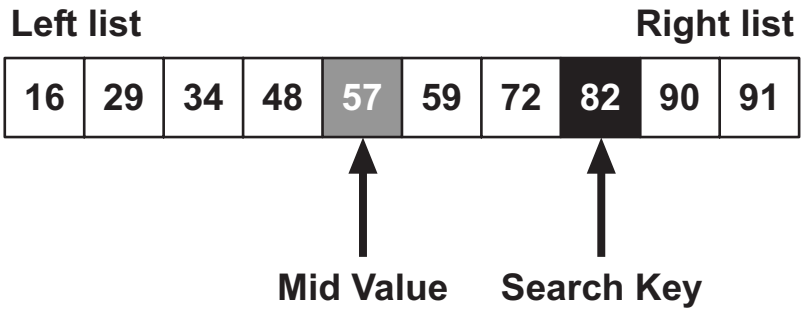
$N(N + 1)/2N = (N + 1)/2 \rightarrow N/2$

So, for a list of 64 items, the average search length will be approximately 32.

7.3 Binary search

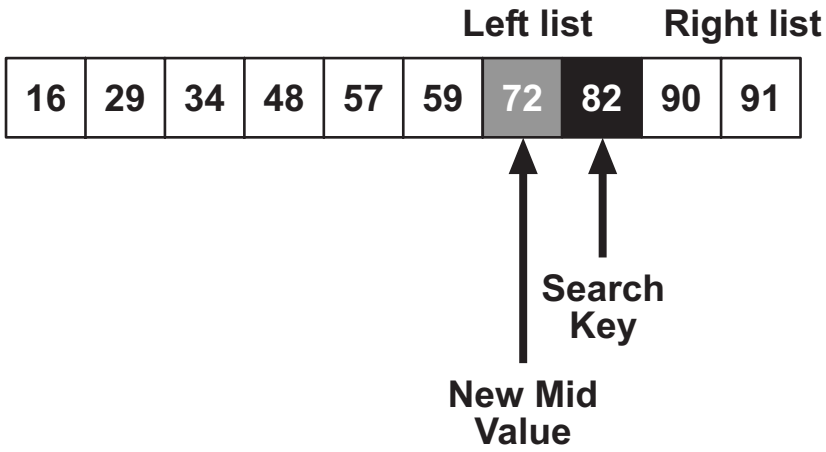
This is a much faster method than the linear search but the data has to be in order. This search is familiar to everyone who uses a telephone directory! Sometimes called a **binary chop** it splits the data list into two sub lists and repeats this process of splitting until the search key is found, if it exists in the list. This is referred to as **divide and conquer**, which features in many sorting algorithms.

Consider the following array of length 10, containing 10 integers in numerical order with a search key of 82.

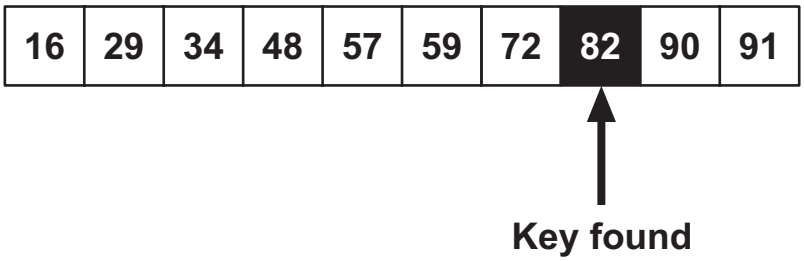


Choosing a value around the mid point of the list, (57) will produce two sub lists - a left list and a right list. Now proceed as follows:

1. Compare mid value with key. $82 > 57$ so ignore left list;
2. Create new mid location between location 4 and location 9. This location 6 (72).



3. Now compare new mid value with key. 82 $>$ 72) so ignore left list;
4. Create new mid location between location 6 and location 9. This is location 7 (82).
5. Compare mid value with key. 82 = 82 so key is found at position 7.



Analysis of the binary search

Recall that in a linear search the number of comparisons made before the search key is found is given by *N/2* where N is the number of data items or length of the list. In the same way we can apply mathematical rules to find the average number of comparisons made during a binary search.

It can be shown that there is a logarithmic relationship between the number of data elements and the number of comparisons required. NB You do not require to understand logarithmic expressions; merely appreciate why they are used for the sake of comparison.

Table 7.1 summarises the number of comparisons made for varying numbers of data elements in a binary search.

Table 7.1: Binary search comparisons

Number of data elements	Number of required comparisons	Relationship
$2 = 2^1$	1	$Log_2 2 = 1$
$4 = 2^2$	2	$Log_2 4 = 2$
$8 = 2^3$	3	$Log_2 8 = 3$
$16 = 2^4$	4	$Log_2 16 = 4$
$32 = 2^5$	5	$Log_2 32 = 5$
$64 = 2^6$	6	$Log_2 64 = 6$

For a list of N data items the average search length would be $Log_2 N$.

So, for a list of 64 items, the average search length will be 6.

7.4 Implementation of a binary search

A binary search can be represented by the following general algorithm:

```
1  Set found = false
2  Input Search key
3  Repeat
4    Set pointer to middle of list {Length of list DIV 2}
5    If array[middle] = key then
6      found = true
7      Output suitable message
8    else
9    If array[middle] < key then
10     last_location = middle      {search left sub list}
11   else
12     First_location = middle     {search right sub list}
13 Until found =true
```

A binary search can be implemented *recursively*. When it determines whether the search key lies in the left or right half of the array, it could make a call to search the appropriate half using the same procedure call.

The binary search algorithm can be implemented in a high level language. The code shown in Code 7.1 is the full implementation in Visual Basic.

```
Option Explicit
Dim Passes As Integer, Swaps As Integer, Fill As Integer,
    Temp AS Integer, Pointer As Integer
Dim List(15) As Variant

Private Sub Command1_Click()
'*********************************************
```

```
'BINARY SEARCH PROGRAM                              *
'                                                   *
'December 2004                                      *
'                                                   *
'This program will perform a BINARY SEARCH on       *
'an array holding 16 integers in ascending order*
'************************************************

Setup                                               'Main program procedures
Populate_List List()
Binary_Search List()
End Sub

Private Sub Setup()                                 'Initialise variables
  PicList.Cls                                       'and clear output boxes
  PicResult.Cls
  PicPass.Cls
  PicSearch.Cls
  PicList.Print
  PicResult.Print
End Sub

Private Sub Populate_List(ByRef List())             'Fill array with 16 integers
  Dim Fill As Integer
  PicList.Print " ";
  List(0) = 8
  List(1) = 11
  List(2) = 17
  List(3) = 22
  List(4) = 29
  List(5) = 38
  List(6) = 51
  List(7) = 58
  List(8) = 65
  List(9) = 67
  List(10) = 74
  List(11) = 81
  List(12) = 87
  List(13) = 90
  List(14) = 96
  List(15) = 97
  For Fill = 0 To 15                                'Output integers
    PicList.Print List(Fill);
  Next Fill
End Sub

Private Sub Binary_Search(ByRef List())
 Dim Count As Integer, Middle As Integer, FirstLocation As Integer
 Dim LastLocation As Integer, SearchKey As Integer, Passes As Integer,
```

```
    Found As Boolean
    Found = False
    Passes = 0
    FirstLocation = 0
    LastLocation = 15
    SearchKey = InputBox("Please input the search key")
    PicSearch.Print SearchKey
    Do
      Passes = Passes + 1
      Pointer = (FirstLocation + LastLocation) / 2    'Find mid position
      If List(Pointer) < SearchKey Then
       FirstLocation = Pointer + 1
      Else
       LastLocation = Pointer - 1
      End If
      If List(Pointer) = SearchKey Then
       Found = True
       PicResult.Print "Search Key "; SearchKey; " found at position ";
                       Pointer
       PicPass.Print " "; Passes
     End If
    Loop Until Found = True Or Passes > 4
      If Passes > 4 Then
       PicResult.Print "Search Key "; SearchKey; " not found in list!"
     End If
  End Sub

  Private Sub CmdExit_Click()
    End
  End Sub
```

Code 7.1: Binary search

Program notes

The main procedure is `Binary_search List()`. The user is prompted to input a search key. A `Do Until` loop controls the program. The list is divided into two by the variable `Pointer` being set to the mid position. The search key is compared to the value at `List(Pointer)`. If search key is greater then bottom half of list discarded else the top half is discarded. The variables `FirstLocation` and `LastLocation` are reassigned to the respective half and the search algorithm is repeated. This continues until the search is found, or not.

Figure 7.1 shows output from a program run:

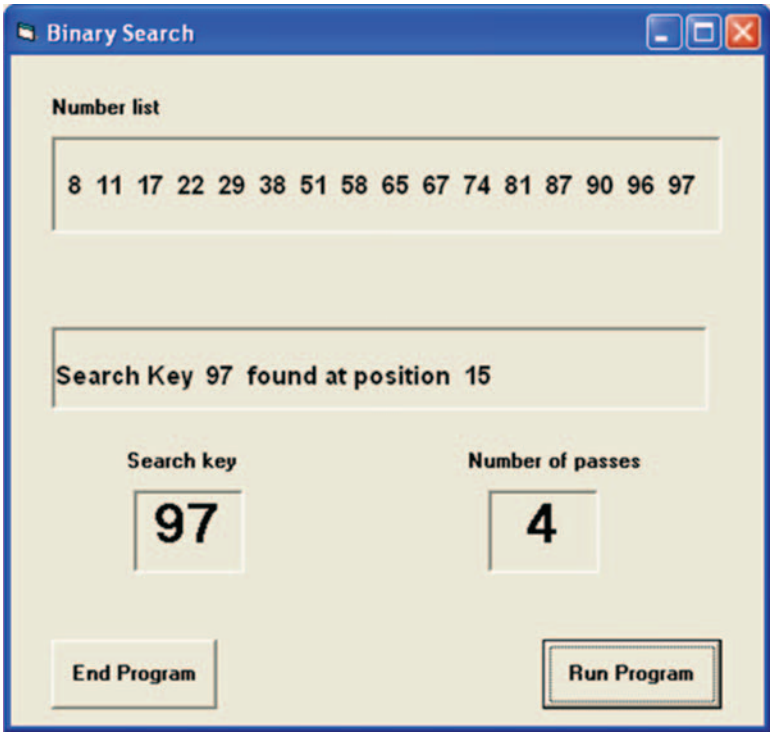


Figure 7.1: Output from binary search

Exercise

Download the code and run the program, choosing values:

1. Near the start of the list
2. At the end of the list
3. In the middle of the list
4. That are not in the list

A summary of the two searching techniques is given in Table 7.2:

Table 7.2: Summary of searches

Linear search	Binary search
Is simple to code and implement	Is more complex to code
Quite efficient for short length data lists	Efficient for any length of data lists
Very slow on large data lists since each data element has to be compared	Fast on any length of data list since it only deals with halve sub-lists. Hence the name binary chop.
Does not require data to be ordered	Data has to be ordered.
Average search length is N/2 where N is the number of data elements.	Search length is $\log_2 N$
Plays a part in other algorithms such as finding maximum, minimum and also in the selection sort	Binary chop is used in fast sorting routines

7.5 Review questions

Q1: Explain how a binary search works.

Q2: Compare a linear search with a binary search, outlining **three** main differences.

7.6 Sorting

Sorting is an important process in computing, especially in data processing. Company records will usually exist in some sort of order, numeric or alphabetic, ascending or descending, based on a key field in the database. It is difficult to be unaware of sorting all around you in daily life; bank statements are sorted in numerical order based on date so the customer can verify the transactions. Telephone directories, book indices, sports league tables, catalogues of various descriptions, 'Top of the pops', weather records, lottery numbers and so on all specify sorting in one form or another.

