## In this chapter you will learn to:

- determine the inputs and outputs required for a particular problem
- produce an IPO diagram from a set of specifications
- develop a systematic approach to the development of software solutions
- document a proposed non-complex software solution
  - represent the flow of data through a system using a context diagram
  - represent a system using a data flow diagram (DFD) to show its components and the data transferred between them
  - represent a system using a structure chart to show the interrelationship between the component modules
  - represent a system using a systems flowchart to show its component modules, files and media
- interpret and use an ASCII table
- identify the maximum decimal value that can be stored in a given number of bits
- recognise the impact of the use of an inappropriate data type
- select the most appropriate data type for the solution to a particular problem and discuss the merit of the chosen type
- create a data dictionary which defines the data appropriately
- identify control structures in an algorithm
- interpret and create algorithms represented in both pseudocode and flowcharts that use standard control structures
- detect logic errors in an algorithm by performing a desk check
- gather solutions from a number of sources and modify them to form an appropriate solution to a specified problem
- represent code from different sources as an algorithm to assist in understanding its purpose and to assess its relevance in a proposed solution
- incorporate a stub for modules for which the detail has not yet been developed

## Which will make you more able to:

- describe and use appropriate data types
- describe the interactions between the elements of a computer system
- describe the effects of program language developments on current practices
- identify the issues relating to the use of software solutions
- investigate a structured approach in the design and implementation of a software solution
- use a variety of development approaches to generate software solutions and distinguish between these approaches
- use and develop documentation to communicate software solutions to others.

## In this chapter you will learn about:

**Understanding the problem**

- clarification of the specifications
- performance requirements
- identification of inputs and required outputs
- determining the steps that, when carried out, will solve the problem
- Input Process Output (IPO) diagrams

**Abstraction/refinement**

- the top-down approach to solution development
  - a system comprises all the programs in the suite
  - a program comprises all of the modules required to perform the required task
  - a module is a group of subroutines that together achieve a subtask
  - a subroutine is a set of statements that performs a single logical task

**Data types**

- data types used in solutions, including:
  - integer
  - string
  - floating point/real
  - Boolean
- integer representation in binary, decimal and hexadecimal
- characters represented as numbers in binary, decimal and hexadecimal
- limitations of particular data types
- data structures, including:
  - one-dimensional array
  - record
- use of records in sequential files

**Structured algorithms**

- control structures which form the basic building blocks of all algorithms:
  - sequence
  - selection (binary, multiway)
  - repetition (pre-test, post-test), including for ..next loops
  - use of subroutines
- methods for representing algorithms:
  - pseudocode
  - flowcharts incorporating standard control structures
- software structure
  - use of a clear uncluttered mainline and subroutines
  - use of a modular approach
  - use of stubs to represent incomplete modules
- use of standard algorithms, including:
  - load an array and print its contents
  - add the contents of an array of numbers
- checking the algorithm for errors
- benefits of using structured algorithms
  - ease of development
  - ease of understanding
  - ease of modification

# 4

# DEFINING AND UNDERSTANDING THE PROBLEM, AND PLANNING AND DESIGNING SOFTWARE SOLUTIONS

## INTRODUCTION TO SOFTWARE DEVELOPMENT

In the previous chapter, we examined various approaches to software development. We now examine the process of developing software solutions in more detail. Although in a text such as this, it is convenient to examine each stage separately, it is important to remember that this need not be the case when developing actual software products. Software development is often a cyclical process. The requirements of the problem will change over time. During development new ideas surface; the processes and techniques used during development should cater to these needs.

The remainder of this text examines each stage of the software development cycle. This chapter examines defining and understanding the problem, and planning and designing software solutions. Chapter 5 discusses the implementing phase, chapter 6 methods for testing and evaluating software solutions and chapter 7 deals with maintaining existing solutions. The final chapter follows the development of a typical project through each phase of its development.

*Fig 4.1* describes the basic processes occurring during each phase of the software development cycle. From defining and understanding the problem, planning and designing the solution, then coding or implementing the solution. Following implementation the solution is tested and evaluated for correctness and to ensure that it meets the original requirements. Software products are seldom static; rather they are continually modified and upgraded to meet new or varied requirements. The maintenance of software is simplified when the solution is well documented.


**GROUP TASK Discussion**
The phases of the software development cycle are really the logical steps used to solve most problems. Describe how you would build a dog kennel in terms of the phases outlined in *Fig 4.1*.

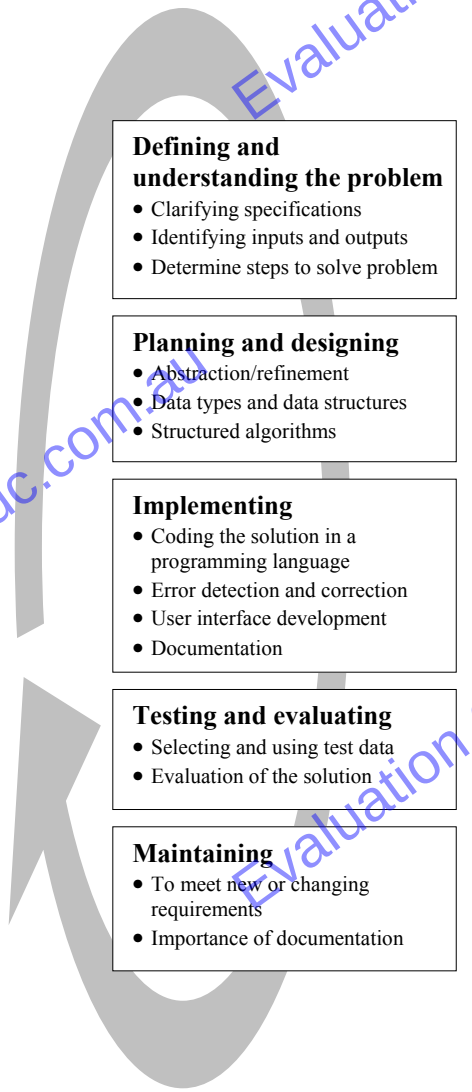



*Fig 4.1*
*Phases of the software development cycle.*

## DEFINING AND UNDERSTANDING THE PROBLEM

Defining the problem precisely is vital to the successful development of any product. An intimate understanding of the nature and requirements of the problem must be realised. Once an understanding of the problem is reached it is possible to consider the inputs into and outputs from the system. The nature of the processing required to transform these inputs into the outputs can then be developed.

There are many other questions and issues that should be addressed during this initial phase. Some common questions follow:

- What needs does the solution hope to meet?
- Is it feasible to develop such a solution?
- What constraints exist?
- What resources are available?
- Does the development team possess the required skills?
- Are there other similar existing solutions?

In the HSC course, we examine many of these questions in more detail.


**GROUP TASK Discussion**

The job of a system analyst is largely about answering many of the above questions. Imagine you are a system analyst. What techniques could you use to assist you in answering questions such as those listed above? Discuss.

### UNDERSTANDING THE PROBLEM

To thoroughly understand the problem involves time and research. It is appropriate to examine similar software solutions as well as non-computer based solutions. Time spent with potential users and existing systems will greatly assist this process. Often users will assume many aspects of a solution are understood because of their familiarity with an existing system. It is the job of software developers to ensure both formal and informal communication with users is maintained. In most cases these communication channels will provide developers with a means of best understanding the problem. Remember, the main advantage of the user-based approach to software development is that the user is the developer and as such, the problem is thoroughly understood. We must try to reach a similar understanding when developing products using other software development approaches.

A list of requirements should be formulated. Such a list can be used to ensure your understanding of the problem will meet the needs of your potential users. These requirements can then be transformed into a more specific set of specifications under which the final product can be evaluated. Often users will communicate their needs in general and non-specific terms e.g. ‘I’d like to be able to print a daily sales summary’. To meet this need requires a deep understanding of what they mean by a sales summary. We must know what data they require, how it should be arranged and summarised, the format of the report and where the source data is stored. Questions in regard to the user interface must be considered. How will the command to print the report be commenced? Is there already a suitable screen that could incorporate the command? Users are often not concerned with such issues until the product is operational. As developers these are important details essential to a complete understanding of the problem.

Consider the following:

You have been asked by a group of your friends to create a website where you can all share your class notes and summaries. Many of you are in the same class for some courses whilst others are in different classes with different teachers. It would be nice if you could have access to the notes from other classes without duplicating those from your own class.


**GROUP TASK Discussion**
In small groups brainstorm a list of needs that could be fulfilled by such a system. Develop these needs into a list of requirements that precisely describe the problem to be solved. Compare and discuss your results with those from other groups.

## IDENTIFICATION OF INPUTS AND REQUIRED OUTPUTS

Understanding the problem will enable a list of required outputs to be created. Outputs are the result of processed input. For example, if a program outputs the number of items purchased in a single week then this implies the required inputs into the process. Each sale during the week is needed, also a range of dates and some input to identify the item to be totalled.

Often inputs will be used by a number of processes resulting in a number of outputs. Some outputs will themselves become the input into other processes. An understanding of the inputs and outputs together with their relationships to each other are vital to the accurate definition of the problem and provide a solid basis for commencing the solution.

The input into a process is known as data and the output as information. The processing occurring transforms the raw data into useful information. There are various modelling techniques available that can be used to describe the data moving through systems. Each technique aims to describe the system in a logical and understandable manner. System flowcharts and data flow diagrams are two such techniques that will be examined in some detail as part of the HSC course.

Context diagrams are level 0 data flow diagrams used to show all the external inputs and outputs to and from a system. Each data item is represented as an arrow or data flow and is labelled with the name of the data item. External entities from which data is obtained (source) or sent (sink) are represented as boxes.

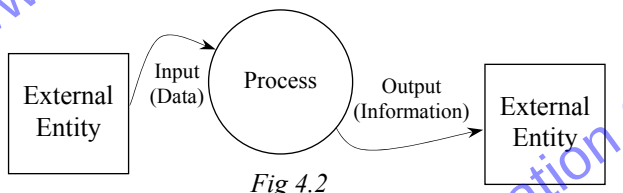



*Fig 4.2*
*A context diagram is used to show the external inputs and outputs into and out of a system.*


**GROUP TASK Activity**
Develop a context diagram to describe the inputs and outputs from an EFTPOS machine. The processing performed by the EFTPOS machine should be considered as the single central process.

## DETERMINING THE STEPS THAT WILL SOLVE A PROBLEM

Once the inputs and outputs have been determined, the nature of the processing required can be considered. Processing transforms the inputs into outputs. The steps required to solve the problem are essentially a description of the processing that when implemented, will fulfil the requirements. For example, making a cup of coffee involves boiling the kettle, putting coffee in the cup, pouring in the boiling water, stirring, then if required adding milk and/or sugar and finally stirring again. The inputs are the boiling water, coffee, milk and sugar and the output is the cup of coffee. The processing transforms the ingredient inputs into the output cup of coffee.

IPO diagrams or charts are useful tools for describing the steps that when carried out will transform inputs into outputs. There are various methods for constructing IPO diagrams. For our present purpose the table format is particularly useful. This format allows a step-by-step description of the processing to be described. The inputs into the process are written beside the step where they are used. Similarly, as outputs are produced by a step they are written to the right. *Fig 4.3* shows an IPO Diagram describing our coffee example. Tabular IPO diagrams are commonly used as documentation for programmers. The programmer uses this information to develop detailed algorithms and to then code the problem in a programming language.

**IPO Diagram**
Making a cup of coffee

| Input | Process | Output |
|---|---|---|
| Water | Boil water in kettle | |
| Coffee | Add coffee to cup | |
| | Pour boiling water in cup | |
| | Stir | |
| Milk | If required add milk to cup | |
| Sugar | If required add sugar to cup | |
| | Stir | Cup of coffee |
| | | |

*Fig 4.3*
*IPO Diagram describing the inputs, processing and outputs required to make a cup of coffee.*

Determining the steps needed to make a cup of coffee is a relatively simple process. For many problems the steps involved will be far from obvious. Even tasks that we are able to perform efficiently and somewhat mechanically can prove difficult to express as a series of unambiguous steps. At this stage, we are interested in determining the general nature of the processing rather than the precise details. Later in this chapter, we examine structured algorithms. Structured algorithms provide a method of describing processing in a detailed and unambiguous manner in preparation for coding in a programming language.

Some techniques that may be useful when determining the steps that will solve a problem include:

- Consider a smaller version of the problem.
- Examine solutions to similar or related problems.
- Work through an example by hand using pen and paper.
- Break down the problem into smaller more manageable pieces.
- Brainstorm possibilities with others.

Consider the following:

We all know how to add a list of numbers, however can we describe the steps involved in a way that could be understood by someone who had no knowledge of addition? Let us attempt this task.

For this scenario the inputs and output are obvious. The numbers are the inputs and the sum is the output. There are also various other things we need to know. We must know the basic addition number facts e.g. 4+5=9, 6+7=13, etc… We need an understanding of place value; in decimal we have a units column, then to the left a tens column, hundreds, thousands, etc… For our solution let us assume this knowledge is understood.

To commence we could consider adding two 3-digit numbers. Say 456 + 789. As we perform the addition on paper we write down each step.

1. Add units column.
2. Write unit part of answer below units column.
3. If result has a tens column component write this above the tens column.
4. Add up the tens column including the carry.
5. Write units part of answer below the tens column.
6. If result has a tens component write it above the hundreds column.
7. Add up the hundreds column including the carry.
8. Write answer below hundreds column.

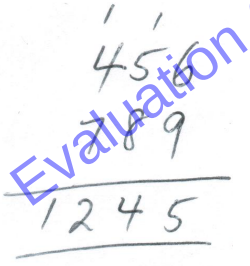



*Fig 4.4*
*Consider a smaller example.*

We notice that much of what we do is repeated. If our steps are to work for very large numbers, then it would be better if we could write a more general description where a block of steps is repeated until we reach the left hand column containing digits. What about lists of more than two numbers? How can we alter the steps to account for this possibility?


**GROUP TASK Activity**
Rewrite the above steps in such a way that they will work for two numbers of any length. Will your steps work if there are more than two numbers? Alter your steps to ensure this is the case.


**GROUP TASK Activity**
Draw up an IPO diagram like the one in *Fig 4.3* for your addition problem. Swap IPO diagrams with other class members and try to perform an addition using only the steps on their chart. Discuss your results and refine your IPO diagram accordingly.

## PLANNING AND DESIGNING SOFTWARE SOLUTIONS

Thorough planning and design of software solutions is essential to the development of quality software products. Planning the solution commences by considering the complete problem. This general overview is then systematically broken down into smaller pieces that include increasing amounts of detail. The nature of the data required is determined and structures to hold this data are designed. Eventually a method of solving each of these component pieces of the solution is designed in the form of an algorithm. Together these data structures and algorithms combine to solve the larger problem.

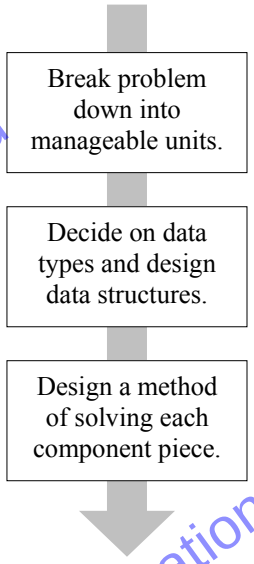



*Fig 4.5*
*Overview of the planning and design phase.*

There are numerous techniques and tools that are available to assist developers during the planning and design phase. Techniques for representing the hierarchy of modules that make up the system and methods for describing algorithms help developers solve the problem. They also provide important documentation for other team members and future maintenance personnel. Computer Aided Software Engineering (CASE) tools are software applications used by developers to streamline a multitude of development tasks - from modelling the system, designing data structures, to creating algorithms, assisting with code generation and finally testing the solution.