The examples quoted above are processed as **external sorts**. This means that the processing of the files will use external storage devices such as magnetic disc or tape units since the files will require large amounts of storage for the sorting techniques. In archiving files, the transaction file is merged with the daily master file to produce a new master file that is in some sort of order. This process usually starts at the end of a day and continues overnight in preparation for the following day.

In this topic we will concern ourselves with **internal sorts**. Here the sorting algorithms are able to use computer memory (RAM) to order the data contained in fairly small lists.

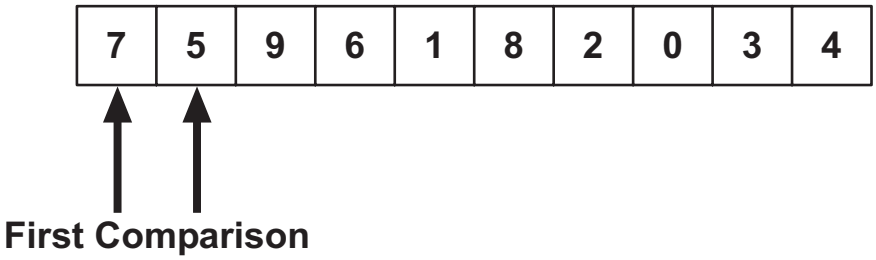
Three sorting algorithms will be described and compared:

1. Simple sort
2. Bubble sort
3. Selection sort using two lists

In the following descriptions all the algorithms will sort the data, stored in a 1-dimensional array, into ascending numerical order.

7.6.1 Simple sort

Sorting cannot get any simpler than this one. Consider the initial list of integers as shown:



The simple sort works as follows:

1. Starting with the first two items (7, 5) compare them. If the first is greater than the second, swap them.
2. Compare first item with third (now 5, 9) and if the third is greater than the first, swap them.
3. Repeat comparing first item with successive values and swap, if required, until end of list is reached
4. Now repeat the process starting with second item and compare with successive values until end of list.
5. Continue comparing each item with remaining items until the list is sorted.

Table 7.3 shows the entire process for the first comparisons (pass1). The compared values are shown in bold and the swaps indicated, where they occur:

Table 7.3: First pass of simple sort

	7	5	9	6	1	8	2	0	3	4	Initial list
Pass 1	**7**	**5**	9	6	1	8	2	0	3	4	Compare 7 and 5 - swap
Pass 1	**5**	7	**9**	6	1	8	2	0	3	4	Compare 5 and 9 - no swap
Pass 1	**5**	7	9	**6**	1	8	2	0	3	4	Compare 5 and 6 - no swap
Pass 1	**5**	7	9	6	**1**	8	2	0	3	4	Compare 5 and 1 - swap
Pass 1	**1**	7	9	6	5	**8**	2	0	3	4	Compare 1 and 8 - no swap
Pass 1	**1**	7	9	6	5	8	**2**	0	3	4	Compare 1 and 2 - no swap
Pass 1	**1**	7	9	6	5	8	2	**0**	3	4	Compare 1 and 0 - swap
Pass 1	**0**	7	9	6	5	8	2	1	**3**	4	Compare 0 and 3 - no swap
Pass 1	**0**	7	9	6	5	8	2	1	3	**4**	Compare 0 and 4 - no swap
	0	7	9	6	5	8	2	1	3	4	End of pass 1
Pass 2	0	**7**	**9**	6	5	8	2	1	3	4	Start of pass 2

By the end of pass 1 the smallest value has migrated to the start of the list, its correct position.

Subsequent passes will establish the correct positions for the remaining values.

Exercise

A good way to follow the process of this and subsequent sorts is to use numbered counters. Manually run through the description and steps in Table 3.11and move the counters accordingly. Use this to determine the sequence of swaps in the second pass and subsequent passes.

7.6.2 Simple sort Implementation

The algorithm will require two looping structures; an outer loop that deals with the main item for comparison and an inner loop that runs through the remaining values that are compared with the main item. The items are stored in an array called list (N).

```
1. for outer = 0 to n
2.  for inner = outer + 1
3.   if List (outer) > List(inner) then
4.    swap values
5.   end if
6.  next inner
7. next outer
```

The algorithm may be implemented in a high level language. The code shown in Code 7.2 is a full implementation in Visual Basic.

```
Option Explicit
Dim Passes As Integer, Swaps As Integer, Fill As Integer, Temp As Integer
Dim Outer As Integer, Inner As Integer, Comparisons As Integer
Dim Flag As Boolean
```

```
Dim List(16) As Variant

Private Sub cmdRun_Click()

'************************************************
'Program Simple Sort                             *
'                                                *
'This program will perform a SIMPLE SORT on      *
'an array holding 16 random integers and output  *
' the ordered list in ascending order together   *
'with the number of passes and swaps.            *
'                                                *
'December 2004                                   *
'************************************************
Setup
Populate_List List()
Simple_Sort List()
Output_Results
End Sub

Private Sub Setup()
 Swaps = 0
 Passes = 0
 Comparisons = 0
 Randomize
 PicList.Cls
 PicList.Print
 PicOutput.Cls
 PicOutput.Print
 PicPass.Cls
 PicSwap.Cls
 PicComparisons.Cls
End Sub

Private Sub Populate_List(ByRef List())
 Dim Fill As Integer
 For Fill = 0 To 15
  List(Fill) = Int(99 * Rnd) + 1
  PicList.Print List(Fill);
 Next Fill
End Sub

Private Sub Simple_Sort(ByRef List())
 For Outer = 0 To 15
   For Inner = Outer + 1 To 15
     Comparisons = Comparisons + 1
     If List(Outer) > List(Inner) Then
       Temp = List(Outer)                    'SWAP values
       List(Outer) = List(Inner)
```

```
            List(Inner) = Temp
            Swaps = Swaps + 1
          End If
        Next Inner
      Passes = Passes + 1
    Next Outer
  End Sub
  Private Sub Output_Results()
    For Outer = 0 To 15
      PicOutput.Print List(Outer);
    Next Outer
    PicPass.Print Passes
    PicSwap.Print Swaps
    PicComparisons.Print Comparisons
  End Sub

  Private Sub cmdExit_Click()
    End
  End Sub
```

Code 7.2: Simple sort

Program notes

The program uses four procedures, described in Table 7.4:

Table 7.4: Simple sort procedures

Setup	Sets up variables and clears output boxes
Populate_ List List()	Creates 16 random numbers which are stored in the array List(15).
Simple_Sort List()	Two for loops perform the scanning routines and comparisons of the values. The swap routine is also present in the procedure.
Output_Results	Results are output to 5 boxes: original list, sorted list, number of passes, number of comparisons and number of swaps.

Sample output is shown in Figure 7.2:

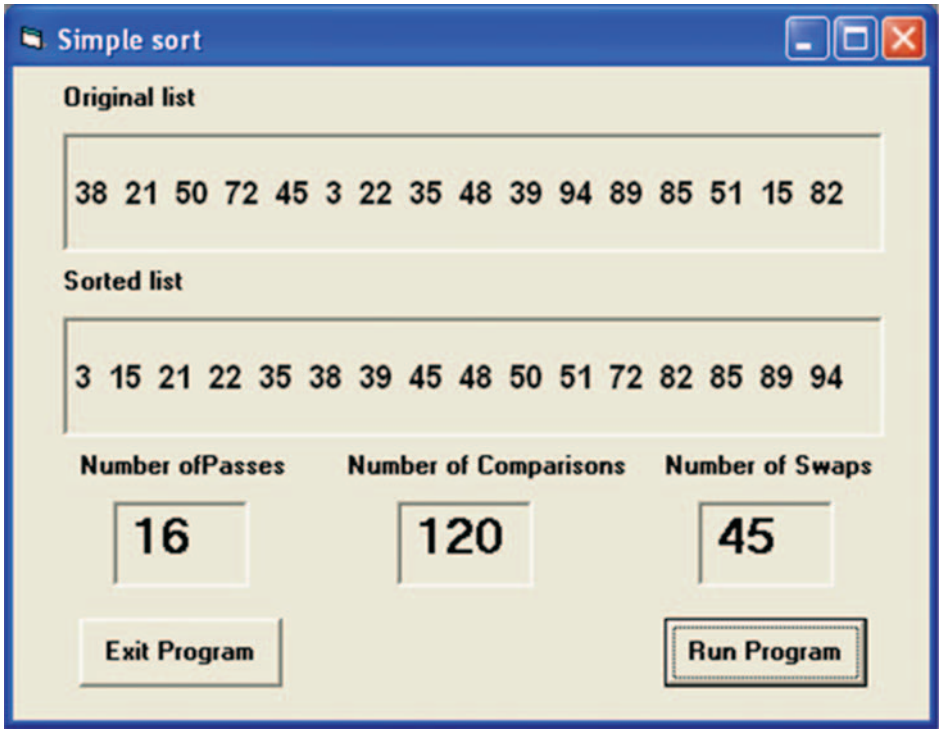


Figure 7.2: Simple sort output

Analysis of the simple sort

The simple sort is a fairly inefficient algorithm since it runs through every value when comparing values.

For an array containing 16 integers the number of passes will always be 16. The number of comparisons will always be 120 but the number of swaps will vary, depending on the original list.

These values are confirmed when the program is run.

The number of comparisons can be shown mathematically to be N(N - 1)/2

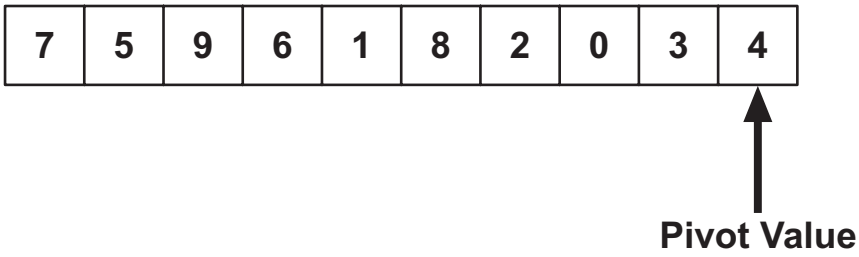
Confirm that this gives the value 120 for an array conatining 16 values.

Allowing for the fact that many comparisons are made the memory overheads for this type of sort are negligible.

7.6.3 Bubble sort

This is, perhaps the best known of all sorting algorithms and also one of the simplest to understand and implement. It is very much like the simple sort but, overall is a fairly inefficient sorting algorithm.

Consider the initial list of integers as shown:



The bubble sort works as follows:

1. Establish the pivot value which is at the top end of the list
2. Starting at the bottom of the list compare first two values (7,5).
3. If first is greater than second then swap.
4. Continue comparing adjacent values up to the pivot value and swap where necessary.
5. Now move pivot to 2nd top location
6. Repeat process of comparing from start of list and moving pivot down until it is pointing at the first location.

The list is now sorted.

The bubble sort is so named since the largest element 'bubbles' to the top during program execution.

Table 7.5 shows the entire process for the first pass. The compared values are shown in bold and the swaps are also indicated, where they occur:

Pivot value = 4.

Table 7.5: First pass of bubble sort

	7	5	9	6	1	8	2	0	3	4	Initial list
Pass 1	**7**	**5**	9	6	1	8	2	0	3	4	Compare 7 and 5 - swap
Pass 1	5	**7**	**9**	6	1	8	2	0	3	4	Compare 7 and 9 - no swap
Pass 1	5	7	**9**	**6**	1	8	2	0	3	4	Compare 9 and 6 - swap
Pass 1	5	7	6	**9**	**1**	8	0	2	3	4	Compare 9 and 1 - swap
Pass 1	5	7	6	1	**9**	**8**	0	2	3	4	Compare 9 and 8 - swap
Pass 1	5	7	6	1	8	**9**	**0**	2	3	4	Compare 9 and 0 - swap
Pass 1	5	7	6	1	8	0	**9**	**2**	3	4	Compare 9 and 2 - swap
Pass 1	5	7	6	1	8	0	2	**9**	**3**	4	Compare 9 and 3 - swap
Pass 1	5	7	6	1	8	0	2	3	**9**	**4**	Compare 9 and 4 - swap
	5	7	6	1	8	0	2	3	4	**9**	End of pass 1
Pass 2	**5**	**7**	1	6	8	0	2	3	4	9	Start of pass 2

By the end of pass 1 the largest value (9) has bubbled to the top of the list, its correct position.

Subsequent passes will establish the correct positions for the remaining values.

Exercise

Establish the sequence of subsequent passes

7.6.4 Bubble sort implementation

A bubble sort can be implemented by the following algorithm. The items are stored in an array called `list(N)`. Two loops are required; an inner one to run through the values to allow comparisons take place and an outer loop that begins at a pivot value and iterates downwards towards the first item.

```
1. for outer  = n - 1 to 0 step -1
2.  for inner = 0 to N - 1
3.   if list(inner) > list(inner + 1) then
4.    swap values
5.   end if
6.  next inner
7. next outer
```

The algorithm terminates when the list is sorted.

The algorithm may be implemented in a high level language. The code shown in Code 7.3 is a full implementation in Visual Basic.

```
Option Explicit
Dim Passes As Integer, Swaps As Integer, Fill As Integer, Temp As Integer
Dim Flag As Boolean, Comparisons As Integer
Dim List(15) As Variant

Private Sub Command1_Click()

'*****************************************
'Program Bubble sort                     *
'                                        *
This program will perform a BUBBLE SORT on   *
'an array holding 16 random integers and     *
'output the ordered list and the number of   *
'swaps, comparisons and passes.              *
'                                        *
'December 2004                           *
'*****************************************
Setup                                        'Main program procedures
Populate_List List()
Bubble_Sort List()
Output_Results
End Sub
```

```
Private Sub Setup()                              'Initialise variables
  Swaps = 0                                      'and clear output boxes
  Passes = 0
  Comparisons = 0
  Flag = True
  Randomize
  PicList.Cls
  PicResult.Cls
  PicPasses.Cls
  PicSwaps.Cls
  PicComparisons.Cls
  PicList.Print
  PicResult.Print
End Sub

Private Sub Populate_List(ByRef List())          'Fill array with
  Dim Fill As Integer                            '16 random numbers
  PicList.Print " ";
  For Fill = 0 To 15
    List(Fill) = Int(99 * Rnd) + 1
    PicList.Print List(Fill);
  Next Fill
End Sub

Private Sub Bubble_Sort(ByRef List())             'Start the sort procedure
Dim Outer As Integer
Dim Inner As Integer
For Outer = 15 To 0 Step -1
  For Inner = 0 To Outer - 1
    Comparisons = Comparisons + 1
    If List(Inner) > List(Inner + 1) Then
      Temp = List(Inner)                           'Swap array contents
      List(Inner) = List(Inner + 1)
      List(Inner + 1) = Temp
      Swaps = Swaps + 1
    End If
  Next Inner
  Passes = Passes + 1
Next Outer
End Sub

Private Sub Output_Results()
Dim Count As Integer
PicResult.Print " ";                               'Output results
For Count = 0 To 15
PicResult.Print List(Count);
Next Count
PicPasses.Print Passes
```

```
PicSwaps.Print Swaps
PicComparisons.Print Comparisons
End Sub

Private Sub Command2_Click()
  End
End Sub
```

Code 7.3: Bubble sort

Program notes

The program uses four procedures as shown in Table 7.6:

Table 7.6: Bubble sort procedures

Setup	Sets up variables and clears output boxes
Populate_ List List()	Creates 16 random numbers which are stored in the array List(15).
Bubble_Sort List()	The value at the top end of the list is taken as the pivot value. Two loops are required, an outer and an inner. The outer loop controls the number of passes through the list starting with the pivot and working downwards to location 0. The inner loop is responsible for the comparisons made and any swaps that have to be performed.
Output_Results	Results are output to 5 boxes: original list, sorted list, number of passes, number of comparisons and number of swaps.

Sample output is shown in Figure 7.3:

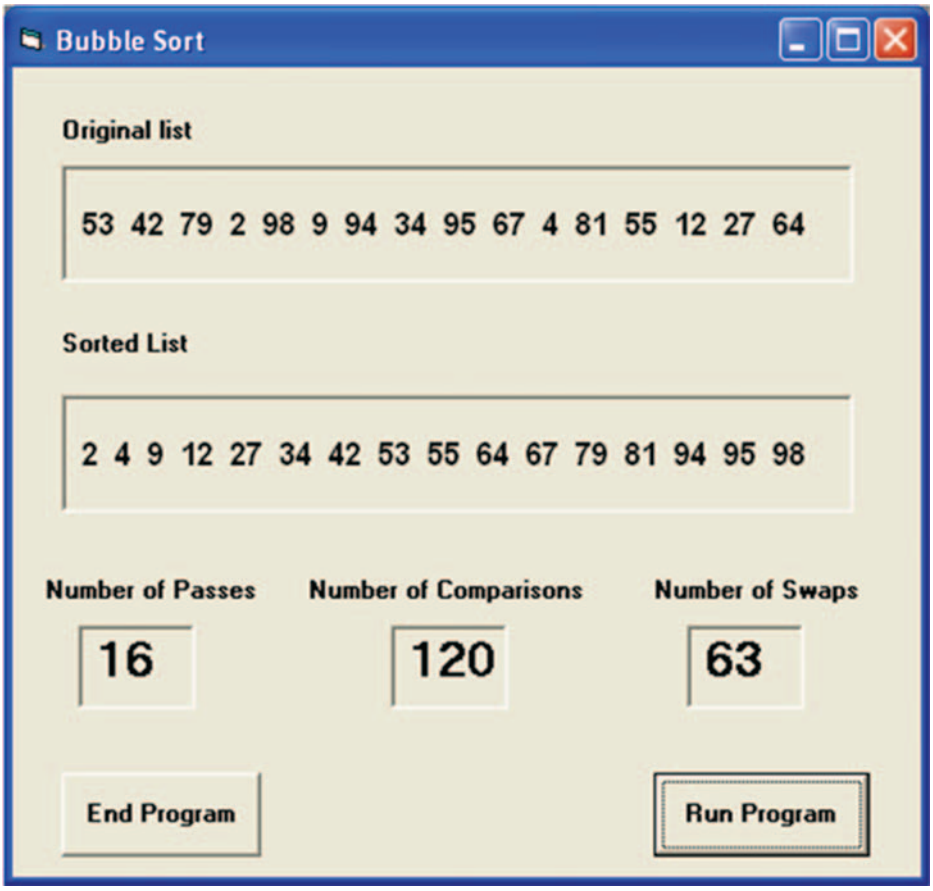


Figure 7.3: Bubble sort output

Analysis of the bubble sort

The bubble sort is also a fairly inefficient algorithm since it makes an excessive number of comparisons and is rarely used for serious sorting applications. It is only useful for small data list up to around 25 items. Its main use is in the teaching of how sorting algorithms work.

It can be shown that in the worst case (original data is in reverse order) the algorithm makes N x N comparisons and swaps. It is referred to as an N^2 sort. What this means is that if it takes 5 seconds to sort a list of 1000 items, then to sort a list of 2000 items it will take 25 seconds.

There are no memory overheads using this algorithm.

Alternative version

There is a more efficient version of the sort that is often cited in literature as the main bubble sort algorithm. In this version a boolean variable is set to keep track of the swaps. If no swaps take place then the algorithm terminates. This reduces the time for the number of passes through the list.

The algorithm is as follows:

```
1.  flag = true
2.  do while flag
3.    flag = false
4.    for count = 0 to N - 1
5.      if list(count) > list(count + 1) then
6.        swap values
7.        flag = true
8.      end if
9.    next count
10. loop
```

This algorithm produces identical sorting to the simpler version in that the smaller elements bubble to the start of the list.

Exercise

Modify the program in Code 7.3 to include the above algorithm and run it a few times.

7.6.5 Selection sort using two lists

Compared to the simple and bubble sort methods, the selection sort is probably the more intuitive in that It seems the natural way to sort data. An enhancement to the algorithm is to use two lists instead of one. Many external sorts and some internal ones use two or more lists that are then merged by various methods into a single, ordered list.

In this selection sort method we are going to use two lists, List A which contains the data and List B which is empty.

The sort works in the following way:

1. Scan List A using a linear search until the minimum value is found.
2. Place this value in the first location of List B
3. Replace this item with a dummy value
4. Repeat scan for minimum value and place in second location in List B
5. Repeat process until List B contains all values in ascending order and List A contains dummy values.

Figure 7.4 shows the initial stages of the process:

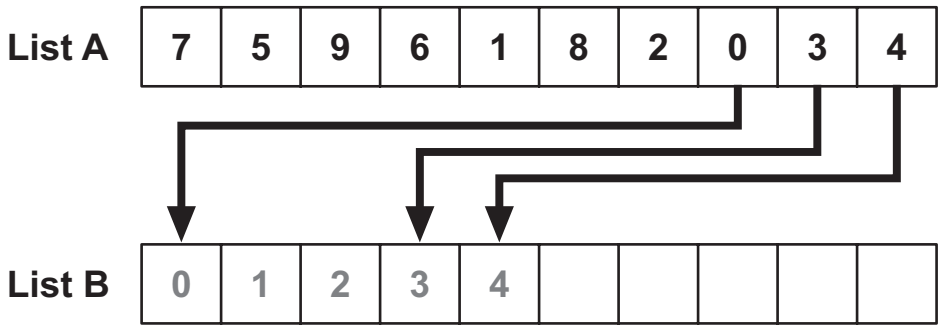


Figure 7.4: Start of selection sort using two lists

7.6.6 Implementation of selection sort using two lists

Starting with List A containing N items and List B empty the algorithm is a follows:

```
1.  for outer = 0 to n - 1
2.    minimum = outer
3.    for inner = 0 to N           {line modified for two lists}
4.      if list_A(inner) < list_A(minimum) then
5.        minimum = inner
6.      end if
7.    next inner
8.    list_B(outer) = list_A(minimum)
9.    list_A(minimum)  =  dummy value
10. next outer
```