In this section we examine firstly the abstraction/refinement process where the problem is broken down into a series of solvable smaller problems. We then examine data types; how they are represented within the computer together with their limitations. Some simple data structures are considered which allow more efficient access to the data within programs. Finally we consider the design of algorithms using flowcharts and pseudocode. Remember that an algorithm is a method of solving a problem using a series of unambiguous steps.


**GROUP TASK Discussion**
List any software tools with which you are familiar that could be classified as CASE tools. Briefly describe the function of each tool.

### ABSTRACTION/REFINEMENT

Software development needs to be a precise and detailed process. However, most problems encountered by developers are too large for the entire solution to be comprehended with sufficient precision and detail. To overcome this situation, we refine the problem by breaking it down into smaller, more manageable modules. This is the process of top-down design. Each of these modules can then be considered as an isolated problem. We can largely ignore the big picture and consider the detail of the smaller piece. This is the process of abstraction. We separate part of the problem and solve it in isolation. Once all component parts have been solved they are combined to solve the larger problem.

**Abstraction**
Taking away or separating part of the solution so it may be considered in isolation.

There are many advantages of the abstraction process. It encourages developers to create reusable modules of code. For example, a module that sorts a list of data can be used in the future as part of the solution to a variety of different problems. Testing is greatly simplified. As each module is small and self-contained it can be thoroughly tested before being included in the total solution. As a consequence, checking the final total solution is greatly simplified. Teams of developers are able to work on individual modules in the knowledge that the work completed by others will not affect their own.

**The top-down approach to solution development**

Top-down design is the most common method of breaking a problem into smaller units. The problem is progressively refined until each unit can be successfully implemented as a subroutine of programming code. In many cases, the design will be strictly hierarchical; each higher-level subroutine accessing or calling one or more lower-level subroutines. The process of developing this hierarchy of subroutines is known as stepwise refinement. Each level or step is refined into a series of lower-level subroutines.

**Top-down design**
Progressively breaking a larger problem into a series of smaller easier to solve problems.

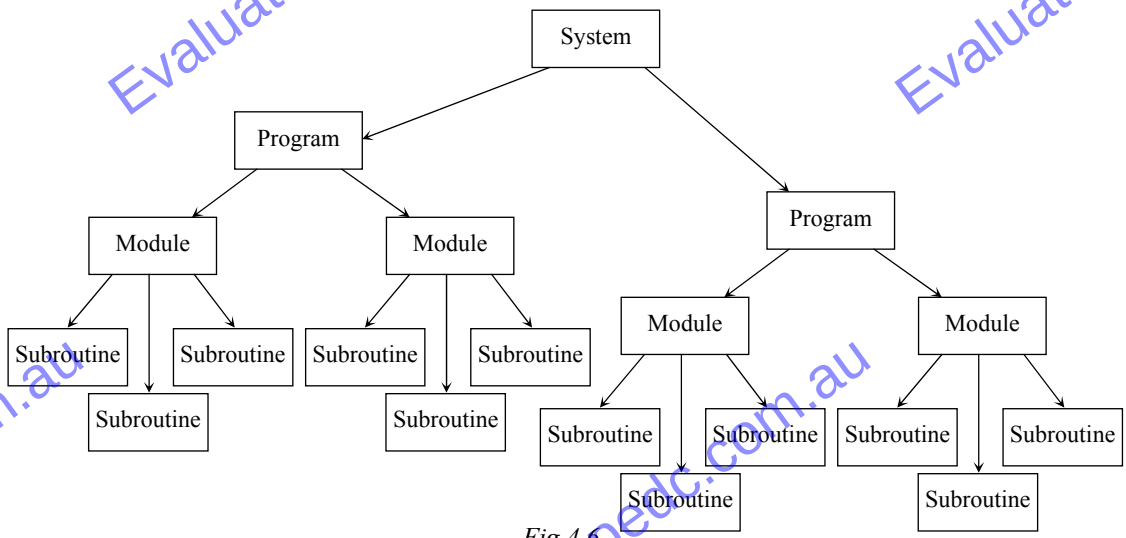



*Fig 4.6*
*Software systems are composed of programs, which are composed of modules, which are composed of subroutines.*

Systems are also composed of a hierarchy of software elements (refer *Fig 4.6*). The system includes a number of programs. Each program is implemented as a number of modules. And each module is a group of related subroutines which together perform related tasks. Subroutines are at the lowest level. Each subroutine implements a single logical task. By combining related subroutines into modules, modules into programs and programs into systems promotes reusability. Particular modules or even programs can be reused as part of many software solutions.

Most programming languages include or provide access to a variety of different libraries; these libraries are modules which perform common tasks. For example, a module (or library of code) for drawing graphics contains subroutines for drawing different shapes including different colours, line weights and fill styles. There is no need for software developers to "reinvent the wheel", rather they include and utilise these existing modules within their own programs.

Hierarchy charts are a simple method used to represent the top-down design of a particular problem. Subroutines are often numbered to indicate their placement in the hierarchy (refer *Fig 4.7*). Structure diagrams are similar to hierarchy charts but enable the inclusion of more detailed information in regard to the order and nature of processing. Data flow diagrams (DFDs) also model the top-down design but focus on the movement of data rather than the detail of the processing performed.

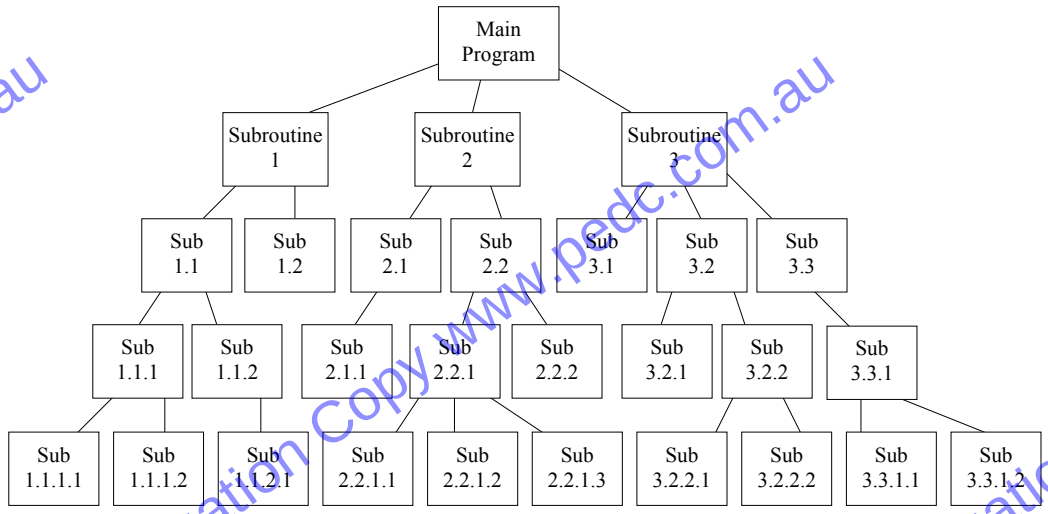
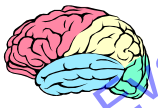



*Fig 4.7*
*Hierarchy charts describe top-down designs.*

Be aware that top-down design does not necessarily lead to the development of reusable modules of code. Often the design of subroutines reflects the particular needs of the current problem. In the past software was often developed in isolation and hence modules and subroutines where specific to the current problem. Today there is an emphasis on developing more general and therefore reusable solutions. There has also been a shift in thinking towards the use of externally developed modules or libraries of code. The use of these generic modules is similar to the way most other products are developed. For example, the majority of components used in a new model of car are derived from generic components. The motor, air conditioning, tyres, brakes and instruments are sourced from outside. These same components are often used with minor alterations on a variety of vehicles.

Top-down design focuses on the processing required to solve a problem. We commence with a set of inputs and move through a series of predetermined steps that process these inputs into outputs. For many problems this should certainly be the focus. For other problems the flow of data, the links between data, the maintainability of the code or response times are of a higher priority. Be aware that there are other different approaches to that of top-down design.



Consider the following:

Imagine you've decided to cook a special meal for your family. Let us develop a top-down design to describe the processes involved in this venture.

Firstly, you must decide on the time and the date of the meal. You must then work out the menu and obtain the ingredients. On the allocated day you must cook and serve the meal. *Fig 4.8* describes our solution so far.

We can now consider in isolation each subroutine on level 1 of our hierarchy chart. As we consider each subroutine we ignore the larger problem – this is the process of abstraction. Deciding on the time and date involves questioning each family member and then picking the most appropriate time and date. We are refining the problem using stepwise refinement. Working out the menu includes considering the tastes of those who'll attend and then examining recipe books. Obtaining the ingredients may include working out the quantities required, checking the pantry and then visiting various shops. Cooking and serving the meal includes preparing the ingredients, cooking them and finally serving to each family member.

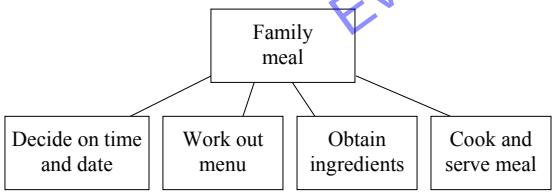



*Fig 4.8*
*Initial top-down design for preparing our family meal.*

Let us expand the preparation of the ingredients branch of our hierarchy chart. This may include rinsing the vegetables, chopping the ingredients, measuring quantities and mixing ingredients. Chopping may further involve first peeling vegetables and perhaps grating and dicing.

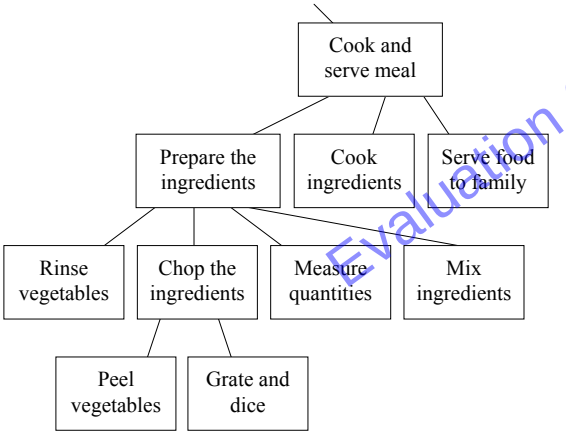



*Fig 4.9*
*Preparing the ingredients refined and expanded.*


**GROUP TASK Activity**
Expand each branch of the initial hierarchy chart shown in *Fig 4.8*. Add a numbering system to your completed chart.

Consider the following:

- A large multi-national software company specialises in the development and marketing of software to the retail industry. One of their largest selling software products is able to interface with various types of cash register and EFTPOS terminals. Currently there is no standard in place for these communication links. As a consequence, the company must develop a new software interface each time new models of cash register or EFTPOS terminal become available.
- A web site developer works from home. He charges an hourly rate to design and develop web sites for various clients. At present he has been in business for about 2 years and is finding that past clients are increasingly requesting modifications to their original sites. Implementing these modifications is proving difficult and time consuming.
- The Year 2000 bug was due to programs only considering and storing the last two digits of the year e.g. 1931 was stored as just 31. The time spent examining software products to ensure Year 2000 compliance was considerable. Perhaps the most annoying aspect of the problem was that most products were found to be compliant without the need for any modification.


**GROUP TASK Discussion**
For each scenario above, discuss how top-down design, abstraction and reusable modules simplify the process of modifying the software.


**GROUP TASK Discussion**
In many cases it is just simpler to throw out the old product and develop a new one from scratch. Do you agree with this statement? Explain and discuss your answer using the above scenarios as examples.

## SYSTEMS MODELLING TOOLS

A model of a system is a representation of that system designed to show the structure and functionality of the system. Diagrams are particularly useful methods of modelling because they are able to give a broad view whilst at the same time conveying necessary detail. Accomplishing the same task in words is difficult. In terms of software development, a model can be thought of as a plan which specifies the design of the software. The model gives direction and specifications to the builders of the product, in the same way as the plan for a house gives builders of the house direction and specification in regard to the house's design and erection.

Different types of modelling are applicable to different aspects of the system. System flowcharts are used to represent the logic and movement of data between the system's components, including hardware, software and manual components. Dataflow diagrams describe the flow of data to and from processes and storage elements. Structure diagrams describe the top-down design and sequence of processing. IPO diagrams explain how inputs are transformed into outputs by processing. Data dictionaries describe the nature and type of the data used in a program. Screen designs and concept prototypes are used to determine user requirements by simulating the final product from the user's perspective. Most projects will use a combination of modelling techniques. We considered IPO diagrams in the previous section, in this section we examine systems flowcharts, data flow diagrams (and context diagrams) and structure charts.

### Systems Flowcharts

Systems flowcharts are used to describe the logic and flow of data through a system. They describe the interactions that occur between input, processing, output and storage, as well as the nature of each of these components. Even manual processes can be included on systems flowcharts. Generally, systems flowcharts are used at a higher level than the other modelling techniques to show an overall view of the entire system.

Although some of the symbols used on system flowcharts are the same as in algorithm flowcharts, the design of system flowcharts is quite different. Flowcharts for algorithms are designed to precisely show the logic of an algorithm, whereas system flowcharts are not describing an algorithm but a system. Logic is only shown where necessary to sensibly describe the flow between components.

Systems flowcharts have been in common use since the 1960s, hence some of the symbols, such as the punched card symbol, are now outdated. *Fig 4.10* details the systems flowchart symbols specified for Software Design and Development. There are many other symbols that can be used for specialised processes such as sorting and merging data. In this course we use the process rectangle for all processing.

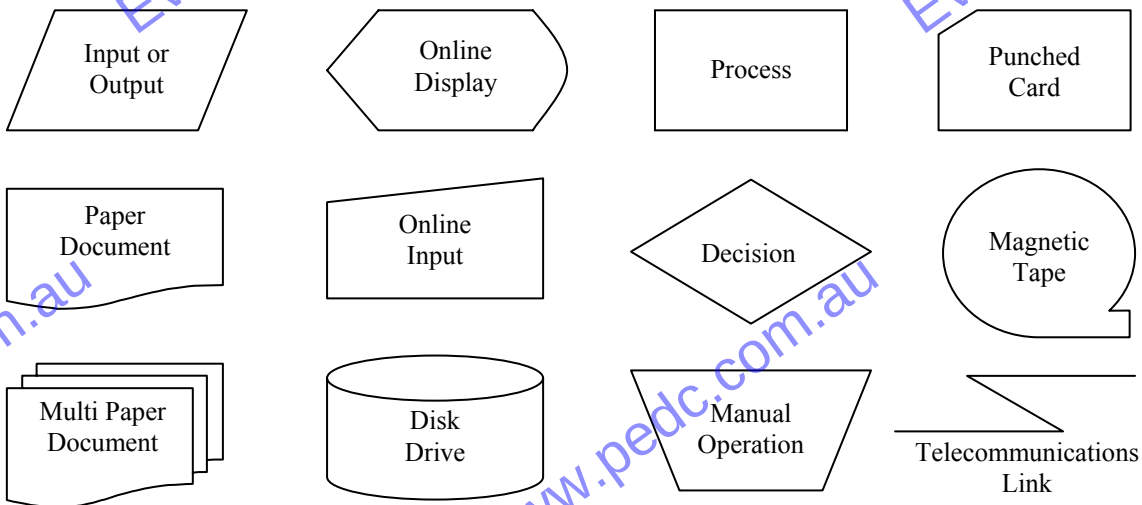



*Fig 4.10*
*Components of System Flowcharts.*

Consider the following:

The systems flowchart at right describes the logic and flow of data for results from an assessment task.