The algorithm may be implemented in a high level language. The code shown in Code 7.4 is a full implementation in Visual Basic.

```
Option Explicit
Dim Passes As Integer, Comparisons As Integer, Fill As Integer,
    Temp As Integer
Dim Flag As Boolean
Dim ListA(15) As Variant
Dim ListB(15) As Variant
Private Sub CmdRun_Click()

'*********************************************
'Program Selection Sort                      *
'                                            *
'This program will perform a SELECTION SORT on  *
'an array holding 16 random integers and     *
'output the ordered list and the number of   *
'swaps and passes.                           *
'                                            *
'December 2004                               *
'*********************************************
Setup                                    'Main program procedures
Populate_List ListA()
```

```
Selection_Sort ListA(), ListB()
Output_Results
End Sub
Private Sub Setup()                          'Initialise variables
  Randomize                                  'and clear output boxes
  PicList.Cls
  PicResult.Cls
  PicPasses.Cls
  PicComp.Cls
  PicList.Print
  PicResult.Print
End Sub

Private Sub Populate_List(ByRef List())      'Fill array with
  Dim Fill As Integer                        '16 random numbers
  PicList.Print " ";
  For Fill = 0 To 15
    ListA(Fill) = Int(99 * Rnd) + 1
    ListB(Fill) = 0
    PicList.Print List(Fill);
  Next Fill
End Sub

Private Sub Selection_Sort(ByRef ListA(), ByRef ListB())
Dim Outer As Integer, Inner As Integer, Minimum As Integer, Temp As Integer
Comparisons = 0
Passes = 0
For Outer = 0 To 15
   Passes = Passes + 1
   Minimum = Outer
   For Inner = 0 To 15                       'Modification for two arrays
     Comparisons = Comparisons + 1
    If ListA(Inner) <= ListA(Minimum) Then   'Perform linear search
     Minimum = Inner
    End If
   Next Inner
   ListB(Outer) = ListA(Minimum)
   ListA(Minimum) = " "
Next Outer
End Sub

Private Sub Output_Results()
Dim Count As Integer
PicResult.Print " ";                         'Output results
For Count = 0 To 15
  PicResult.Print ListB(Count);
Next Count
PicPasses.Print Passes
PicComp.Print Comparisons
```

```
End Sub

Private Sub cmdExit_Click()
End
End Sub
```

Code 7.4: Selection sort

Program notes

The program uses two arrays, one to hold the original list and the other empty. The main procedure is `Selection_sort ListA()` that performs a linear search on the data to find the minimum (or maximum) value. Once this is found on the first pass the variable `Minimum` is set to the smallest value. This value is transferred to `listB` in position dictated by the value of the loop variable outer. `ListA(Minimum)` is then set to a dummy value, " ", to maintain the integrity of the array. Scanning then repeats for the next smallest item which is transferred to `listB` until the sorted list is in `listB`.

A screenshot from a program run is shown in Figure 7.5:

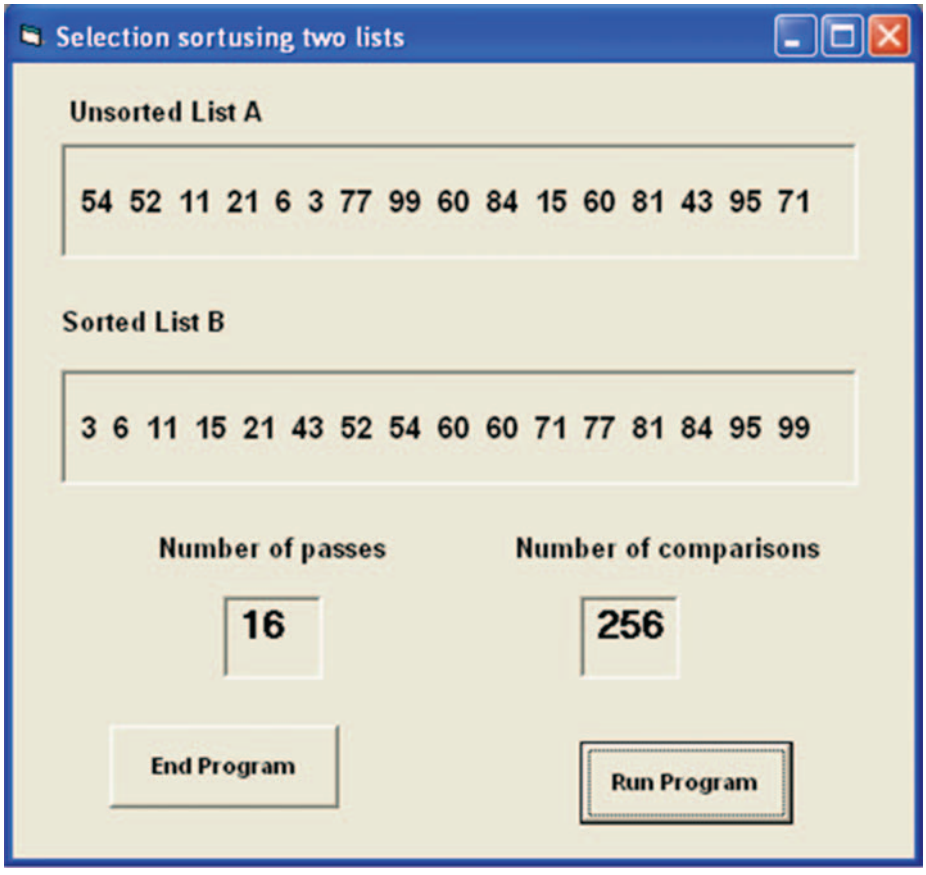


Figure 7.5: Selection sort output

Analysis of the selection sort

The selection sort is one of the simplest of the three but is the worst algorithm in terms of comparisons made. For the two arrays version above this equates to N x N comparisons, which for a 16 item array means 256 comparisons. It also requires N iterations to complete the sort.

There are no memory overheads with this sort.

7.6.7 Summary of three sorting algorithms

Neither of the algorithms is efficient at sorting but they are easy to understand and implement. The criteria for measuring algorithm performance are:

1. Behaviour with different size lists
2. Memory requirements
3. Stability

Stability of an algorithm is simply its ability to maintain the order of data items whenever equal data items are encountered and their order is unchanged. This is important in databases where the order of records is of importance.

Options 1 and 2 can be summarised in Table 7.7:

Table 7.7: Comparison of sorting methods

	Simple sort	**Bubble sort**	**Selection sort using two lists**
Comparisons	N(N -1)/2	N x N	N x N
Passes	N	N	Negligible
Memory	Negligible	Negligible	Small
Uses	Small lists	None	lists
Stability	Stable	Stable	Stable

Playing around with sorting

Try the methods of sample, bubble and selection sort using plastic counters with numbers written on them. This way the counters can be easily moved by following the instructions in each algorithm. This can work for search routines as well. It is a good idea to use coloured counters for key items of data and pointers.

Investigating sorting using special test cases

Try the sort algorithms on data according to the following criteria:

1. A set of data items in numerical order
2. A set of data items in reverse numerical order
3. A set with one item out of place, i.e. 10, 20, 30, 40, 50, 5, 60, 70, 80, 90
4. A set with some degree of order i.e. 10, 12, 19, 1, 6, 9, 5, 11, 21, 25

The three sorting algorithms will need to be modified to accept user input before attempting this exercise.

Should you wish to explore the topic of sorting further, the following links offer excellent information with animations::

http://www.cs.brockport.edu/cs/java/apps/sorters/bubblesort.html

http://www.ship.edu/~cawell/Sorting/selintro.htm

http://maven.smith.edu/~thiebaut/java/sort/demo.html

The following link offers discussions on various sort algorithms:

http://atschool.eduweb.co.uk/mbaker/sorts.html

7.7 User-Defined module libraries

By the time you have reached this stage your programming expertise should be second to none! You will have probably found that throughout your programming experiences there were occasions when you said to yourself "I'm sure I've done this procedure before " or "can I use the function or code that I used in a program last week in the program I'm writing at the moment?".

The answers to both are probably yes! Software developers have at their disposal vast arrays of programming resources and development tools to help them build programs modules and applications in the least time possible. Much of these resources will be module libraries that are depositaries of useful software procedures, functions, subroutines, programs, applications, OS routines, objects, classes, type declarations etc. If they are all packaged as a DLL file (**Dynamic Link Library**) then they can be used within most programming environments simply by calling them up. Windows itself is composed of many DLL files.

A DLL file contains executable code and will link to a programming application at run time rather than at compile time. This means that DLL files can be updated independently of the applications that call them.

The use of DLLs is fine for large, serious programming applications that contain many thousands of lines of code. Also, working with DLL files in Visual Basic can be extremely confusing and difficult to implement without the use of windows Application Programming Interface (API) and so is beyond the scope of this topic.

For our purposes there is no need for this level of complexity. There are more undemanding methods for creating a module library.

7.7.1 Creating a module library

A simple method can be implemented using the following steps:

1. create a new folder and give it a name such as "Module Library"
2. save Visual Basic procedures, sub routines and functions as modules with the .BAS extension to this folder
3. call procedure, sub routine or function when required from VB program

As well as the module library storing BAS files it can also store forms, variable declarations and text files.

The following worked examples will show the steps involved.

Examples

1. Worked example 1

In this example a small procedure that outputs blank lines is saved to the library as a Visual Basic module. The module is saved as a .BAS file. This is a useful procedure in formatting output.

1. Create `c:\Module Library` folder.
2. Save the following example procedure code to this folder as `Blanks.bas`

```
Sub BlankLines(Lines As Integer)
  Dim count
  For count = 1 To Lines
    Form1.Print ""
  Next count
End Sub
```

3. To run the code `Blanks.bas`:
   1. open a new project in Visual Basic
   2. click on Add Project
   3. choose Add Module then Existing Module
   4. open module folder to show a screen like Figure 7.6

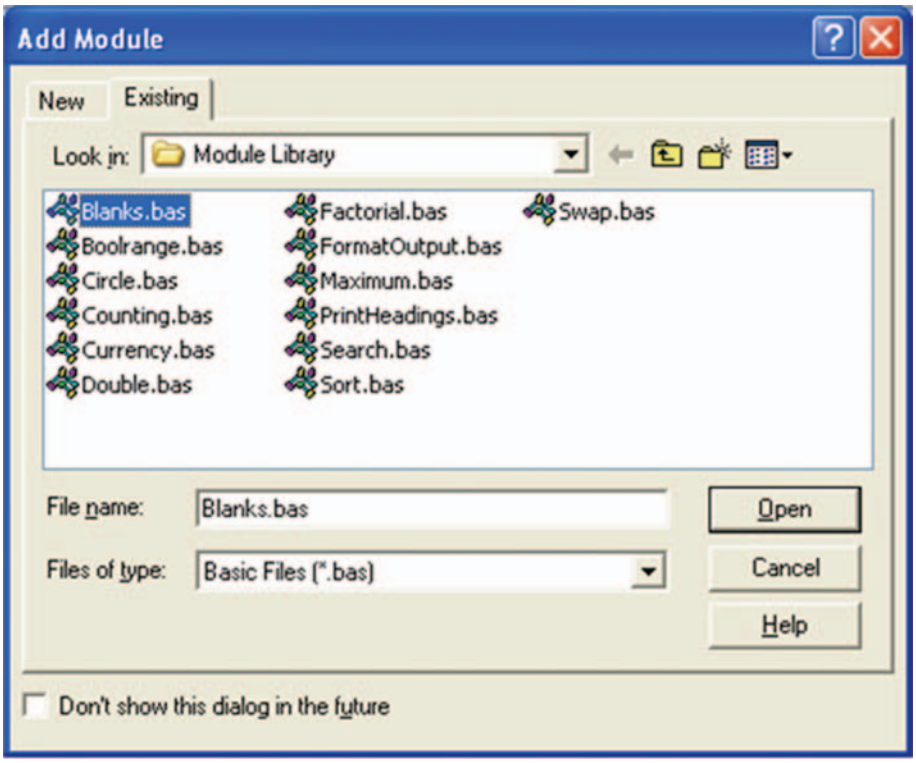


Figure 7.6: Module library folder

5. choose the file Blanks.bas

4. Create a form with a single button to execute the following code:

```
Private Sub CmdRun_Click()
  'Program to produce line spaces
  Print "Heading1"
  BlankLines 2                          'Procedure call
  Print "Sub Heading 2"
  BlankLines 3                          'Procedure call
  Print "End"
End Sub
```

Run the program a few times to show that it works.

Notice that in the module code to enable print output the statement `Form1.Print ""` was used to direct output to Form1. Modules do not have the properties of inputting or outputting data.