Firstly, the teacher marks the test, which is a manual operation. These marks are entered by hand into the teacher's mark book. The marks are entered into the computer, which is an online input, and stored in the school database. At the same time, the student names are being retrieved from the school database. Once all the marks have been entered they are scaled and stored in the school database. Finally, a printout of the results is generated and students are given their results.

This systems flowchart describes the path taken by the marks through the system, from their generation, when the teacher marks the papers, through to the final scaled marks being delivered to the students. The logic is also described to make sense of the flow of the data. Notice that the logic is from an overall system viewpoint rather than just the logic required to code the software part of the solution. This systems flowchart is not significantly different to one that would be used to describe a completely manual system. That is, the software is just one item within the total system.

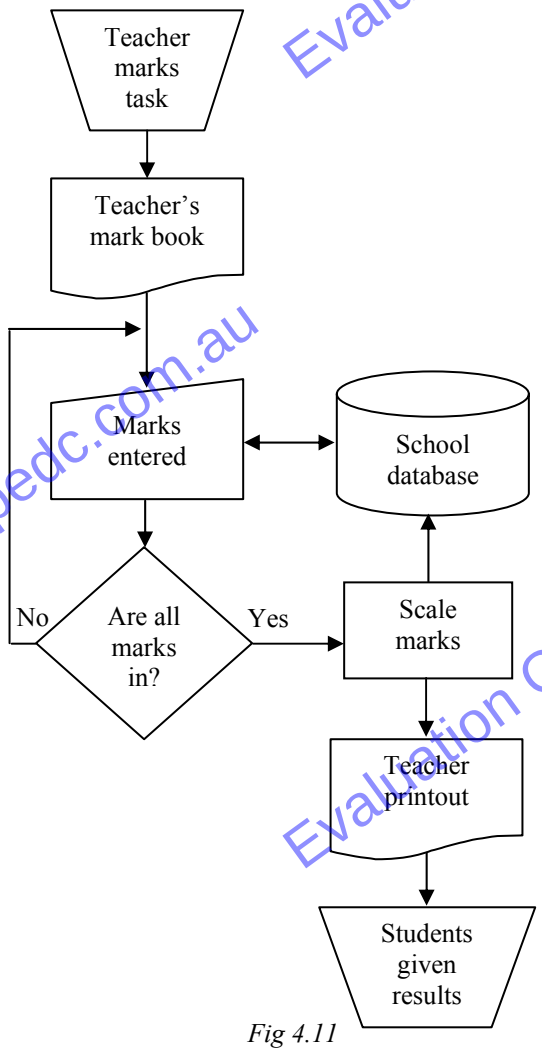



*Fig 4.11*
*System flowchart for results from an assessment task.*


**GROUP TASK Describe**

The above system flowchart uses a number of symbols as part of its construction. Describe in words what is occurring at each symbol on the diagram.

Consider the following:

The systems flowchart below describes a hotel's information system. This model is designed to describe the logic and flow of data through the hotel system for an individual guest.

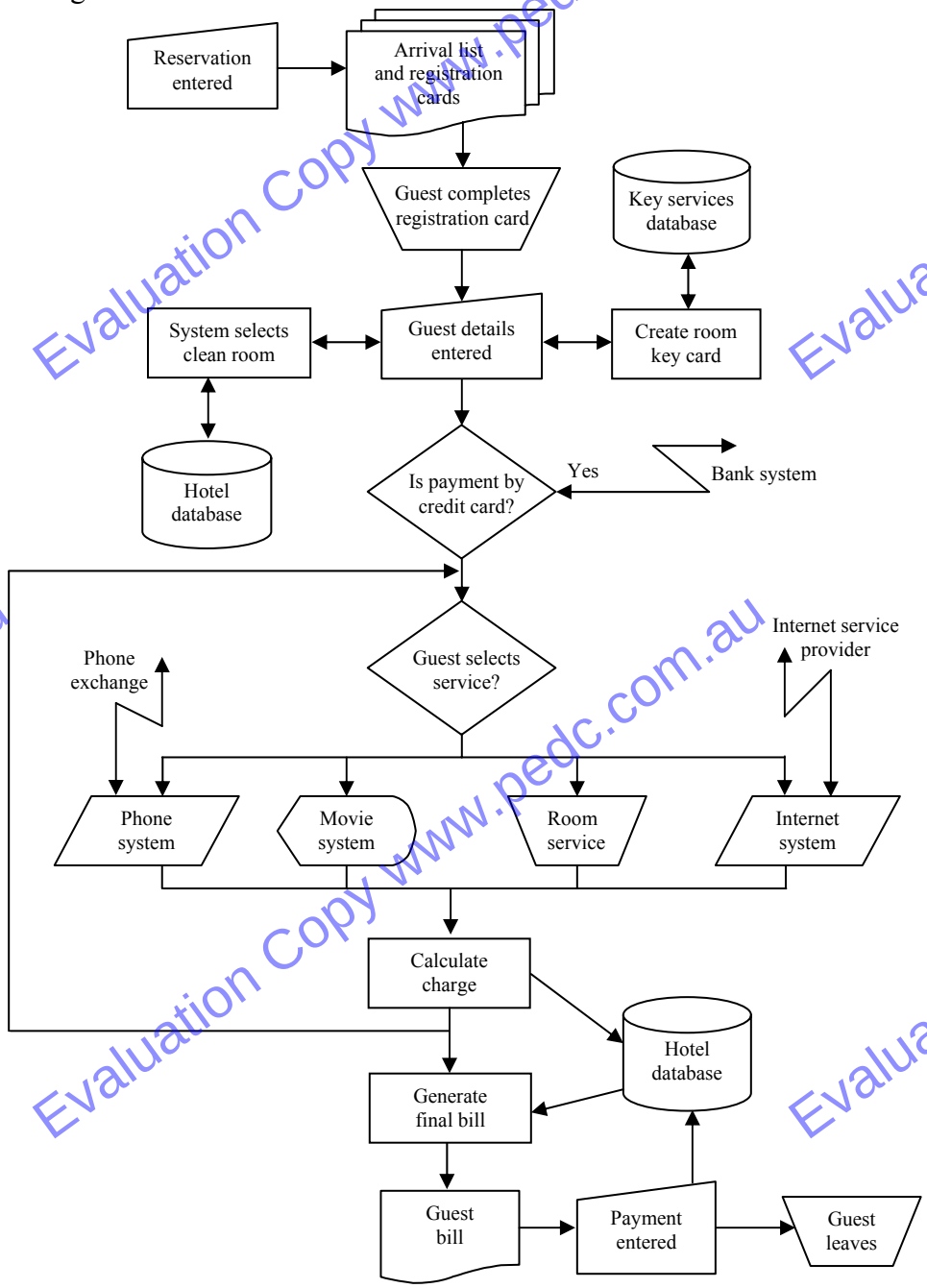



*Fig 4.12*
*System flowchart describing the flow of data for an individual guest in a hotel.*

The above systems flowchart describes the information transfers occurring during a guest's stay in a hotel. A reservation is entered, and then at the start of each day an arrival list and registration cards are printed. When the guest arrives, he or she completes their registration card and the details are entered into the system. A room and a key are allocated to the guest. If payment is to be made by credit card then the bank's system is contacted to ensure there are sufficient available funds. The guest then is able to use a number of services whilst staying, namely the phone, movie or Internet systems or room service. Charges for these services are recorded in the hotel's database. The phone and Internet systems involve telecommunication links to outside systems. At checkout time, the final bill is generated from data stored in the hotel's database and presented to the guest. Finally, the bill is paid and the guest leaves.


**GROUP TASK Discussion**
Name any items on the above systems flowchart that could or should have links to the hotel database but do not. Explain your answer.


**GROUP TASK Discussion**
Systems flowcharts are designed to display an overall view of a system. Specific detail about each process is not required, also the logic need not be as precise as that required for an algorithm. How useful is the above hotel's system flowchart for a software developer working on an upgrade to the hotel's database? Describe any information of use to the software developer provided by this model.

**Data Flow Diagrams (DFDs)**

Data flow diagrams describe the path data takes through a system. No attempt is made to indicate the timing of events. Think of a DFD as a railroad map: it shows where the train tracks are laid, but it does not give the timetables.

Let us now consider the syntax of DFDs: Data flows or vectors are used to join the entities and processes within the model. They can be thought of as pipelines or interfaces for data to flow between the other components of the DFD. A label is placed on the vector or arrow to indicate the nature of the data. The label used should be unique within a given DFD. The origin of data input into the system is known as a source. A destination for data output from the system is known as a sink. Both sources and sinks are external entities and are designated using a square or rectangle symbol. In some references, external entities are also known as terminators as they describe starting and ending points for the DFD. External entities are often groups of people or other systems. Processes are the actions that take place on the data within the system to transform inputs into outputs. Processes are represented on data flow diagrams as circles. All processes in a data flow diagram will have one or more data flows coming in (inputs) and one or more data flows going out (outputs). Data stores are repositories for data. They usually represent databases or files, however manual storage such as filing cabinets or paper files may also be included. Data stores are represented using open rectangles.

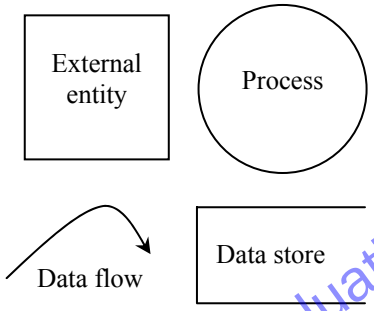



*Fig 4.13*
*Symbols used on data flow diagrams.*

Data flow diagrams are used both as a tool for system analysis and as a tool for system design. Modelling an existing system using a data flow diagram assists in understanding the flow of information through a system and also helps the analyst to separate processing into distinct units resulting in a better understanding of the problem. Data flow diagrams are particularly useful modelling tools during system design. Computer aided software engineering (CASE) tools are available to assist in the creation of data flow diagrams. Many of these tools include data dictionary functions that link to the data flows on the diagrams.

Usually, a context diagram is drawn first. A context diagram shows the entire system as a single process with all the external inputs and outputs to and from any external entities. Context diagrams are also called Level 0 data flow diagrams. Data stores should not be shown on context diagrams. A series of data flow diagrams are produced from the context diagram that progressively break the problem down into lower-level modules. This top-down design technique is continued until the lower-level processes are sufficiently detailed for them to be easily coded.

Consider the following:

The data flow diagram in *Fig 4.14* describes the student administration system at a local college.

The college receives enrolment forms from students. The data on these forms is used to create class lists, which are sent to the staff. Courses taken by students are recorded in the student file. Staff set test papers, which are used to generate results. The results are stored in the student file. Academic records are retrieved from the student file and processed. Graduating students receive a transcript of their degree.

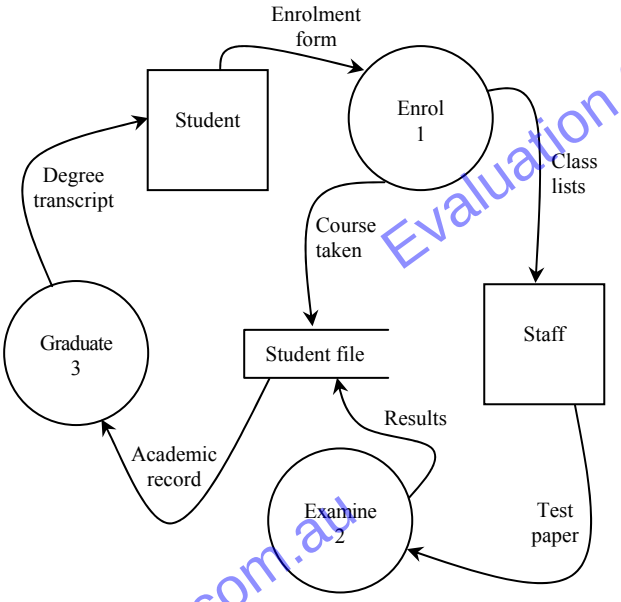



*Fig 4.14*
*Level 1 data flow diagram describing student administration at a college.*

Each of the processes on the level 1 DFD in *Fig 4.14* can be further refined into a series of level 2 DFDs. In *Fig 4.15* the Examine 2 process has been expanded to form a level 2 DFD. Notice the numbering system for the processes; 2.1, 2.2, etc. The processes on the level 3 DFD for the Print exams 2.1 process would be numbered 2.1.1, 2.1.2, 2.1.3, etc. The inputs and outputs to and from a process must be identical at all times. In this Examine 2 level 2 DFD example, Test paper is the input and Results is the output, just as they were on the original level 1 DFD in *Fig 4.14*.

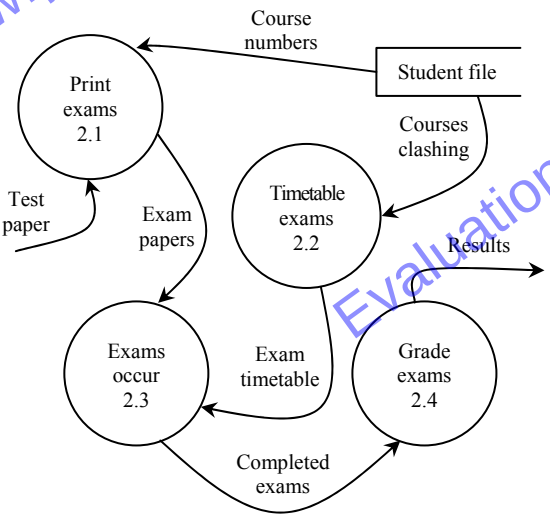



*Fig 4.15*
*Level 2 DFD for the Examine 2 process.*

Pages 163-173 not included in this sample chapter


**GROUP TASK Activity**
Scientific calculators are dedicated computers. What is the range of values that can be represented in scientific notation on your scientific calculator? What is the smallest positive number possible? Write down some calculations beyond the accuracy or range of your calculator.


**GROUP TASK Activity**
Investigate the data type used to represent numbers within a spreadsheet application. Create various formulas to help determine the accuracy and other limitations of the data type used. Determine the number of significant figures that can be relied upon.

**String**

String data types are used to store text data. For example, 'cat', '123abc', 'y<%2g' or 'Once upon a time'. Most programming languages provide predefined string data types. Often a fixed length string and a dynamic length string are provided. The fixed length string can hold up to a set number of characters whereas dynamic strings grow in size as required. Other languages require programmers to define strings themselves using sequences of characters; the character data type being predefined.

Regardless of the available data types, strings are ultimately stored as separate characters. Each character is coded using a standard system. ASCII, EBCDIC, ANSI and Unicode are some common examples. Many of these coding systems use a single byte or less, to represent each character. The Unicode system uses up to 32 bits and is designed to include all the possible characters and marks used in all the languages of the world. The Unicode system incorporates the more widely understood ASCII system. Regardless of the coding system used, each character is stored separately as a unique pattern of bits.


**GROUP TASK Activity**
Examine the character based data types available in a database program with which you are familiar. What are the size limitations for each of these data types? What happens if an entry exceeds these limits?

Let us consider the ASCII system in some detail. ASCII is an acronym for American Standard Code for Information Interchange. Each character is represented using 7 bits. In binary, 7 bits provides 128 different combinations hence there are 128 characters in the standard ASCII character set (see *Fig 4.27* below).

**ASCII**
American Standard Code for Information Interchange. A coding system for characters using 7 bits. Most other coding systems incorporate ASCII.

The ASCII characters are grouped and ordered to facilitate the sorting and searching process for programmers. The 10 digits are in order and come before the uppercase alphabet, which comes before the lowercase alphabet. There are relationships between upper and lower case characters. For example, uppercase 'A' has an ASCII code of 65, which in binary is 1000001. Lowercase 'a' occupies ASCII 97 which is 1100001 in binary. In fact all the lowercase characters are exactly 32 more than their uppercase partner. Sorting without regard to case and converting between cases is simplified.

| Char | Dec | Description |
|---|---|---|
| NUL | 0 | Null character |
| SOH | 1 | Start of header |
| STX | 2 | Start of text |
| ETX | 3 | End of text |
| EOT | 4 | End of transmission |
| ENQ | 5 | Enquiry |
| ACK | 6 | Acknowledge |
| BEL | 7 | Bell |
| BS | 8 | Backspace |
| HT | 9 | Horizontal tab |
| LF | 10 | Line Feed |
| VT | 11 | Vertical tab |
| FF | 12 | Form Feed |
| CR | 13 | Carriage Return |
| SO | 14 | Shift Out |
| SI | 15 | Shift In |
| DLE | 16 | Data link escape |
| DC1 | 17 | Flow control |
| DC2 | 18 | Flow control |
| DC3 | 19 | Flow control |
| DC4 | 20 | Device control 4 |
| NAK | 21 | Neg. acknowledge |
| SYN | 22 | Synchronous idle |
| ETB | 23 | End trans. block |
| CAN | 24 | Cancel |
| EM | 25 | End of medium |
| SUB | 26 | Substitute |
| ESC | 27 | Escape |
| FS | 28 | File separator |
| GS | 29 | Group separator |
| RS | 30 | Record separator |
| US | 31 | Unit separator |
| SP | 32 | Space |
| ! | 33 | Exclamation mark |
| " | 34 | Quotation mark |
| # | 35 | Hash sign |
| $ | 36 | Dollar sign |
| % | 37 | Percent sign |
| & | 38 | Ampersand |
| ' | 39 | Closing single |
| ( | 40 | Left parentheses |
| ) | 41 | Right parentheses |
| * | 42 | Asterisk |

| Char | Dec | Description |
|---|---|---|
| + | 43 | Plus |
| , | 44 | Comma |
| - | 45 | Hyphen |
| . | 46 | Period |
| / | 47 | Forward slash |
| 0 | 48 | |
| 1 | 49 | |
| 2 | 50 | |
| 3 | 51 | |
| 4 | 52 | |
| 5 | 53 | |
| 6 | 54 | |
| 7 | 55 | |
| 8 | 56 | |
| 9 | 57 | |
| : | 58 | Colon |
| ; | 59 | Semicolon |
| < | 60 | Less than |
| = | 61 | Equals sign |
| > | 62 | Greater than |
| ? | 63 | Question |
| @ | 64 | At-sign |
| A | 65 | |
| B | 66 | |
| C | 67 | |
| D | 68 | |
| E | 69 | |
| F | 70 | |
| G | 71 | |
| H | 72 | |
| I | 73 | |
| J | 74 | |
| K | 75 | |
| L | 76 | |
| M | 77 | |
| N | 78 | |
| O | 79 | |
| P | 80 | |
| Q | 81 | |
| R | 82 | |
| S | 83 | |
| T | 84 | |
| U | 85 | |

| Char | Dec | Description |
|---|---|---|
| V | 86 | |
| W | 87 | |
| X | 88 | |
| Y | 89 | |
| Z | 90 | |
| [ | 91 | |
| \ | 92 | Backslash |
| ] | 93 | |
| ^ | 94 | Caret |
| _ | 95 | Underscore |
| ` | 96 | Left quote |
| a | 97 | |
| b | 98 | |
| c | 99 | |
| d | 100 | |
| e | 101 | |
| f | 102 | |
| g | 103 | |
| h | 104 | |
| i | 105 | |
| j | 106 | |
| k | 107 | |
| l | 108 | |
| m | 109 | |
| n | 110 | |
| o | 111 | |
| p | 112 | |
| q | 113 | |
| r | 114 | |
| s | 115 | |
| t | 116 | |
| u | 117 | |
| v | 118 | |
| w | 119 | |
| x | 120 | |
| y | 121 | |
| z | 122 | |
| { | 123 | |
| \| | 124 | Vertical line |
| } | 125 | |
| ~ | 126 | Tilde |
| DEL | 127 | Delete |

*Fig 4.27*
*The standard ASCII character set. Control characters use codes 0 through to 31. Printable characters use the remaining codes from 32 to 127.*

Consider the following:

The string of characters “KONK!” is entered into a computer. Assuming the eighth bit of each byte is always zero the string is stored in binary as 01001011 01001111 01001110 01001011 00100001. Similarly the string “konk!” is stored as 01101011 01101111 01101110 01101011 00100001.


**GROUP TASK Activity**
Convert each byte into decimal and check your result against those shown on the ASCII table in *Fig 4.27*.


**GROUP TASK Discussion**
Examine the above two sets of binary numbers. How could ASCII assist programmers to convert between upper and lowercase? Discuss using KONK! as an example.

### Boolean

The Boolean data type is used to store logical data; the only possible data items being either true or false (or yes or no). The term Boolean is derived from Boolean algebra which was developed by mathematician George Boole. Boolean algebra is used to describe logical propositions where the result must always be either true or false. Other common names used by programming languages rather than Boolean include logical, bit or simply Yes/No. In all cases, only two possible states are possible.

Each Boolean data item requires a single bit of storage, either a binary 1 or 0. It is standard practice to use 0 to mean false and 1 to mean true. In reality, many programming languages allow you to assign Boolean variables any value, however any non-zeros are treated as 1s and hence represent the Boolean value true.

The Boolean data type is commonly used to store any data that can only have two possible states. Examples include, Male or female, eligible or ineligible and on or off. Checkboxes are screen elements used to gather and display Boolean data. Boolean flags are variables often used within programs to signal that a section of code has or has not been executed.


**GROUP TASK Discussion**
The processes occurring within the CPU are all based on logical decisions. These decisions result in the output of sequences of 1s and 0s, therefore all data is ultimately Boolean data. Do you agree? Discuss.


**GROUP TASK Discussion**
Although Boolean data is often thought of as true or false it can be used to represent other data that has just two states. Make a list of at least 10 instances where the Boolean data type would be appropriate.

Consider the following:

The following data are often used and stored by software solutions:

- Phone numbers
- Gender
- Postcodes
- Date of birth
- Time of day
- Email addresses
- Total number of items
- Cost of products
- X, Y coordinates on a screen
- Average of many integers


**GROUP TASK Discussion**
Recommend the most suitable data type for each of the above data elements. Justify your answers.


HSC style question:

(a) Convert the binary number 11001010 into hexadecimal and decimal.

(b) Compare and contrast the binary representation of string data with the binary representation of integer data.

(c) In regard to examinations:

- They are written by teachers and printed
- Students sit the exams to answer questions
- Teachers mark the exams and the results are given to students

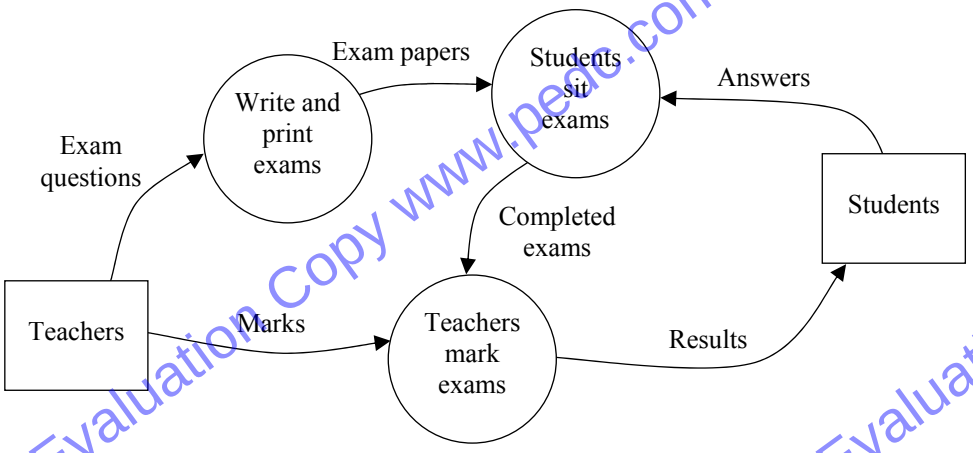
Draw a dataflow diagram to describe the above system using each of the above dot points as individual processes.

**Suggested solutions**

(a) $1100_2 = 8 + 4 = 12 = C$. $1010_2 = 8 + 2 = 10 = A$. Therefore, $11001010_2 = CA_{16}$. $11001010_2 = 128 + 64 + 8 + 2 = 202_{10}$. Or, $CA_{16} = (12 * 16) + 10 = 202_{10}$.

(b) String data is represented as a sequence of individual characters. A particular character is represented using the same binary code each time it occurs within the text. Integer data, on the other hand, represents complete numbers using a single representation. For example each digit in the numeric value 123 is NOT represented individually, rather the entire number is represented as a binary number using the twos complement system. Integer data is represented this way so it can be used to perform mathematical operations.

(c)



**SET 4B**

1. Boolean, Date, Integer and Floating Point are examples of:
   (A) data items.
   (B) data types.
   (C) strings.
   (D) instructions.

2. If a stored result may be of a fractional value, then which data type should be assigned to the applicable variable?
   (A) Floating point.
   (B) Integer.
   (C) Boolean.
   (D) String.

3. Today, all computer data and instructions are ultimately stored in:
   (A) ASCII.
   (B) hexadecimal.
   (C) binary
   (D) decimal.

4. The decimal equivalent of the binary number 10001 is:
   (A) 9.
   (B) 17.
   (C) 24.
   (D) 65.

5. Which is the most accurate data type for storing real numbers?
   (A) double precision floating point
   (B) single precision floating point
   (C) Boolean
   (D) 64 bit integers

6. Which data type is most commonly used to store data that can only have two possible states?
   (A) Integer.
   (B) Floating point.
   (C) String.
   (D) Boolean.

7. A value that is assigned outside of the allowable range will most likely result in:
   (A) a type mismatch error.
   (B) a null value.
   (C) the correct result but represented as a negative value.
   (D) no error occurring.

8. The number 27B is probably an example of:
   (A) a hexadecimal number.
   (B) a binary number.
   (C) an octal number.
   (D) a decimal number.

9. The most common coding system for characters that uses 7 bits is known as:
   (A) ANSI.
   (B) EBCDIC.
   (C) ASCII.
   (D) Unicode.

10. Whole numbers are also known as:
    (A) real numbers.
    (B) floating point numbers.
    (C) integers.
    (D) none of the above.

11. Convert each of the following binary numbers into their decimal and hexadecimal equivalents.

    (a) 01010101 (b) 11110111 (c) 10111010 (d) 01111110

12. Choose the best data type for each of the following. Justify your answers.

    (a) The gender of clients in a database.
    (b) The average of a set of exam results.
    (c) The number of items remaining on the shelf.
    (d) A doctor's notes on patients.

13. Convert the word "Gouldian" into binary ASCII. (set the first bit of each byte to 0)

14. The following hexadecimal numbers represent ASCII characters.

    49 20 6C 6F 76 65 20 41 53 43 49 49

    What does it say?

15. Calculations involving money must be precise. Research how this is achieved when money is often a fractional quantity.

## DATA STRUCTURES

Data structures are arrangements of data items; the aim being to simplify access and processing of the data. When designing data structures for particular problems it is important to carefully consider the nature of the processing that will be required. Well-designed data structures can significantly reduce the complexity of the processing required. Conversely poor or inappropriate data structures can greatly increase the processing required and reduce the software's performance.

In the preliminary course we introduce three data structures: arrays, records and files. These three structures can be combined in various ways to suit the data and processing requirements of individual problems.

### One-dimensional array

Arrays are used to store multiple data items where each data item is of the same data type. Each element within the array has a unique index that is used to access the data contained within the element. Structures similar to arrays are used extensively in non-computing applications. For example, post office boxes (PO boxes). PO boxes are numbered sequentially within each post office. These numbers are used as indexes when locating a particular PO box. Each PO box is a storage container for mail. The mail (data items) will change within a particular storage container but the container and its index remain constant. The ordered and sequential nature of the PO box numbers together with the boxes physical arrangement simplifies the sorting and distribution of mail. In the case of PO boxes there is a single index, the PO box number. In the HSC course we examine arrays that use multiple indexes, called multi-dimensional arrays. At this stage we restrict our discussion to arrays with single indexes, one-dimensional arrays.

Let us consider a typical array data structure. In *Fig 4.28* the array element with an index equal to 2 contains the data item 'Cow'. Notice that each data element in *Fig 4.28* is of the same data type (in this case strings) and that there is an obvious relationship between them (they are all animal names). If we were to call our example array Animals then the array element with an index of 2 is specified as Animals(2). At present Animals(2) contains the data item 'Cow'. Similarly Animals(1) contains 'Frog' and Animals(4) contains 'Cat'.

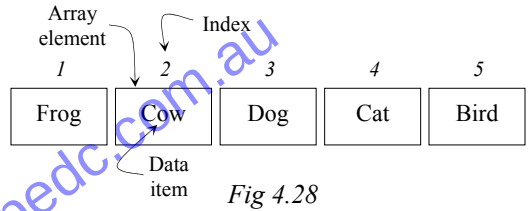



*Fig 4.28*
*Each array element has a unique index.*

As arrays contain a distinct countable number of elements it makes sense to use whole numbers (integers) as the index to arrays. In some programming languages the range of the index may be altered to best suit the needs of the problem. For example, the index may range from 0 to 9 resulting in ten array elements or it could range from -2 to 7 also resulting in ten array elements. For simplicity it is common to commence indexing at either zero or one. The index is often called a subscript because of its similarity to subscripts used in mathematics. The term dimension is also used synonymously with the term index.

**Index**
An integer value used to denote a particular data item held in an array. Often called a dimension or subscript.

So far we have accessed the contents of array elements using a constant value for the index e.g. 2 or 3. The power of arrays is realised when we allow the index to change resulting in access to a sequence of array elements. This is accomplished by using a

variable in place of the constant index value. For example, consider again the Animals array in *Fig 4.28*. If the variable Count ranges in value from 1 to 5 during processing then Animals(Count) would access each of the five animal names in sequence as shown in *Fig 4.29*. Our code is generalised to process each element in the array.