2. Worked example 2

In this example a small procedure is used that swaps two numbers input by a user. This is a useful procedure in sorting routines.

1. Save the following procedure code to the module folder as `Swap.bas`:

```
Sub SwapValues(Number1 As Integer,Number2 As Integer)
  Dim temp As Integer
  temp = Number1
  Number1 = Number2
  Number2 = temp
End Sub
```

2. Create a form with a single button to execute the following code:

```
Private Sub Command1_Click()
  'This program swaps two values
  Dim value1 As Integer, value2 As Integer
  value1 = InputBox("Input first value")
  value2 = InputBox("Input second value")
  Print "Value1 = "; value1; "  Value2 = "; value2
  If value1 > value2 Then
    SwapValues value1, value2         'Procedure call
    Print "Value1 = "; value1; "Value2 = "; value2
    Print "Values swapped!"
  Else
    Print "Values not swapped!"
  End If
End Sub
```

The user is asked to input two values. The SwapValues procedure is called from the module and executed is value1 $>$ value2. The values are printed out before and after any swaps with a simple message.

Run the program a few times to show that it works.

3. Worked example 3

This example uses a function to convert from one currency into another. The steps are identical to the other exercises.

1. Enter and save the following module code as `Currency.bas`:

```
Function Convert(Amount As Currency, ConversionFactor As Double) As Currency
  Convert = ConversionFactor * Amount
End Function
```

2. Enter and save the program code:

```
Private Sub CmdRun_Click()
  'Simple Currency converter
  Dim Pounds As Currency, Dollars As Currency, ExchangeRate As Double
  Pounds = 56.87
  ExchangeRate = InputBox("Enter exchange rate ")
  Dollars = Convert(Pounds, ExchangeRate)
  Print Pounds, Dollars
End Sub
```

3. Open the module Currency.bas and run the program

You can also save your procedures, functions etc. as text files in the module library. To add code in text format to a program go to `Edit` then `Insert File`. You will see a screen like that of Figure 7.7:

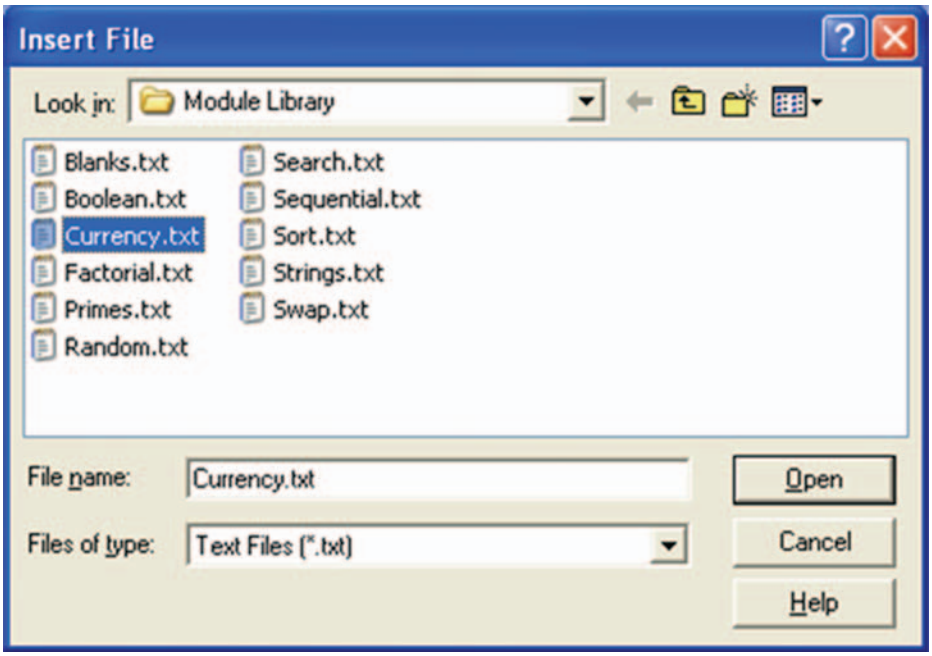


Figure 7.7: Insert file

Choose the file you wish to insert and click on `Open`. The file will then be inserted into the current program code.

The examples given above have been fairly basic but the main issue was to show how a self-created module library may be used. You should now be able to create your own library and save your much-used routines.

7.7.2 Review questions

Q3: The local golf club has 500 members. You have been given a list of the members' handicaps and are required to sort the list into decreasing numerical order for the club captain so that the forthcoming round of competitions can be organised. From your knowledge of sorting algorithms which one would you choose to sort the handicap list? Explain the algorithm.

Q4: Explain what is meant by a user-defined module library.

7.8 Summary

This has been a fairly practical topic. Standard algorithms were revisited and you should now understand the nature of linear and binary search routines and the differences between them. Equally you should have an understanding of three different sort algorithms, how they operate and their differences under various criteria. Finally a look at module libraries and their implementation rounds off the topic in a practical sense.

By the end of this topic you should now be aware of the following objectives:

- describe and compare simple linear and binary search algorithms;
- implement a binary search;
- describe and compare sort algorithms for simple sort, bubble sort and selection sort in terms of number of comparisons and use of memory;
- implement simple, bubble and selection sort algorithms;
- describe and exemplify user-defined module libraries.

End of topic test

Q5: The following questions refer to the list:

Aberdeen, Bath, Coventry, Edinburgh, Glasgow, Hamilton, London, Manchester, Oxford, St Andrews, Tain, Wick

Using a linear search how many items will be examined before *Oxford* is found?

Q6: Using a binary search how many items will be examined before *Oxford* is found?

Q7: You have a file of 10,000 records to sort into ascending order based on a key field. Which one of the following sort algorithms would not be suitable?

a) Bubble sort
b) Simple sort
c) Selection sort
d) None of these

Q8: What is the significance of DLL files in programming?

Q9: During program execution DLL files have to be compiled as part of the main program.

a) True
b) False

Q10: User-defined module libraries are simple to implement.

a) True
b) False

Q11: The following algorithm represents a sort routine.

```
1.  for outer = 0 to n - 1
2.    minimum = outer
3.    for inner = 0 to n
4.      if list_A(inner) < list_A(minimum) then
5.        minimum = inner
6.      end if
7.    next inner
8.    list_B(outer) = list_A(minimum)
9.    list_A(minimum)  =  dummy value
10. next outer
```

Which one does it represent?

a) Simple sort
b) Bubble sort
c) Enhanced bubble sort
d) Selection sort

Q12: What is the signifcance of using a dummy value in line 9?

Q13: How could the algorithm be modified to reverse the order of the output?

Q14: Searching and sorting routines can be performed easier using a 4GL

a) True
b) False

Topic 8

End of Unit Test

Contents

An online assessment is provided to help you review this topic.

Glossary

acceptance testing

The testing of software outside the development organisation and usually at the client site. Also referred to as beta testing

alpha testing

Testing phase of software within the development organisation.

anonymous variable

An underscore character (_) used in a Prolog query to indicate a variable whose value is not required.

assertion

Adding data to a Prolog database.

asynchronous programming

Another term for event-driven programming.

base class

Another name for a super-class

binary chop

Repeated division of a data list by halving to produce sub-lists for searching or sorting.

black box testing

A phase in testing where focus is on the inputs and outputs of a software system, rather than the code.

break-even point

The situation where, in running a new software system the benefits just begin to outweigh costs.

CASE Repository

A CASE database of software project information that can be shared by software developers.

circular queue

A list structure where the front pointer and rear pointer coincide

class

A group of objects that share common characteristics.

class libraries

Collections of classes that can be used in software development, as building blocks for further class definitions.

Client

The person or company that initiates the development process by specifying a problem.

code generator

Automatic generation of source code from analysis and design specifications.

Combined conversion

A mixture of parallel, phased, pilot and direct conversion methods.

Component testing

Part of the testing phase that involves the building blocks of programs such as procedures.

consultants

Experts recruited by the client company to engage in a feasibility study.

cost-benefit-analysis

Part of the feasibility study comparing project development costs with future benefits.

Data dictionary

A repository of data definitions, system components etc., used by upper CASE tools in software development.

derived class

A new class that is created from an existing class, inheriting all the characteristics of the existing class.

Direct conversion

This is the complete changeover from the old software system to the new. This 'big bang' approach attracts the greatest risks.

dispatcher

Part of the operating system that is responsible for executing event handlers.

divide and conquer

Philosophy behind binary halving of data lists to speed the operations of searching and sorting.

document generator

A CASE tool that generates documents directly from comments in programs.

driver

A small program written specifically to supply an un-tested module with test data and an interface.

encapsulation

The binding of data and methods within an object. Data cannot be changed within objects by other objects.

event-driven

A system that responds to an external event such as a mouse click or a key press.

event handlers

Small procedures that respond to events in event-driven programming.

event queue

Multiple triggered events are put into a queue system to wait processing.

executable prototype

An executable source code program obtained directly from the analysis diagrams and specifications in UML

feasibility study

Research by the project leader into whether a project is viable enough to go ahead.

FIFO

First in first out stack structure.

functional specification

This details how the developed program will behave under specified conditions.

information hiding

Another term for encapsulation.

inheritance

The sharing of characteristics between a class of object and newly created sub classes. This allows code re-use by extending an existing class.

inheritance diagram

A diagrammatic representation of an inheritance class hierarchy.

instance

An object is an instance of a class if it belongs to that class.

instantiation

This is the process, in Prolog where a variable is temporarily assigned a value.

LIFO

Last in first out queue structure.

link loader

Part of a compiler that links the object code with source code during compilation.

Lower CASE tools

Collections of programming tools that deal with the implementation, testing and maintenance phases of software development.

macro generators

Assembly language that translated a single line of code into multiple lines of machine code.

Module testing

Testing of collections of procedures, functions etc., that can individually compiled and executed.

operational requirements document

A document describing what a system must be able to do in order for it to meet user requirements.

Parallel conversion

An implementation phase where the old software system is run in parallel with the new software system.

Phased conversion

Similar to parallel conversion but the conversion is done over a larger time scale using smaller parts of the program at a time.

Pilot conversion

This is where portions of the old software system that are known to work are tested in the new environment. This is the safest conversion route.

polymorphic

Two objects derived from a base class are said to be polymorphic in that they can respond to the same message in different ways

postfix notation

Another name for Reverse Polish notation

project leader

The person responsible for overseeing a project from start to finish.

project proposal

A report usually compiled by management of the client organisation outlining the nature of the project in terms of scope and objectives.

queue

A dynamic data structure like a double-ended stack.

Rational Unified Process

A development process, initiated by IBM to deliver best proven practices during each stage of a project.

record

Smallest data type that can be part of a random access file.

reverse engineering

Production of analysis and design diagrams directly from program source code.

Reverse Polish

Notation where operand comes after the operators. For example X + Y becomes X Y +

stack

A dynamic data structure much used by software applications and the computer for storing temporary data. Only access is via the top of the stack.

stack overflow

Situation where a stack becomes full.

stack pointer

Register that hold the address of the top of a stack.

stack underflow

Situation where a stack becomes empty.

stub

A temporary addition to a program used to assist with the testing process.

sub-class

A class member of an existing class system.

super-class

A super class contains instances of sub-classes. Also called a base class.

system investigation

This is the post feasibility study phase where a more detailed exploration is undertaken.

systems analyst

One of the consultants responsible for analysing and determining whether a task is worth pursuing, using a computer. He/she is also responsible for the design of the computer system.

test-harness

Another name for a driver in the context of software testing.