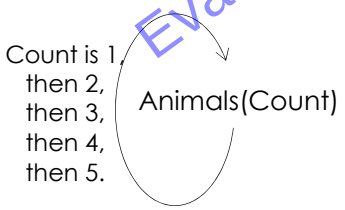



*Fig 4.29*
*Allowing the index to change allows efficient access to multiple array elements.*

Consider the following:

A modification is required to the airline check-in system to increase security as a consequence of the September 11, 2001 attacks on the USA. Currently passengers show identification to obtain their seat allocation and boarding pass. No identification is required as they board the aircraft. This makes it possible for passengers to exchange boarding passes and sit in different seats and even on different aircraft to those for which they hold a ticket.

The modification to the system involves flight personnel at the boarding gate entering the seat number into a computer terminal. The terminal responds by displaying the passenger's name. The staff member must then check the name against that on the passenger's passport and ensure the passport photo is actually the passenger.

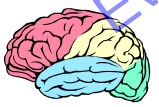
The software developer assigned this task decides to use an array which is declared with the identifier Seating. The index for each element of the Seating array will be the seat number and the data items stored in the array will be the passenger names. At the check-in desk, passengers are assigned seats. This process sets the array element corresponding to the required seat to the passenger's name. For example, Seating(23)="Nerk Fred" means that Fred Nerk will be sitting in seat number 23 on the aircraft. At the boarding gate the flight personnel enter the seat number from the boarding pass. The system responds by displaying the passenger's name. If the seat is empty then no name is returned. The passenger name displayed is compared to the passenger's passport and the passport photo compared to the person to confirm their identity.

```
Seating(1) = "Broermann Kim"
Seating(2) = ""
Seating(3) = "Bradley Marlene"
Seating(4) = "Bradley John"
Seating(5) = ""
Seating(6) = ""
Seating(7) = "Eagle Nicholas"
Seating(8) = "Carrucan Lindsay"
Seating(9) = "Fendall Janine"
Seating(10) = ""
Seating(11) = "Davis Matthew"
Seating(12) = "Davis Kareena"
Seating(13) = ""
Seating(14) = "Fendall Brent"
Seating(15) = "Fendall Richard"
```

*Fig 4.30*
*The Seating array for a 15 seat aircraft after most passengers have checked in.*


**GROUP TASK Discussion**
How does the above modification assist in improving security? Use possible examples to assist in your explanation.


**GROUP TASK Discussion**
The index for the Seating array is itself meaningful data. Often this is not the case. What attributes make the seat numbers appropriate for use as the array's index? Discuss.



Consider the following:

1. 10,000 names need to be searched to find any duplicates.
2. Calculating the average of a set of test results.
3. Storing the personal details for an individual employee.
4. Storing the daily rainfall occurring over a number of years.
5. Inputting a list of numbers and outputting them in reverse order.
6. Preparing a family tree.


**GROUP TASK Discussion**
In which of the above scenarios would an array be of assistance? In each case explain your reasons. If an array is appropriate describe its structure.

### Record

A data structure containing data items that are related but not necessarily of the same data type is called a record. Records are used extensively for database applications. Generally individual records contain all the information about a particular entity (or individual) within the database. For example, an address book contains details of individual people. Each person is an entity within the address book and has their own record. The record may contain the person's surname, Christian name, date of birth, gender, street address, suburb, postcode, phone number, mobile number and email address. Each of the data items within each record is known as a field.

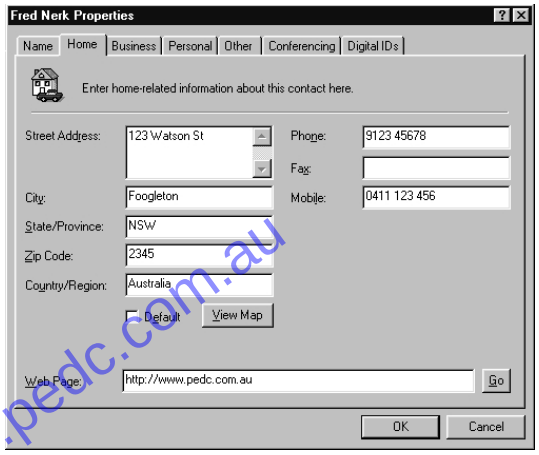


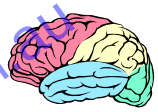

*Fig 4.31*
*Microsoft Outlook includes an address book where an individual's details are stored as a record.*

Each record of the same data type is made up of the same fields. The data items held within these fields will most likely be different but the data type of like fields must be the same. For example, the surname field always holds strings whereas the gender field always holds Boolean data.


**GROUP TASK Discussion**
Records are used extensively as integral parts of many software applications. List and describe some applications that use records.


**GROUP TASK Discussion**
Records are different in many ways to arrays. Describe these differences. Use examples to reinforce your response.

Before a record structure can be used as a data type, its fields must be named and each assigned a data type. The record structure then becomes a user-defined data type that can be used in the same way as the predefined included data types. We consider the programming language statements used to perform these tasks in more detail in Chapter 5. Once the structure of the record has been described we can create individual records of that data type. These records can be manipulated as single data items or each of their component fields can be manipulated individually.



Consider the following:

As part of the development of a computer game the programmer has determined the need for a record data structure to hold information on each player and the level they have achieved during play. Each level contains a number of screens. The records are stored on disk and are examined at the start of a game so players may recommence play at their current level and screen.

Each record contains three fields. Name, Level and Screen. The record structure is assigned the data type PlayerRecord. *Fig 4.32* shows the code required to accomplish this task in Visual Basic. A record with the data type PlayerRecord is declared using the identifier Player. This record can be processed as a complete unit or each of its component fields can be accessed separately.

```
Structure PlayerRecord
   Name As String
   Level As Integer
   Screen As Integer
End Structure
```

*Fig 4.32*
*Creating the user defined data type called PlayerRecord.*

Within the program the disk file is accessed and each record is read in turn into Player. The Name field is examined in search of a match with the current player's name, e.g. does Player.Name = EnteredName. If they do match then the entire record is retained for future use. Assuming that CurrentPlayer is also of type PlayerRecord, the statement CurrentPlayer = Player would achieve this aim. This statement copies the entire contents of Player into CurrentPlayer. If we just wanted to copy the current level then the statement would be CurrentPlayer.Level = Player.Level. A period '.' is the standard notation used to indicate what follows is a field within the record e.g. Player.Level means that Level is a field within the Player record.

```
Player.Name = "Freddo"      Player.Name = "Makka"
Player.Level = 5            Player.Level = 3
Player.Screen = 2           Player.Screen = 4
```

*Fig 4.33*
*Example data that could be held in Player.*


**GROUP TASK Activity**
Create a list of 10 possible records that could be stored in a record of data type PlayerRecord.


**GROUP TASK Discussion**
Why is a record structure a better solution than an array for the above scenario? Could one or more arrays have been used? Discuss.

### Sequential files

Files are used to permanently store data on secondary storage devices. Data can be written to a file or read from a file. Different types of access to files suit different requirements. Sequential files can only be accessed from start to finish. That is, to read or write to the middle of a sequential file requires accessing all the data prior to that position in the file. It is not possible to jump directly to a particular data item.

The operating system controls access to files. As a consequence, software applications must work via the operating system when working with files. To commence using a file the operating system needs to know the location and name of the file together with information in regard to the method of access. Once work on the file is completed the application notifies the operating system and the operating system closes the file.

A sequential file can be likened to an audio-cassette. To play the fifth song on the cassette requires fast forwarding through the first four songs. You cannot record a song between the fourth and fifth songs without overwriting existing music. If you wish to record over the fifth song you must be sure the new one is of identical size to the old. If it is not then you risk either overwriting part of song six or having the end of the old song five left on the tape. However you can easily add (or append) a new song to the end of the tape without affecting the existing songs. Sequential files operate in a similar manner.

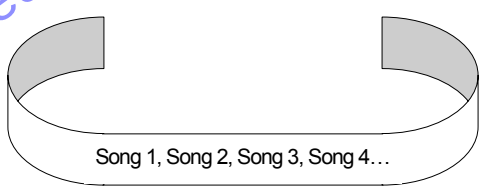



*Fig 4.34*
*Data on audio-cassettes are stored sequentially.*

Sequential files are continuous streams of data (usually characters). The structure of the data is not coded as part of the file. If the data within the file has some structure then the software must understand this structure; the file itself does not contain this information. Many languages contain statements that allow data to be written and read a single character at a time, a data item at a time, record at a time or a line at a time. Separators are inserted by the programming language to separate data items e.g. commas or tabs. Carriage returns and/or line feed characters are added after each line.

Most programming languages include an end of file (EOF) function, which returns the Boolean value True when the end of a file has been reached. This function can be used to ensure errors do not occur due to programs attempting to read past the last character. Programmers often add sentinel values within files to indicate the end of a sequence of data items e.g. "ZZZ" or 9999999.

Consider the following:

It is possible to open most files sequentially. Most word processors allow you to do this by stipulating that the file is a text file. Unfortunately the result often appears to be gibberish. *Fig 4.35* shows part of a jpeg image file opened as a text file in Microsoft Word.

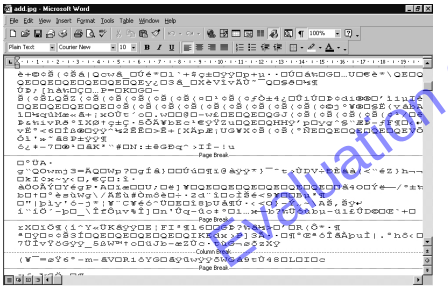
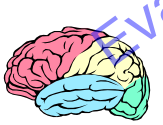


*Fig 4.35*
*Part of a jpeg file opened as a text file.*


**GROUP TASK Discussion**
Using your knowledge of sequential files and ASCII text, explain why the file in *Fig 4.24* appears as gibberish. Discuss.



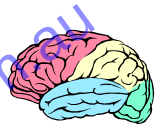
Consider the following:

As part of a program a sequential file is used. The requirements of a particular module mean that new data needs to be inserted at the start of the file. To perform this process the program performs the following steps:

1. Create a new file.
2. Add the new data to this new file.
3. Open the existing file.
4. Read a character from the existing file.
5. Write this character to the new file.
6. Repeat the above two steps until the end of the existing file is reached.
7. Close both files.
8. Delete the original existing file.
9. Rename the new file to the name of the old file.

**GROUP TASK Discussion**
From the user's point of view it would seem that data could be added to the front of the sequential file. Explain why the user may think this is the case.

**GROUP TASK Activity**
Work through the above steps on paper using some sample data. In your own words explain how these steps achieve their purpose.



Consider the following:

An existing sequential file contains the text for a Shakespeare essay you are working on in English. A friend notices that you have incorrectly spelt Shakespeare as Shakespear throughout the essay. Being a keen Software Design and Development student, you decide to correct the problem using your knowledge of sequential file techniques.

**GROUP TASK Discussion**
Develop a series of steps that could be used to accomplish the above task. Remember you can only add or append data to the end of a sequential file.

**GROUP TASK Activity**
Create, on paper, a sample file containing the misspelt word a number of times. Work through your steps developed above to ensure each of the errors has been corrected.

## DATA DICTIONARY

When developing a software solution to a problem, it is vital that all variable names or identifiers are carefully documented. This ensures that all members of the development team are aware of identifiers that have been used, their data type and the scope of usage. A data dictionary is the repository where this information is held. Data dictionary functions are available with most software development CASE tools. If the data dictionary is an integral part of the development environment then confusion due to duplicate identifiers can be eliminated. For smaller development projects, the data dictionary is often maintained on paper with the assistance of a database or word processor.

**Scope**
The extent to which a variable is available for use. Global variables are available for use by all subroutines. Local variables are available to an individual subroutine.

A data dictionary should include a thorough description of each variable used by the program and each field in each database and file used or accessed by the program. A separate data dictionary is created for each module, database and file used by the program. Commonly a data dictionary will be a table containing columns for:

- Identifier name (or field name)
- Data type (including the data structure if applicable)
- Length (Number of characters or decimal places if applicable)
- Scope of the variable
- Purpose or description

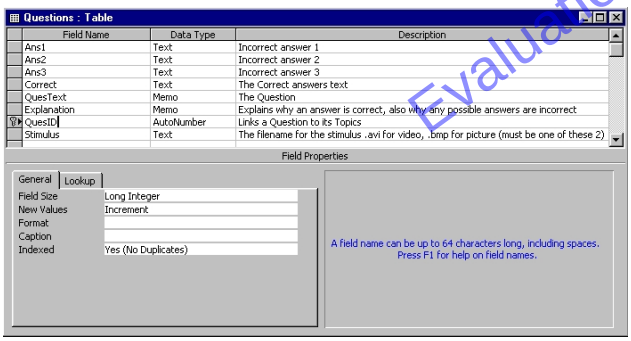



*Fig 4.36*
*A data dictionary from MS-Access, a popular relational database system.*

The exact columns used will be determined by the nature of the data used in the application. The example data dictionary at right (*Fig 4.36*), displays a sub-form to allow entry of information specific to the particular data type of the current field. This data dictionary describes a table in a relational database. It includes an icon of a key to indicate the primary key field in the table.

The screen shown in *Fig 4.37* is part of a data flow diagram CASE tool. This screen allows the modification of the attributes of each data item used in the data flow diagram. As data flows are added to the diagrams the data items are automatically added to the data dictionary. Clicking on the 'Edit Data Item' button opens a screen allowing entry of details describing specific data items, such as data type, range, and a description of the data item's purpose.

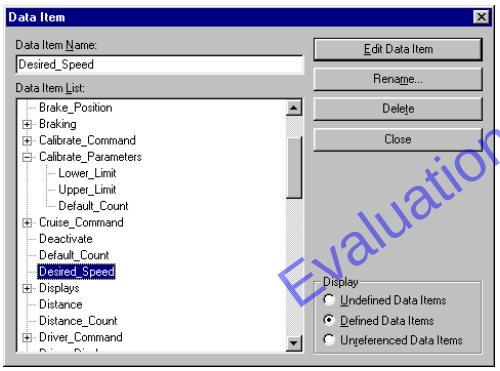
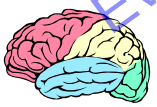



*Fig 4.37*
*Data dictionary screen from the Axiomsys CASE tool.*



Consider the following:

The data dictionary below describes all the identifiers used in the Convert Amount to Words module from the Invoicing System structure diagram shown in *Fig 4.19*.

| Name | Data Type | Length | Scope | Purpose |
|---|---|---|---|---|
| AmountInWords | String | 255 char | Function Name Global | Returns the currency amount in words. |
| Amount | Numeric | Real (2 dec. pl.) | Local | Input parameter. |
| TempDigit | Numeric | Integer | Local | Stores each digit as it is extracted from Amount. |
| DigitWord(19) | Array of strings | 10 char | Local | The word associated with each digit, e.g. DigitWord(5)=”five”. |
| TenPowerWord(9) | Array of strings | 10 char | Local | Word for each power of ten, e.g. TenPowerWord(3)=”thirty” |
| Ten3Word(4) | Array of strings | 10 char | Local | Word for each 3rd power of ten, e.g. Ten3Word(2)=”million” |
| PlaceCounter | Numeric | Integer | Local | Counter incremented for each digit in Amount. |
| TempResult | String | 255 char | Local | Stores the amount in words during processing. |

*Fig 4.38*
*Data dictionary for the ‘Convert Amount to Words’ module.*

The programmer created this data dictionary whilst developing the source code. Data dictionaries are stored, along with other documentation, to assist in future maintenance and upgrading of the software product.


**GROUP TASK Discussion**
How could a data dictionary, such as the one above, be of assistance to future maintenance personnel?