Unified Modelling Language

A standard object-oriented diagramming system suitable for CASE tools

Upper CASE tools

Collections of programming tools that deal with the analysis and design phases of software development.

validation

Validation takes place when data is checked to see if it makes sense. For example are the inputs to a program within the specified range, is the date format correct?

verification

This is a process that determines if data is correct. For example a name may be input wrongly

white box testing

A phase in testing where focus is on the structure of the code, rather than its function.

Answers to questions and activities

1 Software Development Process

Answers from page 10.

Q1: The scope of a project entails its size, complexity, costs and time constraints.

Q2: Financial data (sales), Personnel data (payroll)

Q3: A project proposal document is a written report containing details of a planned project. Items that company management might include in a project proposal document: Company profile, nature of the project, timescale, additional requirements.

Answers from page 14.

Q4: A feasibility is the first stage where the scope and objectives are outlined. The aim is to ascertain whether the project is viable. The project leader usually carries this task.

Q5: Economic, Technical, Legal and Scehdule. Economic feasibility deals with the costs involved in any solutions and any cost- benefits that might be achieved. Technical feasibility deals with determining the technologies required to solve the problem and whether these are currently available and costs are within the project budget. Legal feasibility deals with issues such as the affect of the new system on contracts of work, conditions of employement, liability and Data Protection.

Q6: The client must take the decision to proceed with any further developments. The report will help them to take this decision. The report may form the basis of the next phase of system investigation and will be useful to the developers.

Q7: The analysis of a project in terms of all the costs involved in setting up system and maintaining it in the long term. There comes a point when the system begins to make money.

Q8: To plan and schedule projects involving concurrent tasks.

Answers from page 18.

Q9: This is to determine in some detail what has to be done to solve the stated problem. This is carried out by a systems analyst.

Q10: Reluctance of personnel to talk because they might feel threatened by a new system, fear of losing jobs, wary of anything new.

Q11: The ORD contains a full, detailed description of the problem including inputs, processes and outputs. It is used throught the development process. It is significant since it, if it accepted by the client then it is legally binding.

Q12: It describes exactly what the system has to do and how it should behave.

Q13: Structured design is a technique which emphasises breaking large and complex tasks into successively smaller sections.

Answers from page 23.

Q14: Any four from: Code the system, test the system, set up hardware on client site, produce documentation, train staff

Q15: Black-box testing treats the components of a program as black boxes. You do not need to know what is inside a black box: all you need to know is what output you should expect for a given input. In white-box testing, you know about the workings of the program; you can see the code inside the modules. When you devise your test cases, they are on the code - contrast this with the black-box form of testing, where you need have no idea about the code at all.

Q16: The input, reason for the test case or a statement of what is actually being tested, the expected outcome and the actual outcome.

Q17:

1. The old system runs along the new system until the newer version can take over.
2. Only a small section of a company will use the new system to evaluate its worth
3. Implementation of the new system can be planned over a few months when decisions can be made as to its viability
4. The new system is started as soon as possible probably over a weekend to allow for minimum disruption

Q18: How closely the does the solution match the specification? Is the solution what the clients were looking for?

End of topic test (page 24)

Q19:

A) Definition of the problem
B) Feasibility study
C) Collecting the information requirements of the system
D) Analysis of the requirements of the system
E) Design of the system
F) Implementing and evaluating the system
G) Maintenance of the system

Q20: Implementing and evaluating the system

Q21: feasibility study

Q22: Analysis of the requirements of the system

Q23: Implementing and evaluating the system

Q24: b) Structure chart

Q25: a) True

Q26: Data which is used to determine if software works properly

Q27: d) Black box

Q28: Perfective

2 Interface Design

Answers from page 32.

Q1: Where speed is of importance, for example in real time systems. Expert user rather than novice. Lack of powerful hardware resources, for example memory, display.

Q2: Advantages: faster to implement commands, memory overhead is minimal, users have more control.

Disadvantages: usually poor help screens, commands can be quite complex to remember, restricted to keyboard input.

Q3: Microsoft DOS, UNIX, Linux

Answers from page 42.

Q4:

a) No commands to learn, objects are easy to manipulate using a mouse and pointer, vast availability of help screens, menu system only allows valid options.

b) Slower to operate and more time consuming for user because of a greater number of options to choose from. Takes up more resources (disk space and memory).

Q5:

1. For the computer application, you will no doubt have identified "the usual" components of a modern graphical interface: a main window showing the current state of the document (if the application is a word processor, for example), " close", "minimise" and "maximise" buttons on the window, as well as scroll bars and pull-down menus, possibly one or more tool bars, and perhaps one or more "palettes" (movable tool-bars). The output and feedback is all on the display screen (CRT or flat-screen LCD panel). The primary interaction mode is visual, with symbolic (text) input on the keyboard and mouse (which can also be considered to be provide tactile and gestural input).
2. For the phone, there is a small, usually monochrome, screen. On a cordless fixed-line phone this has room for only one line of text, whereas a mobile can usually display 4 or 5 lines, plus simple graphics as well. Input is symbolic through the keypad and a few extra buttons. The dominant mode of course is speech, but symbolic tactile input is used when dialling, and also to provide data (for example, a credit card number when ordering cinema tickets) and possibly text (when inputting a short text message for transmission).
3. An appliance like a microwave oven has a range of buttons and possibly dials, an LCD display and a bell or ringer which for example sounds when the set cooking period is complete. There is also secondary auditory feedback from the fan which runs when cooking is in progress, and stops when it's finished. The primary mode of interaction is tactile symbolic, with auditory and visual feedback through indicator lights, a small LCD display (or manual clock on older models), and perhaps a rotating platform inside the oven.

Answers from page 45.

Q6: Here are a couple of examples which may be among those you have identified:

1. The "Print" dialogue box, activated by selecting "Print" on the "File" menu on a wide range of PC or Mac applications: the exact details will depend on what kind of printer is attached, and whether it is attached directly or accessed over a local area network, but it is likely to include radio buttons (e.g. for selecting print quality, whole or part document printout, etc.), graphical buttons (e.g. for selecting page orientation, landscape or portrait), text boxes (e.g. to specify start and finish page numbers, if partial output is selected), and maybe also sub-dialogues (e.g. for "Advanced" features). It is likely that default selections will already have been made when the dialogue box appears, and these are the ones that will prevail if you immediately hit the OK button.
2. The "pull-down" menus ("File", Edit", "View" etc.) which appear at the top of every window in all versions of Windows, and at the top of the screen in MacOS: these illustrate the range of elements and behaviour which is part of an almost universal visual language of graphical interaction across a range of graphical user interfaces. These include hierarchical side-menus (e.g. "New" on Windows itself activates a sub-menu to select "File", "Folder", etc.), sub-dialogues (where selecting the menu item brings up a dialogue box, rather than carrying out the selected command immediately - an example is "Print" referred to above, and "New" on some applications), dynamically added menu elements (e.g. a list of recently -accessed files at the bottom of the File menu), keyboard shortcuts (displayed alongside some menu item labels), etc.

End of topic test (page 45)

Q7: The component of the system which facilitates interaction between user and the system.

Q8: Use of menu systems to find programs and other features quickly; Online help facility; Use of WIMP to make it easier for the user.

Q9: Graphical user interface: user clicks on icons to perform tasks. Command-driven: user types in commands or shortcuts. GUI: easier for novices to use the system. Command: faster for experts once the commands are known.

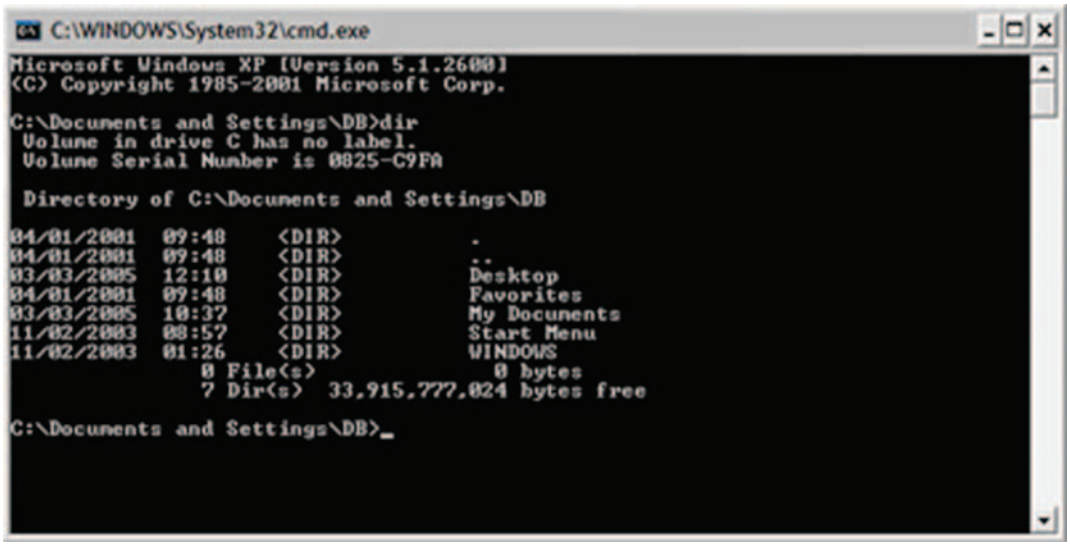
Q10: b) False

Q11: d) None of the above.

Q12: b) Novice users find GUIs intimidating since there are too many options.

Q13: d) None of the above.

Q14: a) MS Dos


C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\DB>dir
Volume in drive C has no label.
Volume Serial Number is 0825-C9FA
Directory of C:\Documents and Settings\DB
04/01/2001 09:48 <DIR> .
04/01/2001 09:48 <DIR> ..
03/03/2005 12:10 <DIR> Desktop
04/01/2001 09:48 <DIR> Favorites
03/03/2005 10:37 <DIR> My Documents
11/02/2003 08:57 <DIR> Start Menu
11/02/2003 01:26 <DIR> WINDOWS
0 File(s) 0 bytes
7 Dir(s) 33,915,777,024 bytes free
C:\Documents and Settings\DB>

3 Software development languages and environments

Exercise (page 54)

Object	Data	Operations
Button	Size, name, colour, position	Drag, resize, drop
Text box	Size, position, background colour, border	Create, drag, resize
Window	Size, position, minimised, overlapping	Maximise, minimise, resize, drag
Dialogue box	Size, name, position	Add text, delete text, resize

Answers from page 55.

Q1: Increasing complexity of programs produced problems in managing and maintaining them.

GUI environments cannot be programmed by the constructs of conventional languages. Difficulty in coding program modules in isolation and the lack of integrity of data using global variables.

Q2: It must exhibit encapsulation, inheritance and polymorphism.

Q3: An object is a unit that contains both data and the operations or methods on that data. It is represented in the language as a single entity with three parameters: name, attributes and operations.

Q4: Attributes: size, shape, colour, position, target type, target location, run mode (normal, maximised, minimised)

Operations: highlight, drag, drop, execute, delete, rename

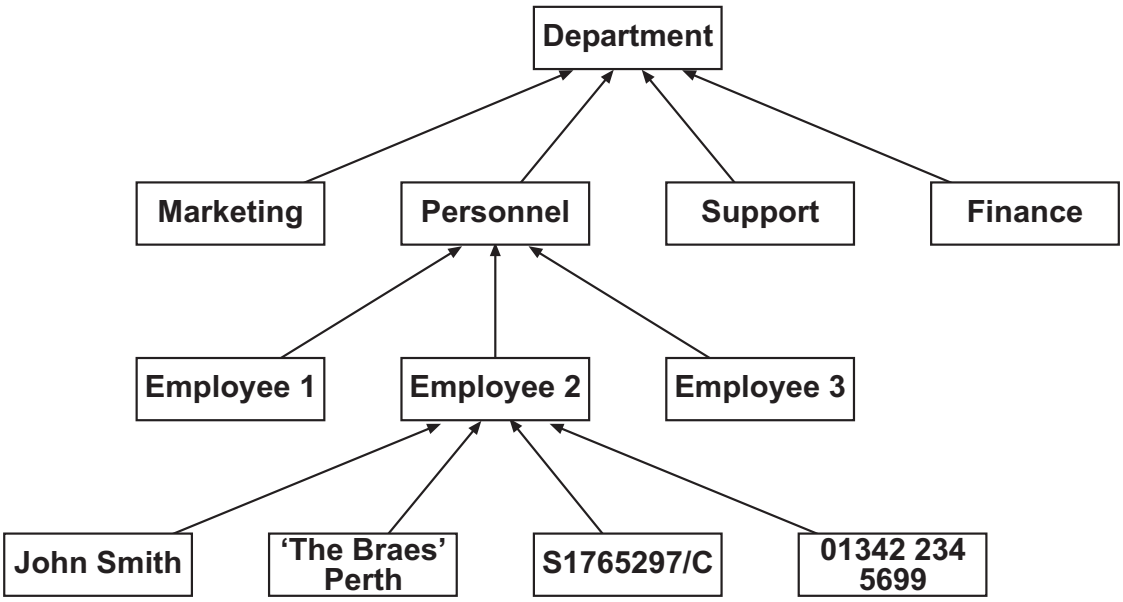
Exercise (page 56)

You could extend the diagram to include:

- 3-door hatchback, 5-door hatchback, petrol driven or diesel driven, 1200cc or 2000cc
- Petrol driven or diesel driven, automatic or manual

Exercise (page 57)

A possible solution would be as shown here:



Answers from page 57.

Q5: A class is a collection of objects each of which have common characteristics.

Q6: Data: accountNumber, customerName, balance, creditRating

Methods:openAccount, debitAmount, CreditAmount, provideBalance

Q7: Inheritance means that new classes can be created by extending existing classes. The new classes will have all the properties of the existing classes. This promotes the reuse of code that can be used elsewhere in other programs.

Q8: Any three from: C++, Visual Basic, Delphi, Java, Simula, Smalltalk, Eiffel, Python, Perl, C#

Answers from page 60.

Q9: Both data and operations are held together as a single entity. The only access to the data is by means of the print statement - *Print RegNumber*. Otherwise the data is hidden from further operations and so cannot be altered.

Q10: A message is the way that objects interact with one another. The only aspect that a message can see in another object is the interface. The interface allows values to be passed to and fro between objects using parameters but the objects' data remain intact.

Q11: Polymorphism is where sub-classes can respond differently to identical messages from the parent or super class. Same syntax, different semantics. Examples could include: identical buttons executing different programs, dialogue box receiving text of different colours, font, size etc.

Exercise (page 63)

Eiffel, Java, Smalltalk, Ruby, Python, Perl

Exercise (page 64)

1. `planet(Planet, _,Distance,_,_,_),Distance>100`
2. `planet(Planet,small,_,atmosphere,none,Moons),Moons>0`

Answers from page 66.

Q12:

Agree: Generally speaking the statement has turned out to be true. There are now much more object-oriented languages in use today, as a search of the literature will prove. Also some of the older type languages have been enhanced to allow them to employ object-oriented concepts. Many computing processes are now adopting object-oriented approaches such as the various programming tools available for software development. Not only that but a program design can be object-oriented even although the resulting program is not and conversely, a program can be object-oriented allowing for the fact that the language it is written in, is not.

Disagree: Object-orientedness is quite a difficult concept to grasp and there must be many programmers who will not wish to change their programming environment just for the sake of it. Expertise and skills in a particular language do not necessarily migrate to an object-oriented environment.

Answers from page 70.

Q13: Invariably the computing environments will employ GUIs and Windows. Objects, classes and methods are a common feature of both the language paradigms; clicking on an icon, for example, the object will instigate an event that is processed by a specific handler in the event-driven language. It is also the case that event-driven languages support object-oriented constructs while many object-oriented languages are event-driven.

Q14: Low level languages execute machine code, the machine's native language. The code varies from machine to machine according to the computer's instruction set. Low level has no high level constructs so the problem cannot be stated using English key words.

Q15: Used in writing programs that execute fast such as software device drivers for peripherals and data recording systems. Also used where computer resources are minimal.

Answers from page 74.

Q16:

- The language is popular because of its relative ease of use and easy to learn; statements and key words resemble those of the English language.
- It has powerful constructs for selection, iteration and control, procedures, functions with parameter passing is standard and this would be favoured by software developers in relation to other languages.
- It can cope with static and dynamic data structures like arrays, records and lists.
- Programs are compiled rather than interpreted and it will probably have powerful debugging features as well.
- It has a fast compiler so that it can deal with large programs, and although the language is meant for the scientific field, it is good at coping with problems in more general areas.
- It can deal with legacy code in that older programs can run on the more modern versions of the language.
- It has retained its status as a popular language because programmers prefer to use it because of its many influential features.

Q17: ALGOL. Universal, general purpose language.

COBOL. Business applications.

LISP. Manipulation of data structures and artificial intelligence.

BASIC. Education

Answers from page 77.

Q18: 4GLs were developed for users to describe a problem in simple terms rather than get involved with the actual coding. They were, therefore problem oriented rather than imperative or procedural driven. This was quite a change in thinking from the conventional 3GLs where the user coded the program into the computer. They were designed for specific areas and incorporated such as SQL, RAD, and report generation.

4GLs arose from the rapid development of application programs, in particular databases, spreadsheets and graphics. Behind many applications today there will be a 4GL. All the features, functions and tools of a 4GL are invisible to the application but they can be implemented rapidly by a key press or mouse click on an icon, for example.

Q19: The term 5GL is rather diffuse. Opinions differ. However there are some pointers

- The term could refer to natural language systems that would be used in artificial intelligence and expert systems. The development of the programming language ADA was heavily promoted in America while the Japanese embarked on a Prolog-based venture, the so called "Fifth Generation" Computer Project.
- With Internet access becoming more popular languages were developed to process user interaction while online. These included scripting languages (JavaScript, VBScript) and editing languages (HTML, XML) that offered the user complete control over their actions.

- With the proliferation of GUIs it required the use of event-driven, visual programming languages (Visual C, Visual Basic, Visual FoxPro), together with object-oriented languages (C++, Java, Visual Basic), to offer the user access to, and manipulation of objects to process their requests.

Q20:

- Object-oriented: combining data and processing affords a more intuitive approach to the problem.
- Program is more reliable and easier to maintain since it consists of objects and their methods.
- Visual: allows for faster development of the program with reduced need for traditional techniques.
- Visual languages take full advantage of the powerful GUI features.

End of topic test (page 78)

Q21: Inheritance

Q22: Object-oriented: language defines the data types and the operations at the same time. Visual programming: allows easy creation and manipulation of objects such as windows, buttons, and screens.

Q23: Object-oriented: data and processing combined, programs easier to maintain using objects. Visual programming: no need for traditional methods, promotes rapid application development

Q24: d) A machine code program can be executed by any CPU

Q25: Java, Visual Basic, HyperCard

Q26: Expansion of specialised application areas; Enhanced compiler techniques allowed more complex programs to be translated. Escalation of new hardware technologies (e.g. networking, cheaper memory).

Q27: d) All of the above

Q28: b) False

4 Software Testing and Tools

Answers from page 85.

Q1: Testing of components is the testing of the building blocks of programs or modules that usually consist of procedures, subroutines or functions that can be compiled and executed on their own.

Q2: Modules are collections of components but they cannot be compiled or executed on their own. If a module is completed and ready for testing then a special program called a driver or test harness is produced that supplies it withappropriate test data and an interface under a test simulation environment

Q3: Alpha testing is performed on the program when it is just able to be run, but without much of the functionality. The testing takes place on the developer's premises

Q4: Acceptance testing, also called beta testing, is performed outside the development company by independent testers. It represents the highest level of testing in the development process. The testing should confirm that the program runs to as near as possible to the original specification.

Q5: The three processes are:

- the finished software product meets the original specification
- the software is robust
- the software is reliable.

Dry Run Exercise (page 88)

Run	a	b	sum	Validation	Output
1	6	0		False	
2	6	3	9	True	9
3	-6			False	
4	0			False	
5	2	3	5	True	5
6	3	8	11	True	11
7	1	-7		False	
8	1	4	5	True	5

The finalvalue for b is not used.

Answers from page 91.

Q6: The testing of software is to establish the presence of faults. Debugging finds and removes errors in the software

Q7: A break point is set in a program to halt at a specified variable or line so that it can be inspected. The program can then continue as normal. A program watch is set to display the value of a variable as the program is run, either in step mode or normal mode.

Q8:

Count	Number1	Count $>$ 4
0	6	False
1	6	False
2	7	False
3	9	False
4	12	False
5	16	True
6	21	True

Answers from page 99.

Q9:

- CASE tools are collections of software programs that are designed to automate the various phases of the software development cycle from analysis right through to implementation,
- More powerful tools with increased functionality were required to cater for the increasing complexity of software systems. Pen and paper methods were insufficient to maintain the increase in pace of developing new systems in shorter time scales.

Q10: Upper CASE tools are devoted to the specification, analysis and design phases of software development. They allow the automation of data flow diagramming and structure charting techniques that, under normal circumstance take time to implement. Lower CASE tools are concerned with the implementation, testing and maintenance phases of software development. They are involved more with code generation and document generation

Q11: Advantages:

- Increased speed of software development
- Better overall documentation and the use of CASE repositories (databases)
- Reduced costs and improved efficiency
- Improved communication throughout the development life cycle.

Limitations:

- CASE tools can be expensive so there has to be a cost benefit analysis undertaken.
- The choice of commercial CASE tools is increasing so choosing the right one can be a daunting experience for a company.
- Training is also an issue. This will add to the overall cost that a company will need to budget for if they opt to use CASE tools.

End of topic test (page 99)

Q12: b) False

Q13: a) True

Q14: b) False

Q15: a) Dry run

Q16: Run-time error such as infinite loop or memory problems.

Q17: To facilitate the tasks involved in the stages of the software development process and to coordinate the activities from analysis through to implementation.

Q18: d) All of the above

Q19: b) False

5 High level programming language constructs 1

Answers from page 111.

Q1: A sequential file is a text only file that stores sequences of ASCII characters, terminated by control characters, LF and CR. Once a file has been opened it can be read and written to but not at the same time.

Q2: A sequential file is created by the process of opening it. In Visual Basic the command is:

```
Open Filename For Output As #1
Close #1
```

Once the fie is open data can be written to it and the information saved when the file is closed. To open a file for reading, the following command is used:

```
Open Filename For Input As #1
Close #1
```

Q3:

```
"Vardon Hubert:", "28"X"Sayers Bill:","6"X"Player Gill:","20"X"trevino Louis:",
"1"XEOF
```

where X represents CR and LF and EOF is end of file.

Exercise (page 113)

Q4: `Weather_record(4,0)`, humidity at site 1

Q5: Value 13, maximum temperature at site 2

Extension work (page 116)

The problem can be broken down into five parts:

1. Create the array
2. Input days of the week
3. Find the day when 1st May falls
4. Input dates
5. Ouput formatted results

Create array

The 6 x 7 array can be created using the following statement:

```
Dim Month(5,6)
```

Notice that the array has been defined with no type. This is done because the array will contain both string and integer values.