**GROUP TASK Discussion**
Explain the relationship between structure diagrams, IPO charts and data dictionaries. Use the above data dictionary as an example to illustrate your answer.

## SET 4C

1. A record can best be described as:
   (A) a data structure containing unrelated data items.
   (B) a data structure containing related data items all of the same data type.
   (C) a data structure containing related data items but not necessarily of the same data type.
   (D) a data structure containing unrelated data items but all of the same data type.

2. A dimension or subscript is another name for:
   (A) an index.
   (B) a data item.
   (C) an array.
   (D) a record.

3. What is often added within a file to indicate the end of a sequence of data items?
   (A) EOF function.
   (B) Sentinel values.
   (C) Boolean values.
   (D) A carriage return.

4. A one-dimensional array has how many indexes?
   (A) One.
   (B) Two.
   (C) Multiple.
   (D) None.

5. Each data item within a record is individually known as:
   (A) an array.
   (B) a file.
   (C) an element.
   (D) a field.

6. Arrays, records and files are examples of:
   (A) data indexes.
   (B) data types.
   (C) data items.
   (D) data structures.

7. Adam is developing a product and has decided to use a record structure in one of the modules. What must Adam do first to ensure it can be used as a user defined data type?
   (A) He must name every field and assign each a data type.
   (B) He does not need to name them, just assign each field a data type.
   (C) He must name them but he does not need to assign each a data type.
   (D) He cannot create a user defined data type.

8. A file that can only be accessed from start to finish is a:
   (A) serial file.
   (B) sequential file.
   (C) random access file.
   (D) direct access file.

9. Arrays are used to:
   (A) store multiple items of the same data type.
   (B) store multiple items of differing data types.
   (C) store a single data item.
   (D) none of the above.

10. File access is controlled by:
    (A) the browser.
    (B) the application software.
    (C) the operating system.
    (D) the FP Unit.

11. Select a data structure for each of the following. Justify your responses.

    (a) Gathering the responses to a questionnaire.
    (b) Storing the output from a word processor.
    (c) A set of exam marks.
    (d) Details of products sold in a shop.

12. Describe the essential differences between an array and a record.

13. Arrays can greatly simplify access to multiple data items. Do you agree? Explain your response.

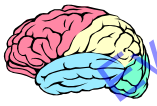
14. Sequential files can be used to store records and arrays, however there are access restrictions. Describe these restrictions.

15. Create an IPO diagram to describe how a data item can be inserted into the middle of a sequential file.

## STRUCTURED ALGORITHMS

An algorithm is a method of solving a problem. The algorithm describes the processing steps necessary to transform the inputs into the outputs. This occurs within a finite amount of time. Algorithms are used to assist in the solution of all types of problems not just computer-based problems. For example, a recipe is an algorithm describing the preparation and cooking steps required to create a meal. In most cases, recipe books call this the 'method'. The method is a sequence of steps that transform the ingredients (input) into the meal (output).

We use algorithms subconsciously. For example, each morning you go through a sequence of decisions and steps to prepare and travel to school. When searching for a lost article we try various techniques; we may try retracing our steps, looking in obvious places, asking other family members, etc… These are all valid methods of solving the problem; in essence you are using algorithms. The algorithm is not the solution but the method used to arrive at the solution. This chapter is about bringing algorithms out of the subconscious and presenting them in such a way that others can understand them. To do this requires us to be able to clearly and logically describe algorithms we devise. There are various methods of algorithm description, in this course we consider two, pseudocode and flowcharts.



Consider the following:

Each morning most high school students need to make decisions about their day. The steps taking place may include the following:

1. Wake up
2. If you are sick then stay in bed.
3. If it is a school day then continue otherwise stay in bed.
4. Get out of bed.
5. Have a shower.
6. Get dressed in school uniform.
7. Eat breakfast.
8. Pack school bag.
9. If mum is home then scab a lift otherwise catch bus to school.

There are problems with the above steps if we work through them in the precise order indicated. If you are sick, then step 2 directs you to stay in bed. We then reach step three, which indicates that on school days we must complete the steps that follow. This seems to contradict step 2's direction. This algorithm has three exit points, at steps 2, 3 and 9. It would be better if we could restructure our algorithm to have a single exit. Structured algorithms require a single exit point and, as we shall see, this is always possible.


**GROUP TASK Activity**

Can you rewrite the steps 1 to 9 from above in such a way that there is only one exit point from the algorithm? Ensure that your new algorithm performs the tasks in the manner intended in the original.

## METHODS FOR REPRESENTING ALGORITHMS

There are many standard methods for representing algorithms. Using standard methods of algorithm description assists in the process of developing algorithms and allows others to understand the algorithms created. In this course, we examine two methods; pseudocode and flowcharts. By adhering to the rules of pseudocode and flowcharts the process of coding is simplified.

Pseudocode is an English-like algorithm description language. Keywords are used in pairs to indicate the start and finish of each control structure. These keywords are written in capitals to make them stand out. Between each pair of keywords, statements are indented to further emphasise each control structure. Flowcharts use rectangles for processes, parallelograms for input and output and diamonds for decisions. These components are connected with flow lines. Although it is possible to draw flow lines in any direction there are standard constructs for each of the standard control structures used for structured algorithms. When using flowcharts we must be careful to use only recognised control structures. It is possible to draw flowcharts that cannot be implemented in programming code.

```
BEGIN MAINPROGRAM
  Get Number
  Set Count to 1
  WHILE Number > Count
    Set Product to Number × Count
    Display Product
    Increment Count
  ENDWHILE
END MAINPROGRAM
```

*Fig 4.39*
*Sample algorithm using pseudocode.*

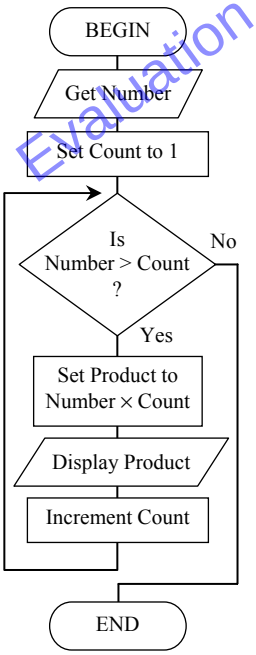



*Fig 4.40*
*Sample algorithm expressed using a flowchart.*

Pseudocode and flowcharts are used to represent exactly the same thing. For example, *Fig 4.39* and *Fig 4.40* represent exactly the same algorithms. Which one is used, is purely personal preference. Some would argue that pseudocode is easier as it is closer to programming code. Others feel flowcharts are more visual and hence easier to construct and understand. In this course, it is necessary to be comfortable writing and reading both pseudocode and flowcharts.

The wording of each process used is not part of the algorithm description method. For some algorithms, maths symbols may be appropriate, for others English phrases are more suitable. For example, `Count = 1` or `Set Count to 1`. It is common practice to be consistent within each algorithm. The aim is to write algorithms that can be easily understood and are logically correct.


**GROUP TASK Discussion**
Compare the pseudocode and flowchart shown in *Fig 4.39* and *Fig 4.40*. Can you see how both algorithms are describing the same method of solution? Discuss.


**GROUP TASK Discussion**
What problem do you think is being solved by the algorithms in *Fig 4.39* and *Fig 4.40*? Describe, in words, the operation of the algorithms that lead to the solution of this problem.

## CONTROL STRUCTURES

Structured algorithms are built using control structures. This process is known as structured programming. Control structures determine the direction or order in which statements within an algorithm are executed. Control is the influence that directs the flow of execution. Control structures are standard constructs used when creating structured algorithms.

The theory behind structured programming is that all problems can be solved using just three different control structures; sequence, selection and iteration. This theory has never been definitively proven however no problem has ever been found that cannot be solved using structured techniques. All imperative and procedural programming languages include statements allowing the implementation of each of these control structures as part of software solutions. Structured programming techniques are used to create the majority of software developed in the world today.

Although only the three control structures; sequence, selection and iteration are required, a fourth, the use of subroutines or subprograms, is desirable when using top-down design development techniques. Let us examine each of these control structures in detail:

### Sequence

In structured programming the order of processing is important. Each process must be performed in the correct order for the solution to be realised. Sequence is the control structure that ensures each process occurs in the correct order. For example, when getting dressed you must put your socks on before your shoes.

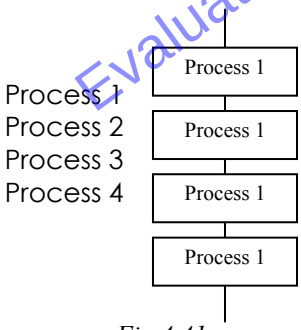



*Fig 4.41*
*Sequence expressed in pseudocode and as a flowchart.*

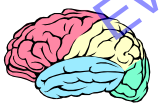
There are programming languages where the sequence of processing is not significant. Consider a spreadsheet; formulas are entered into cells with little need to consider the order in which these formulas will be evaluated. In Prolog a set of facts and rules are interrogated to reach conclusions. The order of events is not a significant consideration.

When using pseudocode each process is written one under the other. Similarly on flowcharts the order of processing moves from top to bottom. Each process is completed before the next is commenced. A single process that is out of order will most likely effect the operation of the entire algorithm.

Consider the following:

To back a car out of a garage requires the following steps: start the car, open the garage door, get in the car, release the hand brake, engage reverse gear, press accelerator, check review mirror, unlock car and place foot on brake.


**GROUP TASK Activity**
The steps above are obviously not in the correct sequence. Rewrite these steps in the correct sequence first as a flowchart and then in pseudocode. Is there only one possible sequence? Discuss.



Consider the following:

Many software solutions require that the contents of two variables need to be swapped. A precise sequence of events must be adhered to if this process is to occur correctly. The four flowchart segments below are attempting to swap the data item in Num1 with the data item in Num2.

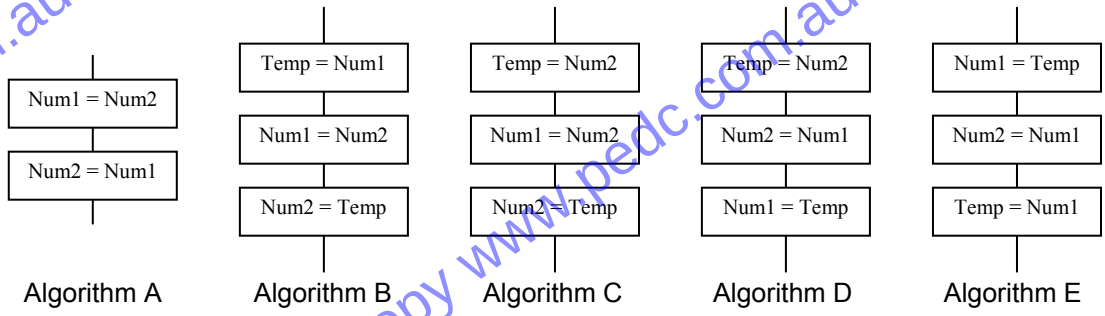



*Fig 4.42*
*Possible swap algorithms.*

**GROUP TASK Discussion**
Some of the algorithms above correctly perform the swap while others do not. Identify the correct and incorrect algorithms. Explain your response for each algorithm.

**GROUP TASK Discussion**
The sequence of steps is vital to the success of the swap algorithm. Describe situations where sequence is not so vital to an algorithm's success. Discuss.

### Selection

Selection is the control structure that allows decisions to be made between different alternative paths. Different paths are executed in response to the outcome of some condition. Once the processes along this path are complete processing continues with the statement following the selection control structure.

- **Binary selection**

In binary selection there are two alternatives, one being selected if the condition is true and the other if the condition is false. The condition results in a Boolean value, either true or false. Often one branch will not involve any additional processes, rather its purpose is to avoid execution of the processes present on the alternate branch. *Fig 4.43* shows the syntax used to represent binary selection using pseudocode and flowcharts.

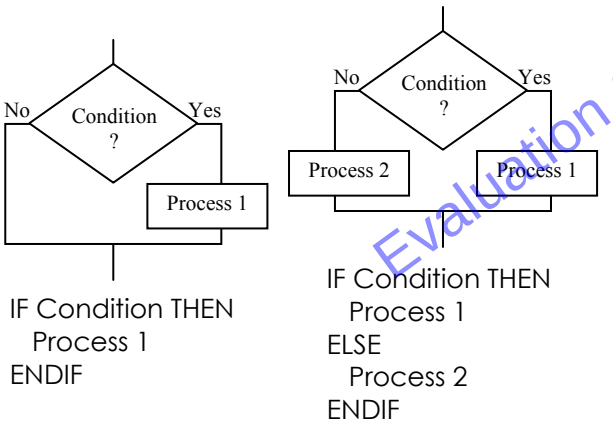



*Fig 4.43*
*Binary selection using flowcharts and pseudocode.*

When using binary selection, the condition can be expressed as a statement e.g. Number > 2 in which case the result is either True or False. It can also be expressed as a question e.g. Is Number > 2? in which case the answer is either Yes or No. In either case, it is important on flowcharts to label each branch appropriately using Yes/No or True/False. In this text we prefer to use questions combined with Yes and No.

Consider the following:

Under current Australian law you must be 18 years of age to vote. An algorithm to determine if someone can vote is described in *Fig 4.44* as a flowchart and in pseudocode.

```
Get Age
IF Age ≥ 18 THEN
  You can vote
ELSE
  You can't vote
ENDIF
```

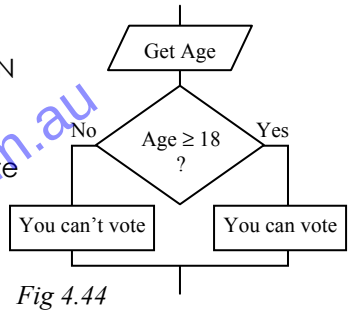



*Fig 4.44*
*Algorithm to decide if someone can vote.*

**GROUP TASK Activity**
There are many different ways of writing the above algorithm so it correctly achieves its purpose. Describe using both pseudocode and a flowchart some alternative algorithms.

**GROUP TASK Discussion**
The flowchart shown in *Fig 4.44* uses the three symbols. Describe the purpose of each symbol in terms of making flowcharts more understandable. Discuss.

Consider the following:

To ride on the roller coaster at an amusement park you must weigh at least 40 kilograms and be taller than 135cm. *Fig 4.45* describes two different algorithms. The first uses pseudocode and the second, a flowchart.

```
Get Weight, Height
IF Weight ≥ 40 THEN
  IF Height > 135 THEN
    You can ride
  ELSE
    You can't ride
  ENDIF
ELSE
  You can't ride
ENDIF
```

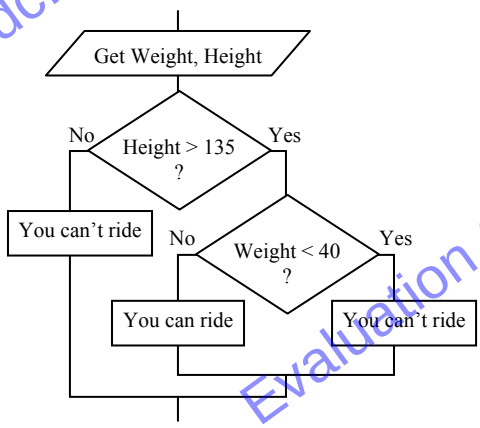



*Fig 4.45*
*Two different algorithms for the roller coaster problem.*

**GROUP TASK Activity**
Rewrite the first algorithm as an equivalent flowchart. Rewrite the second algorithm in its equivalent pseudocode. Which solution do you think is the best? Discuss your answer.

Pages 193-202 not included in this sample chapter

```
BEGIN MAINPROGRAM
  Get First
  Get Second
  Result = Biggest (First, Second)
  Display Result
END MAINPROGRAM

BEGIN Biggest(Item1,Item2)
  IF Item1 > Item2 THEN
    Big = Item1
  ELSE
     Big = Item2
  ENDIF
  RETURN Big
END Biggest
```

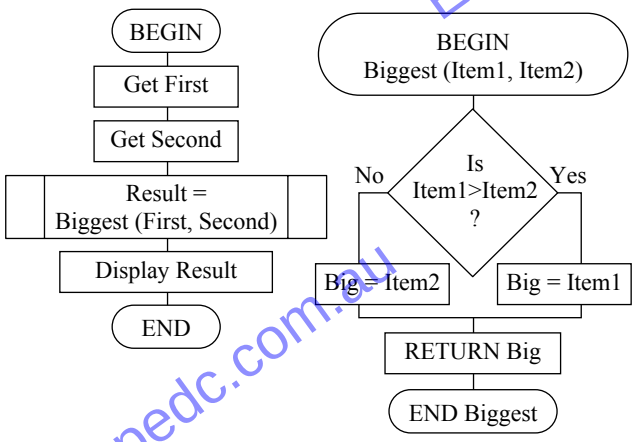



*Fig 4.63*
*Using RETURN to pass values back in pseudocode and flowcharts.*

Consider the following:

Imagine you work part time at the local library. Often you have the task of locating books for the librarian. The librarian gives you a list containing the call numbers of each required book. You then go to the shelves and retrieve these books.

This is similar to the way a call where parameters are passed by reference works. You are playing the part of the subprogram and the librarian is the main program. The shelves are memory areas containing the data (the books). The call numbers represent the actual locations of the books; the actual parameters. There is only a single book with a unique call number. You receive the list of call numbers and use them as formal parameters. Whether you or the librarian find a particular book it remains the same book.

Now imagine the librarian wishes you to comment on a newspaper article. She makes a photocopy of the original article and hands it to you. You then take the photocopy to your desk and as you read you highlight significant words and ideas. You then write down your comments on a piece of paper, hand them to the librarian and toss the photocopy of the article in the bin.

This is like passing by value. Again the librarian is playing the part of the main program and you are playing the subroutine role. The article is passed by value meaning a copy of the data is passed rather than a link to the original. Clearly your highlighting will not appear on the original newspaper article as it is merely a copy of the original. Also, tossing the photocopy in the bin has no effect on the original. The return value is your comments which you (the subroutine) hand back to the librarian (the main program).


**GROUP TASK Activity**
Act out the above library scenario. Pause as the drama unfolds to discuss how each component and process compares and assists your understanding of calling subroutines with parameters.


**GROUP TASK Discussion**
The use of parameters is central to the programming process. Most commands, functions and even operators included within programming languages use parameters. Discuss using examples.

## STANDARD ALGORITHMS

Often similar problems are encountered as part of the solution to many problems. Rather than continually reinvent the wheel, there are many standard algorithms that are used in these circumstances. In life we commonly use standard algorithms. For example, in primary school you learnt how to perform long multiplication using a specific method or algorithm. Imagine if your primary school teacher insisted that you discover and create an algorithm for multiplication on your own. Perhaps eventually you would work it out, however it is unlikely your algorithm would be as easy to use as the one you now use. No doubt you have added modifications to the originally taught multiplication algorithm. Perhaps you just add a zero when multiplying by ten or you halve and add a zero when multiplying by five. Standard algorithms can be altered to suit the requirements of the current problem.

We shall examine standard algorithms for loading and printing arrays, adding the contents of an array of numbers, and processing records from sequential files. In the HSC course, we examine further algorithms that assist us to search and sort data.

### Loading and printing arrays

Loading an array means storing data items in each element of the array. Printing an array is the process of displaying each data item stored in the array. Both these processes are similar, as each and every element of the array must be accessed.

Loading an array (see *Fig 4.64*) could be done using an assignment statement for each element. This may be acceptable when the array contains a small number of elements, however it becomes tedious once the number of elements exceeds ten or so and becomes extremely cumbersome when hundreds or even thousands of elements are involved. A more general solution is required.

```
BEGIN LoadArray
  Get Item(0)
  Get Item(1)
  Get Item(2)
  Get Item(3)
  Get Item(4)
  Get Item(5)
  Get Item(6)
  Get Item(7)
  Get Item(8)
  Get Item(9)
END LoadArray
```

*Fig 4.64*
*Tedious algorithm to load data into an array.*

As each element in the array is normally filled one after the other we can use a variable that increments as the index. An iteration control structure surrounding this statement causes each element of the array to be accessed sequentially. Various possibilities exist for the termination condition for the loop. If the number of data items is known precisely then a counting loop could be used. If not then a sentinel value (say "ZZZ" or 9999 could be used as an indicator that there is no more data to be loaded (see *Fig 4.65*).

```
BEGIN LoadArray
  Set Index to 0
  Get DataItem
  WHILE DataItem is not the sentinel
    Store DataItem in Item(Index)
    Increment Index
    Get DataItem
  ENDWHILE
END LoadArray
```

*Fig 4.65*
*Algorithm to load data into an array.*

Let us consider the algorithm in *Fig 4.65* more closely. We first set the index for the array to its starting value. In most cases arrays are indexed from 0, as is the case in our pseudocode. We then get the first data item from the user and temporarily store it in the variable DataItem. We then check the loop's termination condition before entering the body of the loop. If the initial data item was the sentinel value, we would never enter or execute the body of the loop. If we had chosen a post-test iteration structure we would be forced to process the first data item, in other words the sentinel value would always be stored in the array. This may or may not be desirable depending on the problem. The body of the loop stores the data item in the array element Item(Index). Index is then incremented, meaning 1 is added to the

existing value of Index. So if we have just loaded 'sausage' into Item(5) then Index contains the value 5 and Item(5) contains the string 'sausage'. When Index is incremented its value now becomes 6 in preparation for the next data item to be loaded. This process continues until the sentinel value is encountered.

Note that the value of Index is incremented after each array element is loaded. As a consequence, the algorithm ends with Index equal to the next vacant array element. It may seem appropriate to reduce this value by 1. However as the array is indexed from 0, this value actually reflects the precise number of elements loaded into the array, which is often useful. It is also quite common to append further elements to the end of an array in which case, the value of Index is best left as is.


**GROUP TASK Discussion**

The algorithm described in *Fig 4.65* uses two get statements. Can you rewrite the algorithm using a single get? The result of the processing must be identical to the original in all circumstances.


**GROUP TASK Activity**

Rewrite the algorithm in *Fig 4.65* using a post-test iteration control structure. Ensure that the sentinel value is read into the last array element.

Printing or displaying the contents of an array is a similar process to loading an array. Rather than reading or getting data, we are printing or displaying the data. If we know the precise number of items in the array then the process is further simplified (see *Fig 4.66*). If a sentinel value is stored in the array to indicate the end of the data items then we must be careful not to print or display the sentinel (see *Fig 4.67*).

```
BEGIN PrintArray
  Set Index to 0
  WHILE Index < NumItems
    Display Item(Index)
    Increment Index
  ENDWHILE
END PrintArray
```

*Fig 4.66*
*Algorithm to print or display the data in an array when the number of items is known.*

```
BEGIN PrintArray
  Set Index to 0
  WHILE Item(Index) is not the sentinel
    Display Item(Index)
    Increment Index
  ENDWHILE
END PrintArray
```

*Fig 4.67*
*Algorithm to print or display the data in an array when the last item is a sentinel.*


**GROUP TASK Discussion**

In the above algorithms we displayed the array elements directly whereas when loading the array we read each data item into a temporary variable first and later stored it in the array. Why did we do this? Is it always necessary to use a temporary variable when loading an array? Discuss.


**GROUP TASK Activity**

Rewrite the above algorithms as equivalent flowcharts. Now redraw your flowcharts using post-test loops.

### Add the contents of an array of numbers

Summing each of the elements in an array of numbers is a common task. For example, finding the total sales each month, calculating averages or determining the number of website hits. The general idea is to iterate through the entire array of data items. During each iteration we add the value of the current array item to the total. We must be careful to avoid any array elements that do not contain data such as sentinel values or extra array elements that have not been loaded with data. It is important to ensure the algorithm works as expected when the array is empty or only contains the sentinel value.

```
BEGIN SumArrayContents
  Index = 0
  Total = 0
  WHILE Index < NumItems
    Total = Total + Item(Index)
    Index = Index + 1
  ENDWHILE
  Display "Sum = " Total
END SumArrayContents
```

*Fig 4.68*
*Algorithm to sum the contents of an array of numbers when the number of items is known.*

```
BEGIN SumArrayContents
  Index = 0
  Total = 0
  WHILE Item(Index) is not the sentinel
    Total = Total + Item(Index)
    Index = Index + 1
  ENDWHILE
  Display "Sum = " Total
END SumArrayContents
```

*Fig 4.69*
*Algorithm to sum the contents of an array of numbers when the last item is a sentinel.*


**GROUP TASK Discussion**

Explain what occurs in the above algorithms if the array does not contain any data. Can you write algorithms using either post-test or FOR...NEXT that result in the same output. Discuss.

### Processing using sequential files

Earlier in this chapter we discussed sequential files as data structures. These files contain sequences of characters, including control characters. To read a particular character requires reading all preceding characters. Normally all data is stored in the file as a series of tightly packed characters. For example, the integer -34 would be stored as the three characters -, 3 and 4. Most languages include predefined statements that can simplify reading and writing to files. Commonly, these statements allow us to read and write single characters, fields or complete lines of text.

To use a sequential file to store records requires software developers to design the precise format of the file so each record can be stored logically. Two techniques are commonly used. Either use separators between fields and records or specify the precise length of each field and record. The second technique is used by most programming languages when using random access files; we discuss random access files in the HSC course. The first technique is common when using sequential files and is therefore the technique we shall examine in some detail. As we are dealing with records that are comprised of fields, we would use a statement that reads and writes fields rather than characters or lines of text.

Normally a specific character is used to separate fields. This separator is often called a delimiter. The most common characters used are commas or tabs

```
Smith→John→8.35→29→4¶
Graham→Mary→8.95→31→1¶
Watson→Freda→7.25→19→0¶
Wilson→Martin→10.5→35→2¶
Thomson→John→9.15→30→4¶
Gardner→Jill→8.5→36→10¶
Smith→Margaret→5.35→17→0¶
Milton→Max→10.35→34→6¶
```

(→ tab) (¶ carriage return)

*Fig 4.70*
*A typical sequential file of records as it would appear when opened in a word processor.*

(ASCII code 9). Similarly, a character is used to separate each record. Normally a carriage return (ASCII code 13) or carriage return and linefeed are used. The use of tabs (or commas) and carriage returns as delimiters, allow us to view files using a simple text or word processor and have them appear formatted. Most programming languages include an end of file (EOF) function that can be used to ensure we do not read past the end of a file. It is possible to include a sentinel record that uses dummy values at the end of the file. A check for the sentinel value is used to prevent reading past the end of the file.

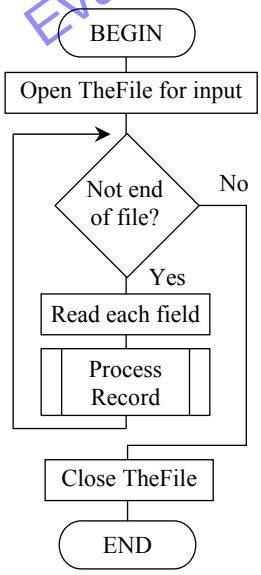



*Fig 4.71*
*Reading a sequential file of records.*

Before reading the contents of a file we must open the file. When reading a file we are inputting data from the file, hence we open the file for input. When writing to a file the file is opened for output. We then loop around once for each record until we reach the end of the file. The body of the loop reads each field in turn and performs the required processing on the record. Once we have completed our reading the file is closed. *Fig 4.71* describes an algorithm that performs this processing. For particular problems the 'Read each field' statement would contain a list of each of the fields into which the data is to be read. For example, if the file shown in *Fig 4.70* were used then the line may be:

```
Read Detail.Surname, Detail.CName, Detail.PayRate, Detail.Hrs, Detail.Otime from ThisFile
```

In this case a record has been declared with the identifier Detail. A dot is used to reference each field within the Detail record. The "Process Record" subroutine would likely calculate the pay due to each employee. This information could then be stored in a new sequential file.

Sequential files can also be used to store the elements in arrays. This allows the contents of an array to be saved and retrieved for later use.

```
BEGIN CreateFileFromArray
  Index = 0
  Open TheFile for output
  WHILE Index < NumItems
    Write TheFile from Item(Index)
    Index = Index + 1
  ENDWHILE
  Close TheFile
END CreateFileFromArray
```

*Fig 4.72*
*Algorithm to create a file from an array of data items.*

```
BEGIN ReadFileIntoArray
  Index = 0
  Open TheFile for input
  WHILE NOT EOF(TheFile)
    Read Item(Index) from TheFile
    Index = Index + 1
  ENDWHILE
  Close TheFile
END ReadFileIntoArray
```

*Fig 4.73*
*Algorithm to read a file of data items into an array.*


**GROUP TASK Activity**

Rewrite the algorithm in *Fig 4.71* so that it will in fact read the file in *Fig 4.70*. Also calculate each employee's pay and write each name and pay to a new sequential file. Assume overtime is paid at 1½ the normal pay rate.


**GROUP TASK Activity**

Create an algorithm that gets a sequence of numeric inputs from the user, writes them to a sequential file and then reads the file and outputs the total of the numbers.

## CHECKING ALGORITHMS FOR ERRORS

The main purpose of creating algorithms is to explain the logic of the solution. Therefore checking algorithms is primarily about checking the correctness of the logic. Checking doesn't just take place once an algorithm has been completed, rather it is an ongoing process occurring during the algorithm's development. The primary technique for checking algorithms is known as desk checking. As the name implies, a desk check is the process of working through an algorithm using pencil and paper. Test data where the expected outputs are known, provide the inputs for the desk check. In Chapter 6 we examine the creation of test data and desk checking in detail.

To be sure an algorithm performs correctly for all expected inputs requires that all paths through the algorithm be tested. Test data should be designed to accomplish this aim. This can often be a laborious task when there are many selection statements and loops. For example, an algorithm with 4 binary selections will have a total of 16 unique possible paths requiring up to 16 sets of test data and 16 desk checks. Many algorithms have far more unique paths than this. Iterations can also greatly increase the time required to desk check an algorithm. Most errors occur either as an iteration commences or as it terminates; be careful as loops start and end.

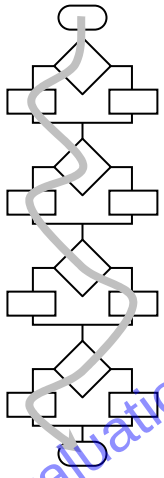


*Fig 4.74*
*One of sixteen unique paths through this algorithm.*

Another common source of logic errors is within decisions themselves. The use of incorrect logical operators being the most common e.g. using > instead of ≥. Be sure to test these boundary conditions carefully by including test data items that equal the boundary value.