Input days

The following days are input one by one into the array `Month`, in row 0:

```
Month(0, 0) = "Sunday"
Month(0, 1) = "Monday"
Month(0, 2) = "Tuesday"
Month(0, 3) = "Wednesday"
Month(0, 4) = "Thursday"
Month(0, 5) = "Friday"
Month(0, 6) = "Saturday"
```

When 1st May falls

Fortunately the 1st of May falls on a Sunday! The output of dates will therefore start at position Month(1,0) and continue over the next 5 rows.

Input dates

The dates could be read in manually much like the days but for a large amount of data this would be rather tedious. Why not let the computer do the work!

The following algorithm will input the values 1 to 31 automatically:

```
1. For Dates = 1 To 31 Step 7
2.  For Columns = 0 To 6
3.   Day = Dates + Columns
4.   Output value of Day;
5.  Next Columns
6.  Print New line
7.  Print New line
8. Next Dates
```

In line 1 the variable Dates takes the values 1 to 31 in steps of 7 because there are 7 columns.

In line 3 the variable Day increments by 1 up to 7 for row 1, then from 8 to 15 for row 2 and so on up to row 5.

Notice the semicolon ";" at the end of line 4. This is to suppress a line feed so that all 7 values are output on the same row.

Lines 6 and 7 are cosmetic to give sufficient spacing between rows for clarity of output.

Finally to overcome the problem of the program printing out values $>$ 31 that occurs in line 1 (next multiple of 7 is 35), the following code is added after line 3 above.

```
If Day <= 31 Then
Ouput value of Day;
End If
```

The complete code listing is shown in Code 8.1:

```
Option Explicit
Dim Month(6, 30)
Dim Day As String
Dim Dates As Integer
Dim Columns As Integer
Dim Test As Integer

'*******************************************
'Program to output the month of May 2005     *
'using a 2-dimensional array                 *
'It contains both string and numerical data  *
'                                            *
'November 2004                               *
'*******************************************

Private Sub cmdRUN_Click()
Dim Count As Integer
'Input days
Month(0, 0) = "Sunday"
Month(0, 1) = "Monday"
Month(0, 2) = "Tuesday"
Month(0, 3) = "Wednesday"
Month(0, 4) = "Thursday"
Month(0, 5) = "Friday"
Month(0, 6) = "Saturday"
'Output days
For Count = 0 To 6
  PicOutput.Print Month(0, Count); "   ";
Next Count
PicOutput.Print
PicOutput.Print
'Output dates
For Dates = 1 To 31 Step 7
 For Columns = 0 To 6
  Day = Dates + Columns
  If Day <= 31 Then
   PicOutput.Print Tab(Columns * 11); Day;
  End If
 Next Columns
PicOutput.Print
PicOutput.Print
Next Dates
End Sub
Private Sub CmdEND_Click()
 End
End Sub
```

Code 8.1: Month program

The program output is shown in Figure 8.1:

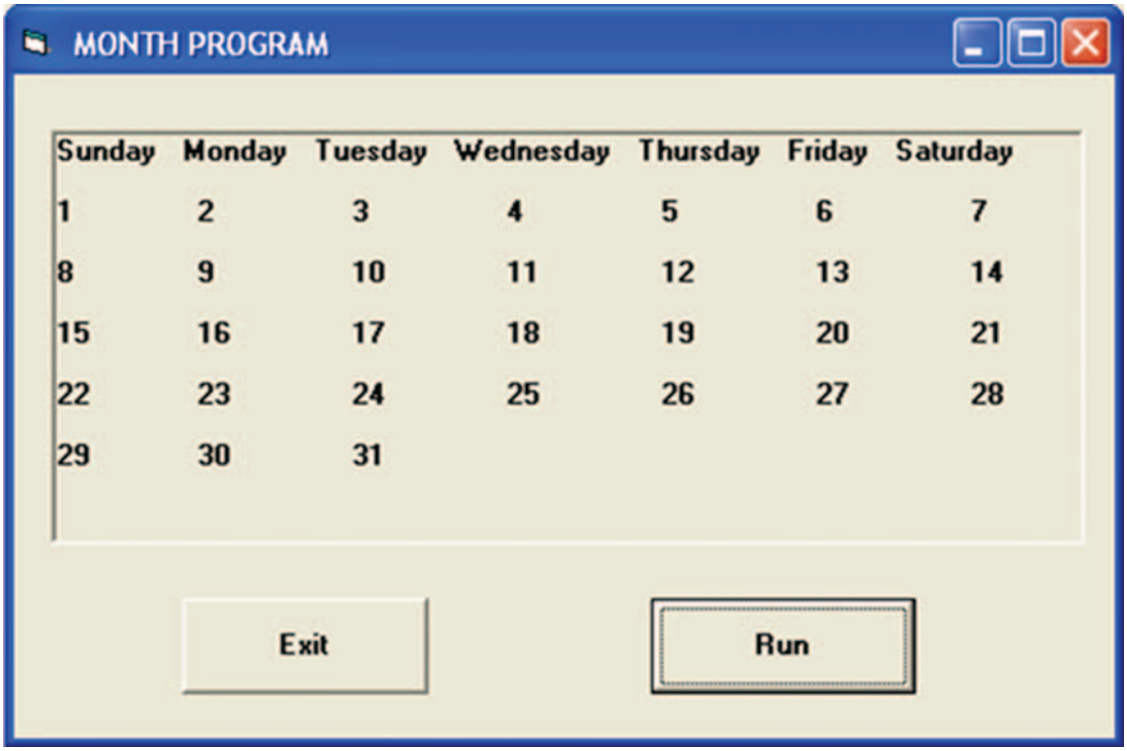


Figure 8.1: Program month output

Exercise (page 117)

In February 2005 the 1st land on a Tuesday. This means that the first line of output of the dates will run from columns 2 to 7. Also February has 28 days so output should terminate at value 28 on row 5.

Answers from page 117.

Q6: The arrays can display tables of related information that correspond to records in a database. Weather information, Ceefax pages, programme times, world temperatures, tide tables, sports league tables, bus /train timetables, crossword puzzles etc.

Q7: For a 2-dimensional array we have to consider two values; the row number and the column number. By default the row number is stated first. For a 3 x 4 array the initialisation can be done manually where each location is set to 0:

array(0,0) = 0

array(1,0) = 0

.....

array(2,3) = 0

Alternatively a looping structure can be used:

for row = 0 1 to 2

for column = 0 to 3

```
  array(row,column) = 0
 next column
next row
```

The latter method is preferable for large arrays.

End of topic test (page 117)

Q8: c) They contain text only and access is slow

Q9: b) Closed

Q10: Sequential files are read from beginning to end and so the files cannot be read and written to simultaneously.

Q11: a) 1-dimensional array

Q12: c) 2-dimensional array

Q13: b) Array(2,4)

Q14: a) Array(1,3)

Q15: d) 15

Q16: To empty the locations of old information that might corrupt new data

Q17: b) False

6 High level programming language constructs 2: Data structures

Exercise 1 (page 122)

```
1. Push 88;
2. Stack pointer = Stack pointer + 1;
3. Pop 88;
4. Stack pointer = stack pointer - 1;
5. Pop 17;
6. Stack pointer = stack pointer - 1.
```

Exercise 2 (page 122)

Q1: Push A, Push B, Push C, Pop C, Pop B, Push D, Pop D, Push E, Pop E, Push F, Pop F, Pop A

Q2: Only 1 and 2 are possible.

Extension Work: Reverse Polish notation (page 123)

Q3: `a b + c / d e - +`

Q4: `7 3 - 9 + 4 5 + 2 / x`

Answers from page 128.

Q5: A stack is a dynamic data structure that can be represented by an array in memory. It is usually of a fixed size and is an example of a LIFO structure meaning that the last item to be added to the top of the stack is the first out. A stack pointer is a register that contains the address of the top of the stack, which is movable. Items are added by pushing and removed by popping; an input stream a,b,c,d,e will be popped in reverse order e,d,c,b,a. Two important error conditions are stack overflow where the stack becomes full, and stack underflow where the stack becomes empty. Both conditions are tested before a stack is used.

A stack is useful in computing since it can store temporary results of arithmetic operations, procedure, function and subroutine calls during programming and evaluating mathematical expressions to produce postfix or Reverse Polish notation. It is also used for storing the contents of registers during the processing of interrupts and DMA routines.

Q6: A queue, like a stack is a dynamic data structure that can be represented by an array in memory. It is usually of a fixed size and is an example of a FIFO structure meaning that the first item to be added is the first out. A queue has two pointers, a front pointer that signifies the front or top of the queue and a rear pointer that indicates the rear of the queue where items are added. The input stream a,b,c,d,e will be output in the same order. Like a stack the queue will have a fixed size in memory. If the front and rear pointers are identical then the queue is empty. If the front pointer indicates the maximum size allowed then the queue is full. A circular queue is formed if the end of

the queue has no more room for items to be added. In this case the items are added to the front of the queue instead.

A queue is useful in computing since it hold items waiting to be processed. These could be print jobs, scheduled tasks in a multitasking environment or events in event-driven programming.

Exercise (page 139)

The code shown in Code 6.3 will open an existing file and output the records to a form.

Answers from page 140.

Q7:

- A sequential file can only be opened for reading or writing but not both. A random access file is capable of being read and written to at the same time.
- Random files are less efficient in memory usage but are faster to retrieve data.
- Sequential files are composed of text only whereas random files can store mixed data in the form of records.

Q8: A record is collection of related items stored under fields that represent the type of data in the record (string, numeric, character). It is the smallest unit of information that can be processed by a random access file.

The length of a record, and hence a file is used by the computer filing system to allocate sufficient space for the file. If the records are stored in equal chunks then it is easier and faster for the filing system to retrieve the information.

Q9: Example field sizes could be:

25 + 15 + 10 + 1 + 40 + 20 + 10 + 12 + 10 + 30 = 173 bytes

100,000 x 173 = 17,300,000 bytes

= 17,300,000/1024 x 1024

= 16.5 Mb

End of topic test (page 141)

Q10: c) a record

Q11: d) a stack

Q12: Overflow and underflow

Q13: c) Queues and stacks are dynamic data structures

Q14: a) True

Q15: b) 250Kb

Q16: a) Stack

7 Standard algorithms

Answers from page 152.

Q1: A binary search starts by finding the mid point of the list of items. The search key is then compared to the value at the mid point location. Depending on whether the search key is greater than or less than this value, either the top half or the bottom half of the list is discarded. The mid point of the remaining half is established and the algorithm repeats itself (recursion) continually dividing the list into two until the search key is found.

Q2:

- Linear search is slower than a binary search
- Linear search can operate on unordered lists, binary lists must be ordered
- Linear search is simple to code and implement, binary search more complex
- Average search length for linear is N/2 and for binary log2N where N is the number of data items

Answers from page 172.

Q3: Probably the simple sort would be chosen.

1. Starting with the first two items compare them. If the second is greater than the first, swap them.
2. Compare first item with third and if the third is greater than the first, swap them.
3. Repeat comparing first item with successive values and swap, if required, until end of list is reached
4. Now repeat the process starting with second item and compare with successive values until end of list.
5. Continue comparing each item with remaining items until the list is sorted.

This will produce the list, highest value first.

Q4: A user-defined module library is a collection of useful routines such as procedures, functions, subroutines, programs, modules etc. that save the programmer from re-writing code every time any of the above are required. They are commonly stored as DLLs and can be called at any time during the execution of a program.

End of topic test (page 173)

Q5: 9

Q6: 2

Q7: d) None of these

Q8: A collection of stored routines that can be called by a program at run time or a collection of user-defined modules.

Q9: b) False

Q10: a) True

Q11: d) Selection sort

Q12: To flag locations that take no further part in the sort process

Q13: Search for maximum value instead of minimum

Q14: a) True