Algorithms need not include detail in regard to validation of unexpected inputs, this is primarily a task undertaken when building the solution in code. It is not necessary to check algorithms respond correctly to unexpected inputs. For example, if you expect a particular input into an algorithm to be an integer then it is not necessary to check the algorithm works when strings or fractions are input.

Consider the following:

To illustrate the creation of test data and the desk checking process consider the algorithm in *Fig 4.75*. This algorithm is designed to find the average of a set of numbers. The process terminates when a negative number is entered.

We require at least two sets of test data; one to test the normal situation where the body of the loop is executed and another to test when the loop's termination condition is immediately false. We should also test using the boundary value zero; this could be included within the first set of test data. Say we use the test data set 34, 0, 65, 40, 85, -1 and the test data set containing just -1.

```
BEGIN MAINPROGRAM
  Input Number
  Sum = 0
  Tally = 0
  WHILE Number ≥ 0
    Sum = Sum + Number
    Tally = Tally + 1
    Input Number
  ENDWHILE
  Average = Sum/Tally
  Print Average
ENDMAINPROGRAM
```

*Fig 4.75*
*Algorithm to calculate the average of a set of numbers.*


**GROUP TASK Discussion**
Do you think the two test data sets are sufficient to ensure the correct operation of this algorithm? Discuss.

Before commencing the desk check we calculate the expected output from each test data set. Our first set should output the average of 34, 0, 65, 40 and 85. As 34 + 0 + 65 + 40 = 224, the average should be 224 ÷ 5 = 44.8. The second data set contains just a negative value so a sensible result would be an output of 0 or no output at all.

We now commence the desk check. A table is drawn by hand with a column for each variable used within the algorithm and an additional column for output (see *Fig 4.76*). The current contents of each variable are written under the appropriate identifier. As the contents of a variable changes the new data is written under the previous item. It is common practice to horizontally align data items assigned within each iteration.

| Number | Sum | Tally | Average | Output |
|---|---|---|---|---|
| 34 | 0 | 0 | | |
| 0 | 34 | 1 | | |
| 65 | 34 | 2 | | |
| 40 | 99 | 3 | | |
| 85 | 139 | 4 | | |
| -1 | 224 | 5 | 224/5 = 44.8 | 44.8 |
| -1 | 0 | 0 | % (Error) | |

*Fig 4.76*
*Desk check for the average algorithm.*

Essentially, the desk check is completed by stepping statement by statement through the algorithm. When an input is required the next test data item is used. The first test data set results in the same output as was expected. The second causes a division by zero error. We need to adjust our algorithm to overcome this problem. Once the algorithm has been edited both desk checks need to be completed again as it is possible that the alterations may solve one problem but cause other problems.


**GROUP TASK Activity**
Alter the algorithm to overcome the division by zero problem. Now perform the desk checks again using your altered algorithm.

Consider the following:

Earlier in this chapter, we considered an algorithm to determine if people are able to ride on a roller coaster. The algorithm is reproduced at right. We wish to thoroughly check the operation of this algorithm using sufficient sets of test data together with a series of desk checks.


**GROUP TASK Activity**
Design appropriate pairs of test data and state the expected output for each. Give reasons why each test data pair is included.


**GROUP TASK Activity**
Perform a desk check of the algorithm using each of your test data pairs.

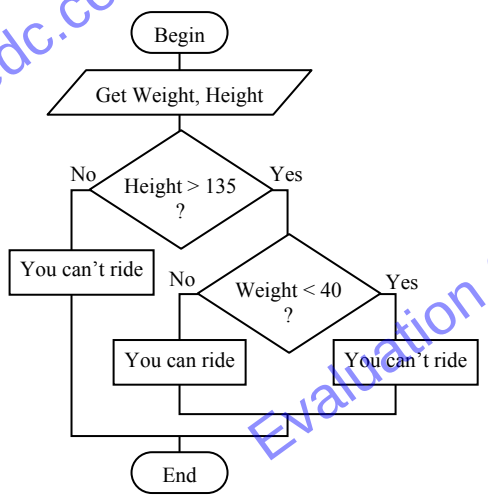



*Fig 4.77*
*Algorithm to determine if a person can ride the roller coaster.*

**SET 4E**

1. In terms of software structure, what provides the interface between different subroutines?
   (A) Indexes.
   (B) Parameters.
   (C) Subroutines.
   (D) algorithms.

2. A primary technique for checking algorithms is known as:
   (A) desk checking.
   (B) peer checking.
   (C) black box checking.
   (D) white box checking.

3. The process of communicating via parameters is known as:
   (A) parsing.
   (B) swapping.
   (C) passing.
   (D) none of the above.

4. The ability to reuse modules is best described as:
   (A) black boxing.
   (B) modularity.
   (C) abstraction.
   (D) top-down design.

5. In regards to a sequential file, what is the term given to the character that is used to separate fields?
   (A) Delimiter.
   (B) Sentinel value.
   (C) Comma.
   (D) Tab.

6. In pseudocode, how is a call to a subroutine indicated?
   (A) The process is in capital letters.
   (B) The process is italicised.
   (C) The process is underlined.
   (D) The process is preceded by a # symbol.

7. In most programming languages, what is used to ensure reading does not continue past the end of the file?
   (A) A delimiter.
   (B) Sentinel value.
   (C) EOF function.
   (D) Carriage return.

8. Subprogram has a similar meaning to:
   (A) lower level process.
   (B) module.
   (C) subroutine.
   (D) all of the above.

9. In both pseudocode and flowcharts, parameters are represented by:
   (A) a bracketed list.
   (B) an underlined list.
   (C) an italicised list.
   (D) capital letters.

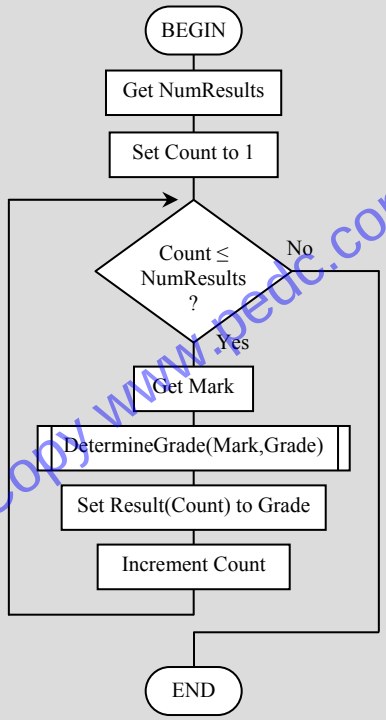
10. An algorithm with 5 binary selections could have as many as _____ unique possible paths?
    (A) 5.
    (B) 16.
    (C) 54.
    (D) 32.

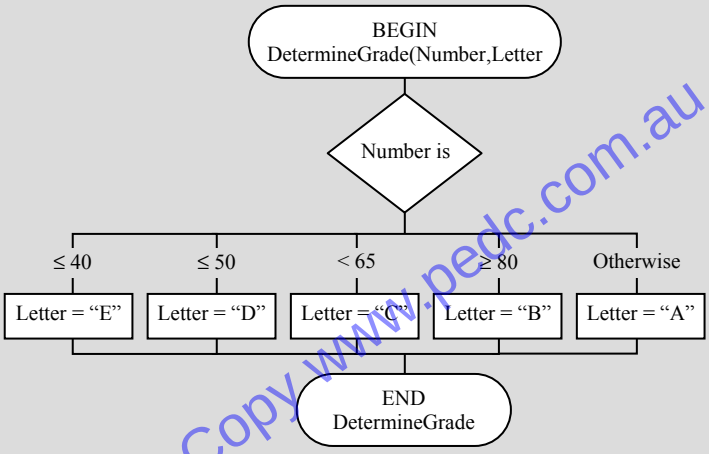
11. Problems can be solved without using subroutines. So, why do we use them? Discuss.

12. Discuss aspects of designs that increase the reusability of code modules.

13. Parameters provide the interface between different modules. What is a parameter and how do they enable data to be shared?

Consider the following flowchart when answering questions 14 and 15.




14. Design a set of test data for the above flowchart.

15. Perform a desk check of the flowchart using your test data.


HSC style question:

RSA security is a company that produces encryption and decryption software products. RSA's encryption is based on pairs of prime numbers. A prime number is an integer that only has 1 and itself as factors, for example 7 is a prime number as its only pair of factors is 1 and 7. However 8 is not a prime, as 2 and 4 are factors in addition to 1 and 8.

To monitor the security of RSA's encryption techniques they run a number of competitions. The data reproduced below is from their RSA-640 factoring competition. The aim is to determine the two prime numbers whose product is the competition number shown below. This competition number contains 193 decimal digits and the sum of these digits is 806. Furthermore if this number were converted to binary it would contain 640 binary digits, hence the competition name RSA-640. There are numerous competitions, the largest being RSA-2048 where the prize money is $200,000.

Name: RSA-640
Prize: $20,000
Digits: 193
Digit Sum: 806
Competition number:
3107418240490043721350750035888567930037346022842727545720161948823206440518081504556346829671723286782437916272838033415471073108501919548529007337724822783525742386454014691736602477652346609

The following algorithm is an attempt to solve the above problem:

```
BEGIN RSA_Solution
    Get CompNum
    CheckNumDigits
    CheckDigitSum
    PossibleFactor = 2
    REPEAT
        IsPrime = CheckIfPrime(PossibleFactor)
        IF IsPrime THEN
            OtherFactor = CompNum/PossibleFactor
            IF OtherFactor is an Integer THEN
                IsPrime = CheckIfPrime(OtherFactor)
                IF IsPrime THEN
                    Display PossibleFactor, OtherFactor
                ENDIF
            ENDIF
        ENDIF
        Increment PossibleFactor
    UNTIL IsPrime
END RSA_Solution
```

(a) The above algorithm checks if OtherFactor is an integer, however there is no check to ensure PossibleFactor is an integer. Why is this? Discuss.

(b) When building this solution a problem will emerge in regard to representing some of the variables using standard data types. Identify and describe the problem.

(c) Perform a desk check of the algorithm using 18 as the input. You may assume all calls to subroutines are performed correctly.

(d) There is a logic error within the algorithm. Identify the error and describe its effect on the operation of the algorithm.

(e) Design an algorithm for the CheckIfPrime routine. You may assume that the variables PossibleFactor and OtherFactor are integers.

**Suggested solutions**

(a) PossibleFactor must be an integer. It starts with a value of 2 and is incremented each time through the loop, hence it remains an integer. OtherFactor is the result of a division and hence will rarely be an integer, hence a check is needed.

(b) The competition number is well outside the range of any standard data type. If CompNum was a long integer it could not hold the competition number, in fact a 640 bit integer type is needed not a 32 bit integer type. A floating point type is not appropriate as the representation must be exact and so too must calculations based on this representation. Similar problems would occur with PossibleFactor and OtherFactor.

(c)

| CompNum | PossibleFactor | IsPrime | OtherFactor |
|---|---|---|---|
| 18 | 2 | T | 9 |
| | | F | |
| | 3 | T | 6 |
| | | F | |
| | 4 | F | |
| | 5 | T | 3.6 |
| | 6 | | |

(d) The use of IsPrime for the PossibleFactor and OtherFactor prime checks, as well as for the loop termination condition cause problems. This means that if PossibleFactor is a prime and OtherFactor is not an integer then the loop will exit. Using the RSA-640 competition number the loop would end after a single iteration as PossibleFactor is a prime (2) and OtherFactor will end in .5. (2 marks)

(e)
```
BEGIN CheckIfPrime (Num)
    Check = True
    Count = 2
    WHILE Count <= square root of Num AND Check
        IF Num/Count is an integer THEN
            Check = False
        ENDIF
        Increment Count
    ENDWHILE
    RETURN Check
END
```

## CHAPTER 4 REVIEW

1. Selecting data types and data structures generally falls under which phase of the software development cycle?
   (A) Defining the problem.
   (B) Planning the solution.
   (C) Building the solution.
   (D) Checking the solution.

2. Floating point data types are used to store which type of data?
   (A) Strings.
   (B) Whole numbers.
   (C) Fractional and very large numbers.
   (D) Integers.

3. In flowcharts, rectangles are used for representing:
   (A) processes.
   (B) decisions.
   (C) input and output.
   (D) subroutines.

4. A step-by-step list of the processing that will take place is usually represented with what?
   (A) A structure chart.
   (B) An IPO chart.
   (C) A context diagram.
   (D) A data flow diagram.

5. A structured algorithm can BEST be described as:
   (A) a method that provides a solution to a problem.
   (B) a method detailing a series of unambiguous steps that provides a solution to a problem.
   (C) a sequence of steps that transforms inputs into outputs.
   (D) any method that gives the correct result.

6. A Boolean data type is used to store which type of data?
   (A) Dates and Times.
   (B) Currency.
   (C) Logical.
   (D) Integer.

7. The control structure that repeats sequences of code is known as what?
   (A) Iteration.
   (B) Sequence.
   (C) Selection.
   (D) Subroutines.

8. Two modelling techniques that are used to describe the flow of data moving through the system are:
   (A) Context diagrams and algorithms.
   (B) Data flow diagrams and system flowcharts.
   (C) Hierarchy charts and IPO charts.
   (D) Abstraction and refinement.

9. Coding in a programming language occurs during which phase of the software development cycle?
   (A) Testing and evaluating
   (B) Maintaining
   (C) Implementing
   (D) Planning and designing

10. Which control structure caters for situations where more than two alternative paths are required?
    (A) Repetition
    (B) Binary selection
    (C) Multiway selection
    (D) Sequence

11. Different data types have different limitations. Describe limitations that exist when using integer, floating point and string data types.

12. In this chapter, we discussed three data structures, namely arrays, records and sequential files. Briefly, describe the nature of each of these structures together with an example of where each would be used.

Consider the algorithm that follows, when answering questions 13, 14 and 15.

```
BEGIN ChangeArray (Item( ), NumItems)
      Get DateItem
      Get Command
      CASEWHERE Command is
          Add      : AppendItem (Item( ), NumItems, DateItem)
          Delete   : RemoveItem (Item( ), NumItems, DataItem)
          Change : AlterItem (Item ( ), NumItems, DataItem)
      ENDCASE
END ChangeArray

BEGIN AppendItem (Array( ), Count, Item)
      Add 1 to Count
      Set Array(Count) to Item
END AppendItem

BEGIN RemoveItem (Array( ), NumItems, DelItem)
      FindIndex (Array( ), NumItems, DelItem, Index)
      IF Index ≥ 0 THEN
          Set Array(Index) to Array(NumItems)
          Decrement NumItems
      ELSE
          Display "Can't delete as item does not exist"
      ENDIF
END Remove Item

 BEGIN AlterItem (Array( ), Count, Original)
       FindIndex (Array( ), Count, Original, Index)
       IF Index ≥ 0 THEN
           Get ChangeItem
           Set Array(Index) to ChangeItem
       ELSE
           Display "Can't change as item does not exist"
       ENDIF
  END AlterItem
```

13. Construct a structure chart to describe the top-down design of the ChangeArray sub-program.

14. There is no algorithm for the FindIndex subroutine. Create this algorithm in pseudocode.

15. Perform a desk check of the algorithm using the following sets of test data:

    Desk, Delete  Table, Change, Door  Goose, Add

    Duck, Delete  Goose, Change, Bird

    (Assume the initial array is indexed from 0 and contains the items Desk, Table, Chair, Plate and Cutlery). Describe any problems you encounter.